

LS-CAT PGPMAC

Generated by Doxygen 1.6.1

Fri Oct 12 13:36:18 2012

Contents

1	The LS-CAT pgpmac Project	1
2	Data Structure Index	5
2.1	Data Structures	5
3	File Index	7
3.1	File List	7
4	Data Structure Documentation	9
4.1	lspg_lock_detector_struct Struct Reference	9
4.1.1	Detailed Description	9
4.1.2	Field Documentation	9
4.1.2.1	cond	9
4.1.2.2	mutex	9
4.1.2.3	new_value_ready	9
4.2	lspg_lock_diffractionmeter_struct Struct Reference	10
4.2.1	Detailed Description	10
4.2.2	Field Documentation	10
4.2.2.1	cond	10
4.2.2.2	mutex	10
4.2.2.3	new_value_ready	10
4.3	lspg_nextshot_struct Struct Reference	11
4.3.1	Detailed Description	14
4.3.2	Field Documentation	14
4.3.2.1	active	14
4.3.2.2	active2	14
4.3.2.3	active2_isnull	15
4.3.2.4	active_isnull	15
4.3.2.5	ax	15

4.3.2.6	ax2	15
4.3.2.7	ax2_isnull	15
4.3.2.8	ax_isnull	15
4.3.2.9	ay	15
4.3.2.10	ay2	15
4.3.2.11	ay2_isnull	15
4.3.2.12	ay_isnull	15
4.3.2.13	az	16
4.3.2.14	az2	16
4.3.2.15	az2_isnull	16
4.3.2.16	az_isnull	16
4.3.2.17	cond	16
4.3.2.18	cx	16
4.3.2.19	cx2	16
4.3.2.20	cx2_isnull	16
4.3.2.21	cx_isnull	16
4.3.2.22	cy	16
4.3.2.23	cy2	17
4.3.2.24	cy2_isnull	17
4.3.2.25	cy_isnull	17
4.3.2.26	dsdir	17
4.3.2.27	dsdir_isnull	17
4.3.2.28	dsdist	17
4.3.2.29	dsdist2	17
4.3.2.30	dsdist2_isnull	17
4.3.2.31	dsdist_isnull	17
4.3.2.32	dsexp	17
4.3.2.33	dsexp2	18
4.3.2.34	dsexp2_isnull	18
4.3.2.35	dsexp_isnull	18
4.3.2.36	dshpid	18
4.3.2.37	dshpid_isnull	18
4.3.2.38	dskappa	18
4.3.2.39	dskappa2	18
4.3.2.40	dskappa2_isnull	18
4.3.2.41	dskappa_isnull	18

4.3.2.42	dsnrg	18
4.3.2.43	dsnrg2	19
4.3.2.44	dsnrg2_isnull	19
4.3.2.45	dsnrg_isnull	19
4.3.2.46	dsomega	19
4.3.2.47	dsomega2	19
4.3.2.48	dsomega2_isnull	19
4.3.2.49	dsomega_isnull	19
4.3.2.50	dsoscaxis	19
4.3.2.51	dsoscaxis2	19
4.3.2.52	dsoscaxis2_isnull	19
4.3.2.53	dsoscaxis_isnull	20
4.3.2.54	dsowidth	20
4.3.2.55	dsowidth2	20
4.3.2.56	dsowidth2_isnull	20
4.3.2.57	dsowidth_isnull	20
4.3.2.58	dsphi	20
4.3.2.59	dsphi2	20
4.3.2.60	dsphi2_isnull	20
4.3.2.61	dsphi_isnull	20
4.3.2.62	dspid	20
4.3.2.63	dspid_isnull	21
4.3.2.64	mutex	21
4.3.2.65	new_value_ready	21
4.3.2.66	no_rows_returned	21
4.3.2.67	sfn	21
4.3.2.68	sfn_isnull	21
4.3.2.69	sindex	21
4.3.2.70	sindex2	21
4.3.2.71	sindex2_isnull	21
4.3.2.72	sindex_isnull	21
4.3.2.73	skey	22
4.3.2.74	skey_isnull	22
4.3.2.75	sstart	22
4.3.2.76	sstart2	22
4.3.2.77	sstart2_isnull	22

4.3.2.78	sstart_isnull	22
4.3.2.79	stype	22
4.3.2.80	stype2	22
4.3.2.81	stype2_isnull	22
4.3.2.82	stype_isnull	22
4.4	lspg_seq_run_prep_struct Struct Reference	24
4.4.1	Detailed Description	24
4.4.2	Field Documentation	24
4.4.2.1	cond	24
4.4.2.2	mutex	24
4.4.2.3	new_value_ready	24
4.5	lspg_wait_for_detector_struct Struct Reference	25
4.5.1	Detailed Description	25
4.5.2	Field Documentation	25
4.5.2.1	cond	25
4.5.2.2	mutex	25
4.5.2.3	new_value_ready	25
4.6	lspgQueryQueueStruct Struct Reference	26
4.6.1	Detailed Description	26
4.6.2	Field Documentation	26
4.6.2.1	onResponse	26
4.6.2.2	qs	26
4.7	lspmac_cmd_queue_struct Struct Reference	27
4.7.1	Detailed Description	27
4.7.2	Field Documentation	27
4.7.2.1	no_reply	27
4.7.2.2	onResponse	27
4.7.2.3	pcmd	27
4.7.2.4	rbuff	28
4.7.2.5	time_sent	28
4.8	lspmac_motor_struct Struct Reference	29
4.8.1	Detailed Description	30
4.8.2	Field Documentation	31
4.8.2.1	actual_pos_cnts_p	31
4.8.2.2	cond	31
4.8.2.3	dac_mvar	31

4.8.2.4	format	31
4.8.2.5	lut	31
4.8.2.6	max_accel	31
4.8.2.7	max_speed	31
4.8.2.8	motion_seen	31
4.8.2.9	motor_num	31
4.8.2.10	moveAbs	32
4.8.2.11	mutex	32
4.8.2.12	name	32
4.8.2.13	nlut	32
4.8.2.14	not_done	32
4.8.2.15	position	32
4.8.2.16	pq	32
4.8.2.17	read	32
4.8.2.18	read_mask	32
4.8.2.19	read_ptr	33
4.8.2.20	reported_position	33
4.8.2.21	requested_pos_cnts	33
4.8.2.22	requested_position	33
4.8.2.23	status1	33
4.8.2.24	status2	33
4.8.2.25	u2c	33
4.8.2.26	units	33
4.8.2.27	update_resolution	33
4.8.2.28	win	34
4.8.2.29	write_fmt	34
4.9	md2StatusStruct Struct Reference	35
4.9.1	Detailed Description	36
4.9.2	Field Documentation	36
4.9.2.1	acc11c_1	36
4.9.2.2	acc11c_2	36
4.9.2.3	acc11c_3	36
4.9.2.4	acc11c_5	37
4.9.2.5	acc11c_6	37
4.9.2.6	alignx_act_pos	37
4.9.2.7	alignx_status_1	37

4.9.2.8	alignx_status_2	37
4.9.2.9	aligny_act_pos	37
4.9.2.10	aligny_status_1	37
4.9.2.11	aligny_status_2	37
4.9.2.12	alignz_act_pos	37
4.9.2.13	alignz_status_1	37
4.9.2.14	alignz_status_2	37
4.9.2.15	analyzer_act_pos	38
4.9.2.16	analyzer_status_1	38
4.9.2.17	analyzer_status_2	38
4.9.2.18	aperturey_act_pos	38
4.9.2.19	aperturey_status_1	38
4.9.2.20	aperturey_status_2	38
4.9.2.21	aperturez_act_pos	38
4.9.2.22	aperturez_status_1	38
4.9.2.23	aperturez_status_2	38
4.9.2.24	back_dac	38
4.9.2.25	capy_act_pos	38
4.9.2.26	capy_status_1	39
4.9.2.27	capy_status_2	39
4.9.2.28	capz_act_pos	39
4.9.2.29	capz_status_1	39
4.9.2.30	capz_status_2	39
4.9.2.31	centerx_act_pos	39
4.9.2.32	centerx_status_1	39
4.9.2.33	centerx_status_2	39
4.9.2.34	centery_act_pos	39
4.9.2.35	centery_status_1	39
4.9.2.36	centery_status_2	39
4.9.2.37	dummy1	40
4.9.2.38	dummy2	40
4.9.2.39	dummy3	40
4.9.2.40	dummy4	40
4.9.2.41	dummy5	40
4.9.2.42	dummy6	40
4.9.2.43	dummy7	40

4.9.2.44	dummy8	40
4.9.2.45	dummy9	40
4.9.2.46	dummyA	40
4.9.2.47	dummyB	40
4.9.2.48	front_dac	41
4.9.2.49	fs_has_opened	41
4.9.2.50	fs_has_opened_globally	41
4.9.2.51	fs_is_open	41
4.9.2.52	kappa_act_pos	41
4.9.2.53	kappa_status_1	41
4.9.2.54	kappa_status_2	41
4.9.2.55	moving_flags	41
4.9.2.56	number_passes	41
4.9.2.57	omega_act_pos	41
4.9.2.58	omega_status_1	41
4.9.2.59	omega_status_2	42
4.9.2.60	phi_act_pos	42
4.9.2.61	phi_status_1	42
4.9.2.62	phi_status_2	42
4.9.2.63	phiscan	42
4.9.2.64	scint_act_pos	42
4.9.2.65	scint_piezo	42
4.9.2.66	scint_status_1	42
4.9.2.67	scint_status_2	42
4.9.2.68	zoom_act_pos	42
4.9.2.69	zoom_status_1	42
4.9.2.70	zoom_status_2	43
4.10	tagEthernetCmd Struct Reference	44
4.10.1	Detailed Description	44
4.10.2	Field Documentation	44
4.10.2.1	bData	44
4.10.2.2	Request	44
4.10.2.3	RequestType	44
4.10.2.4	wIndex	45
4.10.2.5	wLength	45
4.10.2.6	wValue	45

5	File Documentation	47
5.1	lspg.c File Reference	47
5.1.1	Detailed Description	52
5.1.2	Define Documentation	53
5.1.2.1	LS_PG_QUERY_QUEUE_LENGTH	53
5.1.2.2	LS_PG_STATE_IDLE	53
5.1.2.3	LS_PG_STATE_INIT	53
5.1.2.4	LS_PG_STATE_INIT_POLL	53
5.1.2.5	LS_PG_STATE_RECV	53
5.1.2.6	LS_PG_STATE_RESET	53
5.1.2.7	LS_PG_STATE_RESET_POLL	53
5.1.2.8	LS_PG_STATE_SEND	53
5.1.2.9	LS_PG_STATE_SEND_FLUSH	53
5.1.3	Typedef Documentation	53
5.1.3.1	lspg_lock_detector_t	53
5.1.3.2	lspg_lock_diffractionmeter_t	54
5.1.3.3	lspg_query_queue_t	54
5.1.3.4	lspg_seq_run_prep_t	54
5.1.3.5	lspg_wait_for_detector_t	54
5.1.4	Function Documentation	54
5.1.4.1	lspg_blight_lut_cb	54
5.1.4.2	lspg_cmd_cb	55
5.1.4.3	lspg_flight_lut_cb	55
5.1.4.4	lspg_flush	56
5.1.4.5	lspg_getcenter_cb	56
5.1.4.6	lspg_init	57
5.1.4.7	lspg_init_motors_cb	57
5.1.4.8	lspg_lock_detector_all	58
5.1.4.9	lspg_lock_detector_call	58
5.1.4.10	lspg_lock_detector_cb	58
5.1.4.11	lspg_lock_detector_done	58
5.1.4.12	lspg_lock_detector_init	59
5.1.4.13	lspg_lock_detector_wait	59
5.1.4.14	lspg_lock_diffractionmeter_all	59
5.1.4.15	lspg_lock_diffractionmeter_call	59
5.1.4.16	lspg_lock_diffractionmeter_cb	60

5.1.4.17	<code>lspg_lock_diffractionmeter_done</code>	60
5.1.4.18	<code>lspg_lock_diffractionmeter_init</code>	60
5.1.4.19	<code>lspg_lock_diffractionmeter_wait</code>	60
5.1.4.20	<code>lspg_next_state</code>	60
5.1.4.21	<code>lspg_nextaction_cb</code>	61
5.1.4.22	<code>lspg_nextshot_call</code>	62
5.1.4.23	<code>lspg_nextshot_cb</code>	62
5.1.4.24	<code>lspg_nextshot_done</code>	66
5.1.4.25	<code>lspg_nextshot_init</code>	66
5.1.4.26	<code>lspg_nextshot_wait</code>	67
5.1.4.27	<code>lspg_pg_connect</code>	67
5.1.4.28	<code>lspg_pg_service</code>	68
5.1.4.29	<code>lspg_query_next</code>	70
5.1.4.30	<code>lspg_query_push</code>	70
5.1.4.31	<code>lspg_query_reply_next</code>	71
5.1.4.32	<code>lspg_query_reply_peek</code>	71
5.1.4.33	<code>lspg_receive</code>	72
5.1.4.34	<code>lspg_run</code>	73
5.1.4.35	<code>lspg_send_next_query</code>	73
5.1.4.36	<code>lspg_seq_run_prep_all</code>	74
5.1.4.37	<code>lspg_seq_run_prep_call</code>	75
5.1.4.38	<code>lspg_seq_run_prep_cb</code>	75
5.1.4.39	<code>lspg_seq_run_prep_done</code>	76
5.1.4.40	<code>lspg_seq_run_prep_init</code>	76
5.1.4.41	<code>lspg_seq_run_prep_wait</code>	76
5.1.4.42	<code>lspg_sig_service</code>	76
5.1.4.43	<code>lspg_wait_for_detector_all</code>	77
5.1.4.44	<code>lspg_wait_for_detector_call</code>	77
5.1.4.45	<code>lspg_wait_for_detector_cb</code>	77
5.1.4.46	<code>lspg_wait_for_detector_done</code>	77
5.1.4.47	<code>lspg_wait_for_detector_init</code>	78
5.1.4.48	<code>lspg_wait_for_detector_wait</code>	78
5.1.4.49	<code>lspg_worker</code>	78
5.1.4.50	<code>lspg_zoom_lut_cb</code>	79
5.1.5	Variable Documentation	80
5.1.5.1	<code>ls_pg_state</code>	80

5.1.5.2	lspg_connectPoll_response	80
5.1.5.3	lspg_lock_detector	80
5.1.5.4	lspg_lock_diffractionmeter	80
5.1.5.5	lspg_nextshot	80
5.1.5.6	lspg_query_queue	81
5.1.5.7	lspg_query_queue_off	81
5.1.5.8	lspg_query_queue_on	81
5.1.5.9	lspg_query_queue_reply	81
5.1.5.10	lspg_resetPoll_response	81
5.1.5.11	lspg_seq_run_prep	81
5.1.5.12	lspg_thread	81
5.1.5.13	lspg_wait_for_detector	81
5.1.5.14	lspgfd	81
5.1.5.15	pg_queue_mutex	82
5.1.5.16	q	82
5.2	lspmac.c File Reference	83
5.2.1	Detailed Description	90
5.2.2	Define Documentation	91
5.2.2.1	LS_PMAC_STATE_CR	91
5.2.2.2	LS_PMAC_STATE_DETACHED	91
5.2.2.3	LS_PMAC_STATE_GB	91
5.2.2.4	LS_PMAC_STATE_GMR	91
5.2.2.5	LS_PMAC_STATE_IDLE	91
5.2.2.6	LS_PMAC_STATE_RESET	91
5.2.2.7	LS_PMAC_STATE_RR	91
5.2.2.8	LS_PMAC_STATE_SC	91
5.2.2.9	LS_PMAC_STATE_WACK	91
5.2.2.10	LS_PMAC_STATE_WACK_CC	91
5.2.2.11	LS_PMAC_STATE_WACK_NFR	92
5.2.2.12	LS_PMAC_STATE_WACK_RR	92
5.2.2.13	LS_PMAC_STATE_WCR	92
5.2.2.14	LS_PMAC_STATE_WGB	92
5.2.2.15	PMAC_CMD_QUEUE_LENGTH	92
5.2.2.16	pmac_cmd_size	92
5.2.2.17	PMAC_MIN_CMD_TIME	92
5.2.2.18	PMACPORT	92

5.2.2.19	VR_CTRL_RESPONSE	92
5.2.2.20	VR_DOWNLOAD	92
5.2.2.21	VR_FWDOWNLOAD	93
5.2.2.22	VR_IPADDRESS	93
5.2.2.23	VR_PMAC_FLUSH	93
5.2.2.24	VR_PMAC_GETBUFFER	93
5.2.2.25	VR_PMAC_GETLINE	93
5.2.2.26	VR_PMAC_GETMEM	93
5.2.2.27	VR_PMAC_GETRESPONSE	93
5.2.2.28	VR_PMAC_PORT	93
5.2.2.29	VR_PMAC_READREADY	93
5.2.2.30	VR_PMAC_SENDCTRLCHAR	93
5.2.2.31	VR_PMAC_SENDLINE	93
5.2.2.32	VR_PMAC_SETBIT	94
5.2.2.33	VR_PMAC_SETBITS	94
5.2.2.34	VR_PMAC_SETMEM	94
5.2.2.35	VR_PMAC_WRITEBUFFER	94
5.2.2.36	VR_PMAC_WRITEERROR	94
5.2.2.37	VR_UPLOAD	94
5.2.3	Typedef Documentation	94
5.2.3.1	md2_status_t	94
5.2.4	Function Documentation	94
5.2.4.1	cleanstr	94
5.2.4.2	hex_dump	95
5.2.4.3	if	95
5.2.4.4	if	95
5.2.4.5	if	96
5.2.4.6	lsConnect	96
5.2.4.7	lspmac_bio_init	97
5.2.4.8	lspmac_bio_read	98
5.2.4.9	lspmac_dac_init	98
5.2.4.10	lspmac_dac_read	99
5.2.4.11	lspmac_Error	99
5.2.4.12	lspmac_fshut_init	100
5.2.4.13	lspmac_get_status	100
5.2.4.14	lspmac_get_status_cb	100

5.2.4.15	<code>lspmac_GetAllIVars</code>	102
5.2.4.16	<code>lspmac_GetAllIVarsCB</code>	103
5.2.4.17	<code>lspmac_GetAllMVars</code>	103
5.2.4.18	<code>lspmac_GetAllMVarsCB</code>	103
5.2.4.19	<code>lspmac_Getmem</code>	104
5.2.4.20	<code>lspmac_GetmemReplyCB</code>	104
5.2.4.21	<code>lspmac_GetShortReplyCB</code>	104
5.2.4.22	<code>lspmac_init</code>	105
5.2.4.23	<code>lspmac_lut</code>	107
5.2.4.24	<code>lspmac_motor_init</code>	108
5.2.4.25	<code>lspmac_moveabs_bio_queue</code>	109
5.2.4.26	<code>lspmac_moveabs_fshut_queue</code>	110
5.2.4.27	<code>lspmac_moveabs_queue</code>	110
5.2.4.28	<code>lspmac_moveabs_wait</code>	111
5.2.4.29	<code>lspmac_movedac_queue</code>	112
5.2.4.30	<code>lspmac_movezoom_queue</code>	112
5.2.4.31	<code>lspmac_next_state</code>	113
5.2.4.32	<code>lspmac_pmacmotor_read</code>	115
5.2.4.33	<code>lspmac_pop_queue</code>	116
5.2.4.34	<code>lspmac_pop_reply</code>	116
5.2.4.35	<code>lspmac_push_queue</code>	117
5.2.4.36	<code>lspmac_Reset</code>	117
5.2.4.37	<code>lspmac_run</code>	117
5.2.4.38	<code>lspmac_send_command</code>	118
5.2.4.39	<code>lspmac_sendcmd_noch</code>	119
5.2.4.40	<code>lspmac_SendControlReplyPrintCB</code>	119
5.2.4.41	<code>lspmac_Service</code>	120
5.2.4.42	<code>lspmac_shutter_read</code>	123
5.2.4.43	<code>lspmac_SockFlush</code>	123
5.2.4.44	<code>lspmac_SockGetmem</code>	123
5.2.4.45	<code>lspmac_SockSendControlCharPrint</code>	123
5.2.4.46	<code>lspmac_SockSendline</code>	124
5.2.4.47	<code>lspmac_SockSendline_nr</code>	124
5.2.4.48	<code>lspmac_worker</code>	124
5.2.5	Variable Documentation	125
5.2.5.1	<code>alignx</code>	125

5.2.5.2	aligny	125
5.2.5.3	alignz	125
5.2.5.4	anal	125
5.2.5.5	apery	125
5.2.5.6	aperz	126
5.2.5.7	blight	126
5.2.5.8	blight_ud	126
5.2.5.9	capy	126
5.2.5.10	capz	126
5.2.5.11	cenx	126
5.2.5.12	ceny	126
5.2.5.13	cr_cmd	126
5.2.5.14	dbmem	127
5.2.5.15	dbmemIn	127
5.2.5.16	ethCmdOff	127
5.2.5.17	ethCmdOn	127
5.2.5.18	ethCmdQueue	127
5.2.5.19	ethCmdReply	127
5.2.5.20	flight	127
5.2.5.21	fscint	127
5.2.5.22	fshut	127
5.2.5.23	gb_cmd	128
5.2.5.24	getivars	128
5.2.5.25	getmvars	128
5.2.5.26	kappa	128
5.2.5.27	linesReceived	128
5.2.5.28	ls_pmac_state	128
5.2.5.29	lspmac_motors	128
5.2.5.30	lspmac_moving_cond	128
5.2.5.31	lspmac_moving_flags	128
5.2.5.32	lspmac_moving_mutex	129
5.2.5.33	lspmac_nmotors	129
5.2.5.34	lspmac_shutter_cond	129
5.2.5.35	lspmac_shutter_has_opened	129
5.2.5.36	lspmac_shutter_mutex	129
5.2.5.37	lspmac_shutter_state	129

5.2.5.38	md2_status	129
5.2.5.39	md2_status_mutex	129
5.2.5.40	now	129
5.2.5.41	omega	130
5.2.5.42	phi	130
5.2.5.43	pmac_error_strs	130
5.2.5.44	pmac_queue_cond	130
5.2.5.45	pmac_queue_mutex	130
5.2.5.46	pmac_thread	131
5.2.5.47	pmacfd	131
5.2.5.48	rr_cmd	131
5.2.5.49	scinz	131
5.2.5.50	zoom	131
5.3	lsupdate.c File Reference	132
5.3.1	Detailed Description	132
5.3.2	Function Documentation	132
5.3.2.1	lsupdate_init	132
5.3.2.2	lsupdate_run	133
5.3.2.3	lsupdate_updateit	133
5.3.2.4	lsupdate_worker	134
5.3.3	Variable Documentation	134
5.3.3.1	lsupdate_thread	134
5.4	md2cmds.c File Reference	135
5.4.1	Detailed Description	136
5.4.2	Function Documentation	136
5.4.2.1	logtime	136
5.4.2.2	md2cmds_center	137
5.4.2.3	md2cmds_collect	137
5.4.2.4	md2cmds_init	140
5.4.2.5	md2cmds_moveAbs	140
5.4.2.6	md2cmds_mvcenter_move	142
5.4.2.7	md2cmds_mvcenter_prep	142
5.4.2.8	md2cmds_mvcenter_wait	143
5.4.2.9	md2cmds_rotate	143
5.4.2.10	md2cmds_run	143
5.4.2.11	md2cmds_transfer	144

5.4.2.12	md2cmds_worker	144
5.4.3	Variable Documentation	144
5.4.3.1	md2cmds_cmd	144
5.4.3.2	md2cmds_cond	145
5.4.3.3	md2cmds_mutex	145
5.4.3.4	md2cmds_pg_cond	145
5.4.3.5	md2cmds_pg_mutex	145
5.4.3.6	md2cmds_thread	145
5.5	pgpmac.c File Reference	146
5.5.1	Detailed Description	146
5.5.2	Function Documentation	147
5.5.2.1	main	147
5.5.2.2	pgpmac_printf	149
5.5.2.3	stdinService	149
5.5.3	Variable Documentation	150
5.5.3.1	ncurses_mutex	150
5.5.3.2	stdinfda	150
5.5.3.3	term_input	151
5.5.3.4	term_output	151
5.5.3.5	term_status	151
5.5.3.6	term_status2	151
5.6	pgpmac.h File Reference	152
5.6.1	Detailed Description	156
5.6.2	Define Documentation	157
5.6.2.1	LS_DISPLAY_WINDOW_HEIGHT	157
5.6.2.2	LS_DISPLAY_WINDOW_WIDTH	157
5.6.2.3	LS_PG_QUERY_STRING_LENGTH	157
5.6.2.4	MD2CMDS_CMD_LENGTH	157
5.6.3	Typedef Documentation	157
5.6.3.1	lspg_nextshot_t	157
5.6.3.2	lspmac_motor_t	157
5.6.3.3	pmac_cmd_queue_t	157
5.6.3.4	pmac_cmd_t	157
5.6.4	Function Documentation	158
5.6.4.1	lspg_init	158
5.6.4.2	lspg_run	158

5.6.4.3	lspg_seq_run_prep_all	158
5.6.4.4	lspg_zoom_lut_call	159
5.6.4.5	lspmac_init	159
5.6.4.6	lspmac_run	160
5.6.4.7	lsupdate_init	161
5.6.4.8	lsupdate_run	161
5.6.4.9	md2cmds_init	161
5.6.4.10	md2cmds_run	161
5.6.4.11	pgpmac_printf	162
5.6.4.12	PmacSockSendline	162
5.6.5	Variable Documentation	162
5.6.5.1	alignx	162
5.6.5.2	aligny	162
5.6.5.3	alignz	162
5.6.5.4	anal	162
5.6.5.5	apery	163
5.6.5.6	aperz	163
5.6.5.7	blight	163
5.6.5.8	blight_up	163
5.6.5.9	capy	163
5.6.5.10	capz	163
5.6.5.11	cenx	163
5.6.5.12	ceny	163
5.6.5.13	flight	163
5.6.5.14	fscint	164
5.6.5.15	fshut	164
5.6.5.16	kappa	164
5.6.5.17	lspg_nextshot	164
5.6.5.18	lspmac_motors	164
5.6.5.19	lspmac_moving_cond	164
5.6.5.20	lspmac_moving_flags	164
5.6.5.21	lspmac_moving_mutex	164
5.6.5.22	lspmac_nmotors	164
5.6.5.23	lspmac_shutter_cond	165
5.6.5.24	lspmac_shutter_has_opened	165
5.6.5.25	lspmac_shutter_mutex	165

5.6.5.26	lspmac_shutter_state	165
5.6.5.27	md2_status_mutex	165
5.6.5.28	md2cmds_cmd	165
5.6.5.29	md2cmds_cond	165
5.6.5.30	md2cmds_mutex	165
5.6.5.31	md2cmds_pg_cond	165
5.6.5.32	md2cmds_pg_mutex	166
5.6.5.33	ncurses_mutex	166
5.6.5.34	omega	166
5.6.5.35	phi	166
5.6.5.36	scinz	166
5.6.5.37	term_input	166
5.6.5.38	term_output	166
5.6.5.39	term_status	166
5.6.5.40	term_status2	166
5.6.5.41	zoom	167

Chapter 1

The LS-CAT pgpmac Project

[pgpmac.c](#)

Some pmac defines, typedefs, functions suggested by Delta Tau Accessory 54E User Manual, October 23, 2003 (C) 2003 by Delta Tau Data Systems, Inc. All rights reserved.

Original work Copyright (C) 2012 by Keith Brister, Northwestern University, All rights reserved.

This project implements the MD2 communications required for operation at LS-CAT and is intended to replace Windows XP based .NET code provided by MAATEL.

The need to do this is driven by a desire to make the system as efficient and fast as possible by combining various operations. A proof-of-principle version of this code saw frame rates of 23/minute as opposed to the nominal 18/minute we normally quote for 1 second exposures.

Additionally, as we rapidly approach EOL for Windows XP an alternative is urgently needed.

Structure

The project is roughly broken down as follows:

[pgpmac.h](#) All includes and defines. The only file included by the .c files in this project.

[pgpmac.c](#) Main: parses command line and starts up the various threads

[lspg.c](#) Handles communications with the controlling postgresql database

[md2cmds.c](#) Provides the equivalent (mostly) of the LS-CAT BLUMax code.

pmac_md2_ls-cat.pmc Code for the PMAC: compile and install with pmac executable program.

pmac_md2.sql Tables and procedures for the postgresql side of the project.

Notes:

- The postgresql and the pmac communications interfaces are asynchronous and rely heavily on the unix "poll" routine.
- The project is multithreaded and based on "pthreads".
- Most threads maintain a queue of commands to simplify communications with each other.
- Note that a MAATEL supported interface for a more recent version of Windows may be available, however, a bit of effort will be required to implement it at LS-CAT as the BLUMax code will likely require some revisions. This is still an option should the present project become intractable.

- An important constraint has been to run the MD2 either from the windows .NET environment or from the pgpmac environment. A consequence is that the pmac "pmc" file has been augmented to include new capabilities without destroying the code that the .NET interface requires.
- Epics support could come by adapting the "e.c" code to work here directly or could come by making use of the existing kv pair mechanism already in place or, as is most likely, a combination of the two.
- Ncurses support could include input lines for SQL queries and direct commands for supporting homing etc. Perhaps the F keys could change modes or use of special mode changing text commands. Output is not asynchronous. Although this is unlikely to cause a problem I'd hate to have the program hang because terminal output is hung up.
- PG queries come back as text instead of binary. We could reduce the numeric errors by using binary and things would run a tad faster, though it is unlikely anyone would notice or care about the speed.

MD2 Motors and Coordinate Systems

```

CS          Motor

1 1 X = Omega

2 17 X = Center X
18 Y = Center Y

3 2 X = Alignment X
3 Y = Alignment Y
4 Z = Alignment Z

--          5          Analyzer

4 6 X = Zoom

5 7 Y = Aperture Y
8 Z = Aperture Z
9 U = Capillary Y
    10 V = Capillary Z
    11 W = Scintillator Z

6 (None)

7          19 X = Kappa
    20 Y = Phi

```

MD2 Motion Programs

```

before calling, set
M4XX = 1:  flag to indicate we are running program XX
P variables as arguments

```

Program Description

```
1 home omega
2 home alignment table X
3 home alignment table Y
4 home alignment table Z
6 home camera zoom
7 home aperture Y
8 home aperture Z
9 home capillary Y
10 home capillary Z
11 home scintillator Z
17 home center X
18 home center Y
19 home kappa
20 home phi (Home position is not defined for phi ...)
25 kappa stress test

26 Combined Incremental move of X and Y in selected coordinate system
(Does not reset M426)
P170 = X increment
P171 = Y increment

31 scan omega
P170 = Start
P171 = End
P173 = Velocity (float)
P174 = Sample Rate (I5049)
P175 = Acceleration time
P176 = Gathering source
P177 = Number of passes
P178 = Shutter rising distance (units of omega motion)
P179 = Shutter falling distance (units of omega motion)
P180 = Exposure Time

34 Organ Scan
P169 = Motor Number
P170 = Start Position
P171 = End Position
P172 = Step Size
P173 = Motor Speed

35 Organ Homing

37 Organ Move (microdiff_hard.ini says we don't use this anymore)
P169 = Capillary Z
P170 = Scintillator Z
P171 = Aperture Z

50 Combined Incremental move of X and Y
P170 = X increment
P171 = Y increment

52 X oscillation (while M320 == 1)
(Does not reset M452)

53 Center X and Y Synchronized homing
```

54 Combined X, Y, Z absolute move

P170 = X

P171 = Y

P172 = Z

131 LS-CAT Modified Omega Scan

P170 = Shutter open position, in counts

P171 = Delta omega, in counts

P173 = Omega velocity (counts/msec)

P175 = Acceleration Time (msec)

P177 = Number of passes

P178 = Shutter Rising Distance

P179 = Shutter Falling Distance

P180 = Exposure Time (msec)

140 LS-CAT Move X Absolute

Q10 = X Value (cts)

141 LS-CAT Move Y Absolute

Q11 = Y Value (cts)

142 LS-CAT Move Z Absolute

Q12 = Z Value (cts)

150 LS-CAT Move X, Y Absolute

Q20 = X Value

Q21 = Y Value

160 LS-CAT Move X, Y, Z Absolute

Q30 = X Value

Q31 = Y Value

Q32 = Z Value

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

lspg_lock_detector_struct (Lock detector object Implements detector lock for exposure control)	9
lspg_lock_diffractionmeter_struct (Object used to impliment locking the diffractometer Critical to exposure timing)	10
lspg_nextshot_struct (Storage definition for nextshot query)	11
lspg_seq_run_prep_struct (Data collection running object)	24
lspg_wait_for_detector_struct (Object that implements detector / spindle timing We use database locks for exposure control and this implements the md2 portion of this handshake) . .	25
lspgQueryQueueStruct (Store each query along with it's callback function)	26
lspmac_cmd_queue_struct (PMAC command queue item)	27
lspmac_motor_struct (Motor information)	29
md2StatusStruct (The block of memory retrieved in a status request)	35
tagEthernetCmd (PMAC ethernet packet definition)	44

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

lspg.c (Postgresql support for the LS-CAT pgpmac project)	47
lspmac.c (Routines concerned with communication with PMAC)	83
lsupdate.c (Brings this MD2 code and the database kvs table into agreement)	132
md2cmds.c (Implements commands to run the md2 diffractometer attached to a PMAC controled by postgresql)	135
pgpmac.c (Main for the pgpmac project)	146
pgpmac.h (Headers for the entire pgpmac project)	152

Chapter 4

Data Structure Documentation

4.1 `lspg_lock_detector_struct` Struct Reference

lock detector object Implements detector lock for exposure control

Data Fields

- `pthread_mutex_t` [mutex](#)
- `pthread_cond_t` [cond](#)
- `int` [new_value_ready](#)

4.1.1 Detailed Description

lock detector object Implements detector lock for exposure control

Definition at line 718 of file `lspg.c`.

4.1.2 Field Documentation

4.1.2.1 `pthread_cond_t lspg_lock_detector_struct::cond`

Definition at line 720 of file `lspg.c`.

4.1.2.2 `pthread_mutex_t lspg_lock_detector_struct::mutex`

Definition at line 719 of file `lspg.c`.

4.1.2.3 `int lspg_lock_detector_struct::new_value_ready`

Definition at line 721 of file `lspg.c`.

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

- [lspg.c](#)

4.2 `lspg_lock_diffractometer_struct` Struct Reference

Object used to impliment locking the diffractometer Critical to exposure timing.

Data Fields

- `pthread_mutex_t` [mutex](#)
- `pthread_cond_t` [cond](#)
- `int` [new_value_ready](#)

4.2.1 Detailed Description

Object used to impliment locking the diffractometer Critical to exposure timing.

Definition at line 659 of file `lspg.c`.

4.2.2 Field Documentation

4.2.2.1 `pthread_cond_t lsgp_lock_diffractometer_struct::cond`

Definition at line 661 of file `lspg.c`.

4.2.2.2 `pthread_mutex_t lsgp_lock_diffractometer_struct::mutex`

Definition at line 660 of file `lspg.c`.

4.2.2.3 `int lsgp_lock_diffractometer_struct::new_value_ready`

Definition at line 662 of file `lspg.c`.

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

- [lspg.c](#)

4.3 lspg_nextshot_struct Struct Reference

Storage definition for nextshot query.

```
#include <pgpmac.h>
```

Data Fields

- pthread_mutex_t [mutex](#)
Our mutex for sanity in the multi-threaded program.
- pthread_cond_t [cond](#)
Condition to wait for a response from our postgresql server.
- int [new_value_ready](#)
Our flag for the condition to wait for.
- int [no_rows_returned](#)
flag indicating that no rows were returned.
- char * [dsdir](#)
Directory for data relative to the ESAF home directory.
- int [dsdir_isnull](#)
- char * [dspid](#)
ID string identifying this dataset.
- int [dspid_isnull](#)
- double [dsowidth](#)
dataset defined oscillation width
- int [dsowidth_isnull](#)
- char * [dsoscaxis](#)
dataset defined oscillation axis (always omega)
- int [dsoscaxis_isnull](#)
- double [dsexp](#)
dataset defined exposure time
- int [dsexp_isnull](#)
- long long [skey](#)
key identifying a particular image
- int [skey_isnull](#)
- double [sstart](#)
starting angle
- int [sstart_isnull](#)
- char * [sfn](#)
file name

- int [sfn_isnull](#)
- double [dsphi](#)
dataset defined starting phi angle
- int [dsphi_isnull](#)
- double [dsomega](#)
dataset defined starting omega angle
- int [dsomega_isnull](#)
- double [dskappa](#)
dataset defined starting kappa angle
- int [dskappa_isnull](#)
- double [dsdist](#)
dataset defined detector distance
- int [dsdist_isnull](#)
- double [dsnrg](#)
dataset defined energy
- int [dsnrg_isnull](#)
- unsigned int [dshpid](#)
sample holder ID
- int [dshpid_isnull](#)
- double [cx](#)
centering table x position
- int [cx_isnull](#)
- double [cy](#)
centering table y position
- int [cy_isnull](#)
- double [ax](#)
alignment table x position
- int [ax_isnull](#)
- double [ay](#)
alignment table y position
- int [ay_isnull](#)
- double [az](#)
alignment table z position
- int [az_isnull](#)
- int [active](#)
flag: 1=move to indicated center position, 0=don't move center or alignment tables
- int [active_isnull](#)

- int `sindex`
index of frame (used to generate the file extension)
- int `sindex_isnull`
- char * `stype`
"Normal" or "Gridsearch"
- int `stype_isnull`
- double `dsowidth2`
next image oscillation width
- int `dsowidth2_isnull`
- char * `dsoscaxis2`
next image ascillation axis (always "omega")
- int `dsoscaxis2_isnull`
- double `dsexp2`
next image exposure time
- int `dsexp2_isnull`
- double `sstart2`
next image start angle
- int `sstart2_isnull`
- double `dsphi2`
next image phi position
- int `dsphi2_isnull`
- double `dsomega2`
next image omega position
- int `dsomega2_isnull`
- double `dskappa2`
next image kappa position
- int `dskappa2_isnull`
- double `dsdist2`
next image distance
- int `dsdist2_isnull`
- double `dsnrg2`
next image energy
- int `dsnrg2_isnull`
- double `cx2`
next image centering table x position
- int `cx2_isnull`
- double `cy2`
next image centering table y position

- int [cy2_isnull](#)

- double [ax2](#)

next image alignment x position

- int [ax2_isnull](#)

- double [ay2](#)

next image alignment y position

- int [ay2_isnull](#)

- double [az2](#)

next image alignment z position

- int [az2_isnull](#)

- int [active2](#)

flag: 1 if next image should use the above centering parameters

- int [active2_isnull](#)

- int [sindex2](#)

next image index number

- int [sindex2_isnull](#)

- char * [stype2](#)

next image type ("Normal" or "Gridsearch")

- int [stype2_isnull](#)

4.3.1 Detailed Description

Storage definition for nextshot query. The next shot query returns all the information needed to collect the next data frame. Since SQL allows for null fields independently from blank strings a separate integer is used as a flag for this case. This adds to the program complexity but allows for some important cases. Suck it up.

Definition at line 111 of file pgpmac.h.

4.3.2 Field Documentation

4.3.2.1 int `lspg_nextshot_struct::active`

flag: 1=move to indicated center position, 0=don't move center or alignment tables

Definition at line 174 of file pgpmac.h.

4.3.2.2 int `lspg_nextshot_struct::active2`

flag: 1 if next image should use the above centering parameters

Definition at line 225 of file pgpmac.h.

4.3.2.3 `int lspg_nextshot_struct::active2_isnull`

Definition at line 226 of file `pgpmac.h`.

4.3.2.4 `int lspg_nextshot_struct::active_isnull`

Definition at line 175 of file `pgpmac.h`.

4.3.2.5 `double lspg_nextshot_struct::ax`

alignment table x position

Definition at line 165 of file `pgpmac.h`.

4.3.2.6 `double lspg_nextshot_struct::ax2`

next image alignment x position

Definition at line 216 of file `pgpmac.h`.

4.3.2.7 `int lspg_nextshot_struct::ax2_isnull`

Definition at line 217 of file `pgpmac.h`.

4.3.2.8 `int lspg_nextshot_struct::ax_isnull`

Definition at line 166 of file `pgpmac.h`.

4.3.2.9 `double lspg_nextshot_struct::ay`

alignment table y position

Definition at line 168 of file `pgpmac.h`.

4.3.2.10 `double lspg_nextshot_struct::ay2`

next image alignment y position

Definition at line 219 of file `pgpmac.h`.

4.3.2.11 `int lspg_nextshot_struct::ay2_isnull`

Definition at line 220 of file `pgpmac.h`.

4.3.2.12 `int lspg_nextshot_struct::ay_isnull`

Definition at line 169 of file `pgpmac.h`.

4.3.2.13 double lspg_nextshot_struct::az

alignment table z position

Definition at line 171 of file pgpmac.h.

4.3.2.14 double lspg_nextshot_struct::az2

next image alignment z position

Definition at line 222 of file pgpmac.h.

4.3.2.15 int lspg_nextshot_struct::az2_isnull

Definition at line 223 of file pgpmac.h.

4.3.2.16 int lspg_nextshot_struct::az_isnull

Definition at line 172 of file pgpmac.h.

4.3.2.17 pthread_cond_t lspg_nextshot_struct::cond

Condition to wait for a response from our postgresql server.

Definition at line 113 of file pgpmac.h.

4.3.2.18 double lspg_nextshot_struct::cx

centering table x position

Definition at line 159 of file pgpmac.h.

4.3.2.19 double lspg_nextshot_struct::cx2

next image centering table x position

Definition at line 210 of file pgpmac.h.

4.3.2.20 int lspg_nextshot_struct::cx2_isnull

Definition at line 211 of file pgpmac.h.

4.3.2.21 int lspg_nextshot_struct::cx_isnull

Definition at line 160 of file pgpmac.h.

4.3.2.22 double lspg_nextshot_struct::cy

centering table y position

Definition at line 162 of file pgpmac.h.

4.3.2.23 double `lspg_nextshot_struct::cy2`

next image centering table y position

Definition at line 213 of file `pgpmac.h`.

4.3.2.24 int `lspg_nextshot_struct::cy2_isnull`

Definition at line 214 of file `pgpmac.h`.

4.3.2.25 int `lspg_nextshot_struct::cy_isnull`

Definition at line 163 of file `pgpmac.h`.

4.3.2.26 char* `lspg_nextshot_struct::dsdir`

Directory for data relative to the ESAF home directory.

Definition at line 117 of file `pgpmac.h`.

4.3.2.27 int `lspg_nextshot_struct::dsdir_isnull`

Definition at line 118 of file `pgpmac.h`.

4.3.2.28 double `lspg_nextshot_struct::dsdist`

dataset defined detector distance

Definition at line 150 of file `pgpmac.h`.

4.3.2.29 double `lspg_nextshot_struct::dsdist2`

next image distance

Definition at line 204 of file `pgpmac.h`.

4.3.2.30 int `lspg_nextshot_struct::dsdist2_isnull`

Definition at line 205 of file `pgpmac.h`.

4.3.2.31 int `lspg_nextshot_struct::dsdist_isnull`

Definition at line 151 of file `pgpmac.h`.

4.3.2.32 double `lspg_nextshot_struct::dsexp`

dataset defined exposure time

Definition at line 129 of file `pgpmac.h`.

4.3.2.33 double lspg_nextshot_struct::dsexp2

next image exposure time

Definition at line 189 of file pgpmac.h.

4.3.2.34 int lspg_nextshot_struct::dsexp2_isnull

Definition at line 190 of file pgpmac.h.

4.3.2.35 int lspg_nextshot_struct::dsexp_isnull

Definition at line 130 of file pgpmac.h.

4.3.2.36 unsigned int lspg_nextshot_struct::dshpid

sample holder ID

Definition at line 156 of file pgpmac.h.

4.3.2.37 int lspg_nextshot_struct::dshpid_isnull

Definition at line 157 of file pgpmac.h.

4.3.2.38 double lspg_nextshot_struct::dskappa

dataset defined starting kappa angle

Definition at line 147 of file pgpmac.h.

4.3.2.39 double lspg_nextshot_struct::dskappa2

next image kappa position

Definition at line 201 of file pgpmac.h.

4.3.2.40 int lspg_nextshot_struct::dskappa2_isnull

Definition at line 202 of file pgpmac.h.

4.3.2.41 int lspg_nextshot_struct::dskappa_isnull

Definition at line 148 of file pgpmac.h.

4.3.2.42 double lspg_nextshot_struct::dsnr

dataset defined energy

Definition at line 153 of file pgpmac.h.

4.3.2.43 double lspg_nextshot_struct::dsnrg2

next image energy

Definition at line 207 of file pgpmac.h.

4.3.2.44 int lspg_nextshot_struct::dsnrg2_isnull

Definition at line 208 of file pgpmac.h.

4.3.2.45 int lspg_nextshot_struct::dsnrg_isnull

Definition at line 154 of file pgpmac.h.

4.3.2.46 double lspg_nextshot_struct::dsomega

dataset defined starting omega angle

Definition at line 144 of file pgpmac.h.

4.3.2.47 double lspg_nextshot_struct::dsomega2

next image omega position

Definition at line 198 of file pgpmac.h.

4.3.2.48 int lspg_nextshot_struct::dsomega2_isnull

Definition at line 199 of file pgpmac.h.

4.3.2.49 int lspg_nextshot_struct::dsomega_isnull

Definition at line 145 of file pgpmac.h.

4.3.2.50 char* lspg_nextshot_struct::dsoscaxis

dataset defined oscillation axis (always omega)

Definition at line 126 of file pgpmac.h.

4.3.2.51 char* lspg_nextshot_struct::dsoscaxis2

next image ascillation axis (always "omega")

Definition at line 186 of file pgpmac.h.

4.3.2.52 int lspg_nextshot_struct::dsoscaxis2_isnull

Definition at line 187 of file pgpmac.h.

4.3.2.53 int lspg_nextshot_struct::dsoscaxis_isnull

Definition at line 127 of file pgpmac.h.

4.3.2.54 double lspg_nextshot_struct::dsowidth

dataset defined oscillation width

Definition at line 123 of file pgpmac.h.

4.3.2.55 double lspg_nextshot_struct::dsowidth2

next image oscillation width

Definition at line 183 of file pgpmac.h.

4.3.2.56 int lspg_nextshot_struct::dsowidth2_isnull

Definition at line 184 of file pgpmac.h.

4.3.2.57 int lspg_nextshot_struct::dsowidth_isnull

Definition at line 124 of file pgpmac.h.

4.3.2.58 double lspg_nextshot_struct::dsphi

dataset defined starting phi angle

Definition at line 141 of file pgpmac.h.

4.3.2.59 double lspg_nextshot_struct::dsphi2

next image phi position

Definition at line 195 of file pgpmac.h.

4.3.2.60 int lspg_nextshot_struct::dsphi2_isnull

Definition at line 196 of file pgpmac.h.

4.3.2.61 int lspg_nextshot_struct::dsphi_isnull

Definition at line 142 of file pgpmac.h.

4.3.2.62 char* lspg_nextshot_struct::dspid

ID string identifying this dataset.

Definition at line 120 of file pgpmac.h.

4.3.2.63 `int lspg_nextshot_struct::dspid_isnull`

Definition at line 121 of file `pgpmac.h`.

4.3.2.64 `pthread_mutex_t lspg_nextshot_struct::mutex`

Our mutex for sanity in the multi-threaded program.

Definition at line 112 of file `pgpmac.h`.

4.3.2.65 `int lspg_nextshot_struct::new_value_ready`

Our flag for the condition to wait for.

Definition at line 114 of file `pgpmac.h`.

4.3.2.66 `int lspg_nextshot_struct::no_rows_returned`

flag indicating that no rows were returned.

Definition at line 115 of file `pgpmac.h`.

4.3.2.67 `char* lspg_nextshot_struct::sfn`

file name

Definition at line 138 of file `pgpmac.h`.

4.3.2.68 `int lspg_nextshot_struct::sfn_isnull`

Definition at line 139 of file `pgpmac.h`.

4.3.2.69 `int lspg_nextshot_struct::sindex`

index of frame (used to generate the file extension)

Definition at line 177 of file `pgpmac.h`.

4.3.2.70 `int lspg_nextshot_struct::sindex2`

next image index number

Definition at line 228 of file `pgpmac.h`.

4.3.2.71 `int lspg_nextshot_struct::sindex2_isnull`

Definition at line 229 of file `pgpmac.h`.

4.3.2.72 `int lspg_nextshot_struct::sindex_isnull`

Definition at line 178 of file `pgpmac.h`.

4.3.2.73 long long lspg_nextshot_struct::skey

key identifying a particular image

Definition at line 132 of file pgpmac.h.

4.3.2.74 int lspg_nextshot_struct::skey_isnull

Definition at line 133 of file pgpmac.h.

4.3.2.75 double lspg_nextshot_struct::sstart

starting angle

Definition at line 135 of file pgpmac.h.

4.3.2.76 double lspg_nextshot_struct::sstart2

next image start angle

Definition at line 192 of file pgpmac.h.

4.3.2.77 int lspg_nextshot_struct::sstart2_isnull

Definition at line 193 of file pgpmac.h.

4.3.2.78 int lspg_nextshot_struct::sstart_isnull

Definition at line 136 of file pgpmac.h.

4.3.2.79 char* lspg_nextshot_struct::stype

"Normal" or "Gridsearch"

Definition at line 180 of file pgpmac.h.

4.3.2.80 char* lspg_nextshot_struct::stype2

next image type ("Normal" or "Gridsearch")

Definition at line 231 of file pgpmac.h.

4.3.2.81 int lspg_nextshot_struct::stype2_isnull

Definition at line 232 of file pgpmac.h.

4.3.2.82 int lspg_nextshot_struct::stype_isnull

Definition at line 181 of file pgpmac.h.

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

- [pgpmac.h](#)

4.4 `lspg_seq_run_prep_struct` Struct Reference

Data collection running object.

Data Fields

- `pthread_mutex_t` [mutex](#)
- `pthread_cond_t` [cond](#)
- `int` [new_value_ready](#)

4.4.1 Detailed Description

Data collection running object.

Definition at line 776 of file `lspg.c`.

4.4.2 Field Documentation

4.4.2.1 `pthread_cond_t lspg_seq_run_prep_struct::cond`

Definition at line 778 of file `lspg.c`.

4.4.2.2 `pthread_mutex_t lspg_seq_run_prep_struct::mutex`

Definition at line 777 of file `lspg.c`.

4.4.2.3 `int lspg_seq_run_prep_struct::new_value_ready`

Definition at line 779 of file `lspg.c`.

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

- [lspg.c](#)

4.5 lspg_wait_for_detector_struct Struct Reference

Object that implements detector / spindle timing We use database locks for exposure control and this implements the md2 portion of this handshake.

Data Fields

- pthread_mutex_t [mutex](#)
- pthread_cond_t [cond](#)
- int [new_value_ready](#)

4.5.1 Detailed Description

Object that implements detector / spindle timing We use database locks for exposure control and this implements the md2 portion of this handshake.

Definition at line 594 of file lspg.c.

4.5.2 Field Documentation

4.5.2.1 pthread_cond_t lspg_wait_for_detector_struct::cond

Definition at line 596 of file lspg.c.

4.5.2.2 pthread_mutex_t lspg_wait_for_detector_struct::mutex

Definition at line 595 of file lspg.c.

4.5.2.3 int lspg_wait_for_detector_struct::new_value_ready

Definition at line 597 of file lspg.c.

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

- [lspg.c](#)

4.6 lspgQueryQueueStruct Struct Reference

Store each query along with it's callback function.

Data Fields

- char [qs](#) [LS_PG_QUERY_STRING_LENGTH]
our queries should all be pretty short as we'll just be calling functions: fixed length here simplifies memory management
- void(* [onResponse](#))(struct [lspgQueryQueueStruct](#) *qq, PGresult *pgr)
Callback function for when a query returns a result.

4.6.1 Detailed Description

Store each query along with it's callback function. All calls are asynchronous

Definition at line 49 of file [lspg.c](#).

4.6.2 Field Documentation

4.6.2.1 void(* [lspgQueryQueueStruct::onResponse](#))(struct [lspgQueryQueueStruct](#) *qq, PGresult *pgr)

Callback function for when a query returns a result.

4.6.2.2 char [lspgQueryQueueStruct::qs](#)[LS_PG_QUERY_STRING_LENGTH]

our queries should all be pretty short as we'll just be calling functions: fixed length here simplifies memory management

Definition at line 50 of file [lspg.c](#).

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

- [lspg.c](#)

4.7 lspmac_cmd_queue_struct Struct Reference

PMAC command queue item.

```
#include <pgpmac.h>
```

Data Fields

- [pmac_cmd_t pcmd](#)
the pmac command to send
- [int no_reply](#)
1 = no reply is expected, 0 = expect a reply
- [struct timespec time_sent](#)
time this item was dequeued and sent to the pmac
- [unsigned char rbuff \[1400\]](#)
buffer for the returned bytes
- [void\(* onResponse\)\(struct lspmac_cmd_queue_struct *, int, unsigned char *\)](#)
function to call when response is received. args are (int fd, nreturned, buffer)

4.7.1 Detailed Description

PMAC command queue item. Command queue items are fixed length to simplify memory management.

Definition at line 56 of file pgpmac.h.

4.7.2 Field Documentation

4.7.2.1 [int lspmac_cmd_queue_struct::no_reply](#)

1 = no reply is expected, 0 = expect a reply

Definition at line 58 of file pgpmac.h.

4.7.2.2 [void\(* lspmac_cmd_queue_struct::onResponse\)\(struct lspmac_cmd_queue_struct *, int, unsigned char *\)](#)

function to call when response is received. args are (int fd, nreturned, buffer)

4.7.2.3 [pmac_cmd_t lspmac_cmd_queue_struct::pcmd](#)

the pmac command to send

Definition at line 57 of file pgpmac.h.

4.7.2.4 unsigned char lspmac_cmd_queue_struct::rbuff[1400]

buffer for the returned bytes

Definition at line 60 of file pgpmac.h.

4.7.2.5 struct timespec lspmac_cmd_queue_struct::time_sent [read]

time this item was dequeued and sent to the pmac

Definition at line 59 of file pgpmac.h.

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

- [pgpmac.h](#)

4.8 lspmac_motor_struct Struct Reference

Motor information.

```
#include <pgpmac.h>
```

Data Fields

- pthread_mutex_t [mutex](#)
coordinate waiting for motor to be done
- pthread_cond_t [cond](#)
- void(* [read](#))(struct [lspmac_motor_struct](#) *)
function to read the motor status and position
- int [not_done](#)
set to 1 when request is queued, zero after motion has toggled
- int [motion_seen](#)
set to 1 when motion has been verified to have started
- struct [lspmac_cmd_queue_struct](#) * [pq](#)
the queue item requesting motion. Used to check time request was made
- int [requested_pos_cnts](#)
requested position
- int * [actual_pos_cnts_p](#)
pointer to the md2_status structure to the actual position
- double [position](#)
scaled position
- double [reported_position](#)
previous position reported to the database
- double [requested_position](#)
The position as requested by the user.
- double [update_resolution](#)
Change needs to be at least this big to report as a new position to the database.
- int * [status1](#)
First 24 bit PMAC motor status word.
- int * [status2](#)
Second 24 bit PMAC motor status word.
- int [motor_num](#)
pmac motor number

- char * [dac_mvar](#)
controlling mvariable as a string
- char * [name](#)
Name of motor as referred by ls database kvs table.
- char * [units](#)
string to use as the units
- char * [format](#)
printf format
- char * [write_fmt](#)
Format string to write requested position to PMAC used for binary io.
- int * [read_ptr](#)
With read_mask finds bit to read for binary i/o.
- int [read_mask](#)
With read_ptr find bit to read for binary i/o.
- void(* [moveAbs](#))(struct [lspmac_motor_struct](#) *, double)
function to move the motor
- double [u2c](#)
conversion from counts to units: 0.0 means not loaded yet
- double * [lut](#)
lookup table (instead of u2c)
- int [nlut](#)
length of lut
- double [max_speed](#)
our maximum speed (cts/msec)
- double [max_accel](#)
our maximum acceleration (cts/msec²)
- WINDOW * [win](#)
our ncurses window

4.8.1 Detailed Description

Motor information. A catchall for motors and motor like objects. Not all members are used by all objects.
Definition at line 69 of file pgpmac.h.

4.8.2 Field Documentation

4.8.2.1 `int* lspmac_motor_struct::actual_pos_cnts_p`

pointer to the md2_status structure to the actual position

Definition at line 78 of file pgpmac.h.

4.8.2.2 `pthread_cond_t lspmac_motor_struct::cond`

Definition at line 71 of file pgpmac.h.

4.8.2.3 `char* lspmac_motor_struct::dac_mvar`

controlling mvariable as a string

Definition at line 86 of file pgpmac.h.

4.8.2.4 `char* lspmac_motor_struct::format`

printf format

Definition at line 89 of file pgpmac.h.

4.8.2.5 `double* lspmac_motor_struct::lut`

lookup table (instead of u2c)

Definition at line 95 of file pgpmac.h.

4.8.2.6 `double lspmac_motor_struct::max_accel`

our maximum acceleration (cts/msec²)

Definition at line 98 of file pgpmac.h.

4.8.2.7 `double lspmac_motor_struct::max_speed`

our maximum speed (cts/msec)

Definition at line 97 of file pgpmac.h.

4.8.2.8 `int lspmac_motor_struct::motion_seen`

set to 1 when motion has been verified to have started

Definition at line 74 of file pgpmac.h.

4.8.2.9 `int lspmac_motor_struct::motor_num`

pmac motor number

Definition at line 85 of file pgpmac.h.

4.8.2.10 void(* lspmac_motor_struct::moveAbs)(struct lspmac_motor_struct *, double)

function to move the motor

4.8.2.11 pthread_mutex_t lspmac_motor_struct::mutex

coordinate waiting for motor to be done

Definition at line 70 of file pgpmac.h.

4.8.2.12 char* lspmac_motor_struct::name

Name of motor as referred by ls database kvs table.

Definition at line 87 of file pgpmac.h.

4.8.2.13 int lspmac_motor_struct::nlut

length of lut

Definition at line 96 of file pgpmac.h.

4.8.2.14 int lspmac_motor_struct::not_done

set to 1 when request is queued, zero after motion has toggled

Definition at line 73 of file pgpmac.h.

4.8.2.15 double lspmac_motor_struct::position

scaled position

Definition at line 79 of file pgpmac.h.

4.8.2.16 struct lspmac_cmd_queue_struct* lspmac_motor_struct::pq [read]

the queue item requesting motion. Used to check time request was made

Definition at line 75 of file pgpmac.h.

4.8.2.17 void(* lspmac_motor_struct::read)(struct lspmac_motor_struct *)

function to read the motor status and position

4.8.2.18 int lspmac_motor_struct::read_mask

With read_ptr find bit to read for binary i/o.

Definition at line 92 of file pgpmac.h.

4.8.2.19 int* lspmac_motor_struct::read_ptr

With read_mask finds bit to read for binary i/o.

Definition at line 91 of file pgpmac.h.

4.8.2.20 double lspmac_motor_struct::reported_position

previous position reported to the database

Definition at line 80 of file pgpmac.h.

4.8.2.21 int lspmac_motor_struct::requested_pos_cnts

requested position

Definition at line 77 of file pgpmac.h.

4.8.2.22 double lspmac_motor_struct::requested_position

The position as requested by the user.

Definition at line 81 of file pgpmac.h.

4.8.2.23 int* lspmac_motor_struct::status1

First 24 bit PMAC motor status word.

Definition at line 83 of file pgpmac.h.

4.8.2.24 int* lspmac_motor_struct::status2

Second 24 bit PMAC motor status word.

Definition at line 84 of file pgpmac.h.

4.8.2.25 double lspmac_motor_struct::u2c

conversion from counts to units: 0.0 means not loaded yet

Definition at line 94 of file pgpmac.h.

4.8.2.26 char* lspmac_motor_struct::units

string to use as the units

Definition at line 88 of file pgpmac.h.

4.8.2.27 double lspmac_motor_struct::update_resolution

Change needs to be at least this big to report as a new position to the database.

Definition at line 82 of file pgpmac.h.

4.8.2.28 WINDOW* lspmac_motor_struct::win

our ncurses window

Definition at line 99 of file pgpmac.h.

4.8.2.29 char* lspmac_motor_struct::write_fmt

Format string to write requested position to PMAC used for binary io.

Definition at line 90 of file pgpmac.h.

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

- [pgpmac.h](#)

4.9 md2StatusStruct Struct Reference

The block of memory retrieved in a status request.

Data Fields

- int [dummy1](#)
- int [omega_status_1](#)
- int [alignx_status_1](#)
- int [aligny_status_1](#)
- int [alignz_status_1](#)
- int [analyzer_status_1](#)
- int [zoom_status_1](#)
- int [aperturey_status_1](#)
- int [aperturez_status_1](#)
- int [copy_status_1](#)
- int [capz_status_1](#)
- int [scint_status_1](#)
- int [centerx_status_1](#)
- int [centery_status_1](#)
- int [kappa_status_1](#)
- int [phi_status_1](#)
- int [dummy2](#)
- int [omega_status_2](#)
- int [alignx_status_2](#)
- int [aligny_status_2](#)
- int [alignz_status_2](#)
- int [analyzer_status_2](#)
- int [zoom_status_2](#)
- int [aperturey_status_2](#)
- int [aperturez_status_2](#)
- int [copy_status_2](#)
- int [capz_status_2](#)
- int [scint_status_2](#)
- int [centerx_status_2](#)
- int [centery_status_2](#)
- int [kappa_status_2](#)
- int [phi_status_2](#)
- int [dummy3](#)
- int [omega_act_pos](#)
- int [alignx_act_pos](#)
- int [aligny_act_pos](#)
- int [alignz_act_pos](#)
- int [analyzer_act_pos](#)
- int [zoom_act_pos](#)
- int [aperturey_act_pos](#)
- int [aperturez_act_pos](#)
- int [copy_act_pos](#)
- int [capz_act_pos](#)

- int [scint_act_pos](#)
- int [centerx_act_pos](#)
- int [centery_act_pos](#)
- int [kappa_act_pos](#)
- int [phi_act_pos](#)
- int [acc11c_1](#)
- int [acc11c_2](#)
- int [acc11c_3](#)
- int [acc11c_5](#)
- int [acc11c_6](#)
- int [front_dac](#)
- int [back_dac](#)
- int [scint_piezo](#)
- int [dummy4](#)
- int [dummy5](#)
- int [dummy6](#)
- int [dummy7](#)
- int [dummy8](#)
- int [dummy9](#)
- int [dummyA](#)
- int [dummyB](#)
- int [fs_is_open](#)
- int [phiscan](#)
- int [fs_has_opened](#)
- int [fs_has_opened_globally](#)
- int [number_passes](#)
- int [moving_flags](#)

4.9.1 Detailed Description

The block of memory retrieved in a status request.

Definition at line 182 of file `lspmac.c`.

4.9.2 Field Documentation

4.9.2.1 int md2StatusStruct::acc11c_1

Definition at line 249 of file `lspmac.c`.

4.9.2.2 int md2StatusStruct::acc11c_2

Definition at line 250 of file `lspmac.c`.

4.9.2.3 int md2StatusStruct::acc11c_3

Definition at line 251 of file `lspmac.c`.

4.9.2.4 int md2StatusStruct::acc11c_5

Definition at line 252 of file lspmac.c.

4.9.2.5 int md2StatusStruct::acc11c_6

Definition at line 253 of file lspmac.c.

4.9.2.6 int md2StatusStruct::alignx_act_pos

Definition at line 233 of file lspmac.c.

4.9.2.7 int md2StatusStruct::alignx_status_1

Definition at line 199 of file lspmac.c.

4.9.2.8 int md2StatusStruct::alignx_status_2

Definition at line 216 of file lspmac.c.

4.9.2.9 int md2StatusStruct::aligny_act_pos

Definition at line 234 of file lspmac.c.

4.9.2.10 int md2StatusStruct::aligny_status_1

Definition at line 200 of file lspmac.c.

4.9.2.11 int md2StatusStruct::aligny_status_2

Definition at line 217 of file lspmac.c.

4.9.2.12 int md2StatusStruct::alignz_act_pos

Definition at line 235 of file lspmac.c.

4.9.2.13 int md2StatusStruct::alignz_status_1

Definition at line 201 of file lspmac.c.

4.9.2.14 int md2StatusStruct::alignz_status_2

Definition at line 218 of file lspmac.c.

4.9.2.15 int md2StatusStruct::analyzer_act_pos

Definition at line 236 of file lspmac.c.

4.9.2.16 int md2StatusStruct::analyzer_status_1

Definition at line 202 of file lspmac.c.

4.9.2.17 int md2StatusStruct::analyzer_status_2

Definition at line 219 of file lspmac.c.

4.9.2.18 int md2StatusStruct::aperturey_act_pos

Definition at line 238 of file lspmac.c.

4.9.2.19 int md2StatusStruct::aperturey_status_1

Definition at line 204 of file lspmac.c.

4.9.2.20 int md2StatusStruct::aperturey_status_2

Definition at line 221 of file lspmac.c.

4.9.2.21 int md2StatusStruct::aperturez_act_pos

Definition at line 239 of file lspmac.c.

4.9.2.22 int md2StatusStruct::aperturez_status_1

Definition at line 205 of file lspmac.c.

4.9.2.23 int md2StatusStruct::aperturez_status_2

Definition at line 222 of file lspmac.c.

4.9.2.24 int md2StatusStruct::back_dac

Definition at line 255 of file lspmac.c.

4.9.2.25 int md2StatusStruct::copy_act_pos

Definition at line 240 of file lspmac.c.

4.9.2.26 int md2StatusStruct::capy_status_1

Definition at line 206 of file lspmac.c.

4.9.2.27 int md2StatusStruct::capy_status_2

Definition at line 223 of file lspmac.c.

4.9.2.28 int md2StatusStruct::capz_act_pos

Definition at line 241 of file lspmac.c.

4.9.2.29 int md2StatusStruct::capz_status_1

Definition at line 207 of file lspmac.c.

4.9.2.30 int md2StatusStruct::capz_status_2

Definition at line 224 of file lspmac.c.

4.9.2.31 int md2StatusStruct::centerx_act_pos

Definition at line 243 of file lspmac.c.

4.9.2.32 int md2StatusStruct::centerx_status_1

Definition at line 209 of file lspmac.c.

4.9.2.33 int md2StatusStruct::centerx_status_2

Definition at line 226 of file lspmac.c.

4.9.2.34 int md2StatusStruct::centery_act_pos

Definition at line 244 of file lspmac.c.

4.9.2.35 int md2StatusStruct::centery_status_1

Definition at line 210 of file lspmac.c.

4.9.2.36 int md2StatusStruct::centery_status_2

Definition at line 227 of file lspmac.c.

4.9.2.37 int md2StatusStruct::dummy1

Definition at line 197 of file lspmac.c.

4.9.2.38 int md2StatusStruct::dummy2

Definition at line 214 of file lspmac.c.

4.9.2.39 int md2StatusStruct::dummy3

Definition at line 231 of file lspmac.c.

4.9.2.40 int md2StatusStruct::dummy4

Definition at line 258 of file lspmac.c.

4.9.2.41 int md2StatusStruct::dummy5

Definition at line 259 of file lspmac.c.

4.9.2.42 int md2StatusStruct::dummy6

Definition at line 260 of file lspmac.c.

4.9.2.43 int md2StatusStruct::dummy7

Definition at line 261 of file lspmac.c.

4.9.2.44 int md2StatusStruct::dummy8

Definition at line 262 of file lspmac.c.

4.9.2.45 int md2StatusStruct::dummy9

Definition at line 263 of file lspmac.c.

4.9.2.46 int md2StatusStruct::dummyA

Definition at line 264 of file lspmac.c.

4.9.2.47 int md2StatusStruct::dummyB

Definition at line 265 of file lspmac.c.

4.9.2.48 int md2StatusStruct::front_dac

Definition at line 254 of file lspmac.c.

4.9.2.49 int md2StatusStruct::fs_has_opened

Definition at line 269 of file lspmac.c.

4.9.2.50 int md2StatusStruct::fs_has_opened_globally

Definition at line 270 of file lspmac.c.

4.9.2.51 int md2StatusStruct::fs_is_open

Definition at line 267 of file lspmac.c.

4.9.2.52 int md2StatusStruct::kappa_act_pos

Definition at line 245 of file lspmac.c.

4.9.2.53 int md2StatusStruct::kappa_status_1

Definition at line 211 of file lspmac.c.

4.9.2.54 int md2StatusStruct::kappa_status_2

Definition at line 228 of file lspmac.c.

4.9.2.55 int md2StatusStruct::moving_flags

Definition at line 273 of file lspmac.c.

4.9.2.56 int md2StatusStruct::number_passes

Definition at line 271 of file lspmac.c.

4.9.2.57 int md2StatusStruct::omega_act_pos

Definition at line 232 of file lspmac.c.

4.9.2.58 int md2StatusStruct::omega_status_1

Definition at line 198 of file lspmac.c.

4.9.2.59 int md2StatusStruct::omega_status_2

Definition at line 215 of file lspmac.c.

4.9.2.60 int md2StatusStruct::phi_act_pos

Definition at line 246 of file lspmac.c.

4.9.2.61 int md2StatusStruct::phi_status_1

Definition at line 212 of file lspmac.c.

4.9.2.62 int md2StatusStruct::phi_status_2

Definition at line 229 of file lspmac.c.

4.9.2.63 int md2StatusStruct::phiscan

Definition at line 268 of file lspmac.c.

4.9.2.64 int md2StatusStruct::scint_act_pos

Definition at line 242 of file lspmac.c.

4.9.2.65 int md2StatusStruct::scint_piezo

Definition at line 256 of file lspmac.c.

4.9.2.66 int md2StatusStruct::scint_status_1

Definition at line 208 of file lspmac.c.

4.9.2.67 int md2StatusStruct::scint_status_2

Definition at line 225 of file lspmac.c.

4.9.2.68 int md2StatusStruct::zoom_act_pos

Definition at line 237 of file lspmac.c.

4.9.2.69 int md2StatusStruct::zoom_status_1

Definition at line 203 of file lspmac.c.

4.9.2.70 int md2StatusStruct::zoom_status_2

Definition at line 220 of file lspmac.c.

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

- [lspmac.c](#)

4.10 tagEthernetCmd Struct Reference

PMAC ethernet packet definition.

```
#include <pgpmac.h>
```

Data Fields

- unsigned char [RequestType](#)
VR_UPLOAD or VR_DOWNLOAD.
- unsigned char [Request](#)
The command to run (VR_PMAC_GETMEM, etc).
- unsigned short [wValue](#)
Command parameter 1.
- unsigned short [wIndex](#)
Command parameter 2.
- unsigned short [wLength](#)
Number of bytes in bData.
- unsigned char [bData](#) [1492]
The data buffer, if required.

4.10.1 Detailed Description

PMAC ethernet packet definition. Taken directly from the Delta Tau documentation.

Definition at line 43 of file pgpmac.h.

4.10.2 Field Documentation

4.10.2.1 unsigned char tagEthernetCmd::bData[1492]

The data buffer, if required.

Definition at line 49 of file pgpmac.h.

4.10.2.2 unsigned char tagEthernetCmd::Request

The command to run (VR_PMAC_GETMEM, etc).

Definition at line 45 of file pgpmac.h.

4.10.2.3 unsigned char tagEthernetCmd::RequestType

VR_UPLOAD or VR_DOWNLOAD.

Definition at line 44 of file pgpmac.h.

4.10.2.4 unsigned short tagEthernetCmd::wIndex

Command parameter 2.

Definition at line 47 of file pgpmac.h.

4.10.2.5 unsigned short tagEthernetCmd::wLength

Number of bytes in bData.

Definition at line 48 of file pgpmac.h.

4.10.2.6 unsigned short tagEthernetCmd::wValue

Command parameter 1.

Definition at line 46 of file pgpmac.h.

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

- [pgpmac.h](#)

Chapter 5

File Documentation

5.1 lspg.c File Reference

Postgresql support for the LS-CAT pgpmac project. `#include "pgpmac.h"`

Data Structures

- struct [lspgQueryQueueStruct](#)
Store each query along with it's callback function.
- struct [lspg_wait_for_detector_struct](#)
Object that implements detector / spindle timing We use database locks for exposure control and this implements the md2 portion of this handshake.
- struct [lspg_lock_diffractionmeter_struct](#)
Object used to impliment locking the diffractometer Critical to exposure timing.
- struct [lspg_lock_detector_struct](#)
lock detector object Implements detector lock for exposure control
- struct [lspg_seq_run_prep_struct](#)
Data collection running object.

Defines

- `#define` [LS_PG_STATE_INIT](#) -4
- `#define` [LS_PG_STATE_INIT_POLL](#) -3
- `#define` [LS_PG_STATE_RESET](#) -2
- `#define` [LS_PG_STATE_RESET_POLL](#) -1
- `#define` [LS_PG_STATE_IDLE](#) 1
- `#define` [LS_PG_STATE_SEND](#) 2
- `#define` [LS_PG_STATE_SEND_FLUSH](#) 3
- `#define` [LS_PG_STATE_RECV](#) 4
- `#define` [LS_PG_QUERY_QUEUE_LENGTH](#) 16318

Why such a long queue? you might ask.

Typedefs

- typedef struct [lspgQueryQueueStruct](#) [lspg_query_queue_t](#)
Store each query along with it's callback function.
- typedef struct [lspg_wait_for_detector_struct](#) [lspg_wait_for_detector_t](#)
Object that implements detector / spindle timing We use database locks for exposure control and this implements the md2 portion of this handshake.
- typedef struct [lspg_lock_diffractionmeter_struct](#) [lspg_lock_diffractionmeter_t](#)
Object used to impliment locking the diffractometer Critical to exposure timing.
- typedef struct [lspg_lock_detector_struct](#) [lspg_lock_detector_t](#)
lock detector object Implements detector lock for exposure control
- typedef struct [lspg_seq_run_prep_struct](#) [lspg_seq_run_prep_t](#)
Data collection running object.

Functions

- [lspg_query_queue_t](#) * [lspg_query_next](#) ()
Return the next item in the postgresql queue.
- void [lspg_query_reply_next](#) ()
Remove the oldest item in the queue.
- [lspg_query_queue_t](#) * [lspg_query_reply_peek](#) ()
Return the next item in the reply queue but don't pop it since we may need it more than once.
- void [lspg_query_push](#) (void(*cb)([lspg_query_queue_t](#) *, PGresult *), char *fmt,...)
Place a query on the queue.
- void [lspg_init_motors_cb](#) ([lspg_query_queue_t](#) *qqp, PGresult *pgr)
Motor initialization callback.
- void [lspg_zoom_lut_cb](#) ([lspg_query_queue_t](#) *qqp, PGresult *pgr)
Zoom motor look up table callback.
- void [lspg_flight_lut_cb](#) ([lspg_query_queue_t](#) *qqp, PGresult *pgr)
Front Light Lookup table query callback Install the lookup table for the Front Light.
- void [lspg_blight_lut_cb](#) ([lspg_query_queue_t](#) *qqp, PGresult *pgr)
Back Light Lookup Table Callback Install the lookup table for the Back Light.
- void [lspg_nextshot_cb](#) ([lspg_query_queue_t](#) *qqp, PGresult *pgr)

Next Shot Callback.

- void [lspg_nextshot_init](#) ()
Initialize the nextshot variable, mutex, and condition.
- void [lspg_nextshot_call](#) ()
Queue up a nextshot query.
- void [lspg_nextshot_wait](#) ()
Wait for the next shot query to get processed.
- void [lspg_nextshot_done](#) ()
Called when the next shot query has been processed.
- void [lspg_wait_for_detector_init](#) ()
initialize the detector timing object
- void [lspg_wait_for_detector_cb](#) (lspg_query_queue_t *qqp, PGresult *pgr)
Callback for the wait for detector query.
- void [lspg_wait_for_detector_call](#) ()
initiate the wait for detector query
- void [lspg_wait_for_detector_wait](#) ()
Pause the calling thread until the detector is ready Called by the MD2 thread.
- void [lspg_wait_for_detector_done](#) ()
Done waiting for the detector.
- void [lspg_wait_for_detector_all](#) ()
Combined call to wait for the detector.
- void [lspg_lock_diffractionmeter_init](#) ()
initialize the diffractionmeter locking object
- void [lspg_lock_diffractionmeter_cb](#) (lspg_query_queue_t *qqp, PGresult *pgr)
Callback routine for a lock diffractionmeter query.
- void [lspg_lock_diffractionmeter_call](#) ()
Request that the database grab the diffractionmeter lock.
- void [lspg_lock_diffractionmeter_wait](#) ()
Wait for the diffractionmeter lock.
- void [lspg_lock_diffractionmeter_done](#) ()
Finish up the lock diffractionmeter call.
- void [lspg_lock_diffractionmeter_all](#) ()
Convenience function that combines lock diffractionmeter calls.

- void `lspg_lock_detector_init ()`
Initialize detector lock object.
- void `lspg_lock_detector_cb (lspg_query_queue_t *qqp, PGresult *pgr)`
Callback for when the detector lock has be grabbed.
- void `lspg_lock_detector_call ()`
Request (demand) a detector lock.
- void `lspg_lock_detector_wait ()`
Wait for the detector lock.
- void `lspg_lock_detector_done ()`
Finish waiting.
- void `lspg_lock_detector_all ()`
Detector lock convinence function.
- void `lspg_seq_run_prep_init ()`
Initialize the data collection object.
- void `lspg_seq_run_prep_cb (lspg_query_queue_t *qqp, PGresult *pgr)`
Callback for the seq_run_prep query.
- void `lspg_seq_run_prep_call (long long skey, double kappa, double phi, double cx, double cy, double ax, double ay, double az)`
queue up the seq_run_prep query
- void `lspg_seq_run_prep_wait ()`
Wait for seq run prep query to return.
- void `lspg_seq_run_prep_done ()`
Indicate we are done waiting.
- void `lspg_seq_run_prep_all (long long skey, double kappa, double phi, double cx, double cy, double ax, double ay, double az)`
Convinence function to call seq run prep.
- void `lspg_getcenter_cb (lspg_query_queue_t *qqp, PGresult *pgr)`
TODO: implement getcenter code.
- void `lspg_nextaction_cb (lspg_query_queue_t *qqp, PGresult *pgr)`
Queue the next MD2 instruction.
- void `lspg_cmd_cb (lspg_query_queue_t *qqp, PGresult *pgr)`
Send strings directly to PMAC queue.
- void `lspg_flush ()`
Flush psql output buffer (ie, send the query).

- void [lspg_send_next_query](#) ()
send the next queued query to the DB server
- void [lspg_receive](#) ()
Receive a result of a query.
- void [lspg_sig_service](#) (struct pollfd *evt)
Service a signal Signals here are treated as file descriptors and fits into our poll scheme.
- void [lspg_pg_service](#) (struct pollfd *evt)
I/O control to/from the postgresql server.
- void [lspg_pg_connect](#) ()
Connect to the pg server.
- void [lspg_next_state](#) ()
Implements our state machine Does not strictly only set the next state as it also calls some functions that, perhaps, alters the state mid-function.
- void * [lspg_worker](#) (void *dummy)
The main loop for the lspg thread.
- void [lspg_init](#) ()
Initiallize the lspg module.
- void [lspg_run](#) ()
Start 'er runnin'.

Variables

- static int [ls_pg_state](#) = LS_PG_STATE_INIT
State of the lspg state machine.
- static pthread_t [lspg_thread](#)
our worker thread
- static pthread_mutex_t [pg_queue_mutex](#)
keep the queue from getting tangled
- static struct pollfd [lspgfd](#)
our poll info
- static [lspg_query_queue_t](#) [lspg_query_queue](#) [LS_PG_QUERY_QUEUE_LENGTH]
Our query queue.
- static unsigned int [lspg_query_queue_on](#) = 0
Next position to add something to the queue.
- static unsigned int [lspg_query_queue_off](#) = 0

The last item still being used (on == off means nothing in queue).

- static unsigned int [lspg_query_queue_reply](#) = 0

The current item being digested.

- static PGconn * [q](#) = NULL

Database connector.

- static PostgresPollingStatusType [lspg_connectPoll_response](#)

Used to determine state while connecting.

- static PostgresPollingStatusType [lspg_resetPoll_response](#)

Used to determine state while reconnecting.

- [lspg_nextshot_t](#) [lspg_nextshot](#)

the nextshot object

- static [lspg_wait_for_detector_t](#) [lspg_wait_for_detector](#)

Instance of the detector timing object.

- static [lspg_lock_diffractionmeter_t](#) [lspg_lock_diffractionmeter](#)

- static [lspg_lock_detector_t](#) [lspg_lock_detector](#)

- static [lspg_seq_run_prep_t](#) [lspg_seq_run_prep](#)

5.1.1 Detailed Description

Postgresql support for the LS-CAT pgpmac project.

Date:

2012

Author:

Keith Brister All Rights Reserved

Database state machine

State Description

```
-4 Initiate connection
-3 Poll until connection initialization is complete
-2 Initiate reset
-1 Poll until connection reset is complete
 1 Idle (wait for a notify from the server)
 2 Send a query to the server
 3 Continue flushing a command to the server
 4 Waiting for a reply
```

Definition in file [lspg.c](#).

5.1.2 Define Documentation

5.1.2.1 `#define LS_PG_QUERY_QUEUE_LENGTH 16318`

Why such a long queue? you might ask. A huge queue is used here to insure that we don't have to worry too much about over running it. Typically we'll be adding a few queries at a time (for example, to initialize the motors) but not much more than that. When we over run the queue we'll need to look deeply into the root cause as something has gone terribly wrong.

Definition at line 61 of file `lspg.c`.

5.1.2.2 `#define LS_PG_STATE_IDLE 1`

Definition at line 34 of file `lspg.c`.

5.1.2.3 `#define LS_PG_STATE_INIT -4`

Definition at line 30 of file `lspg.c`.

5.1.2.4 `#define LS_PG_STATE_INIT_POLL -3`

Definition at line 31 of file `lspg.c`.

5.1.2.5 `#define LS_PG_STATE_RECV 4`

Definition at line 37 of file `lspg.c`.

5.1.2.6 `#define LS_PG_STATE_RESET -2`

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

5.1.2.7 `#define LS_PG_STATE_RESET_POLL -1`

Definition at line 33 of file `lspg.c`.

5.1.2.8 `#define LS_PG_STATE_SEND 2`

Definition at line 35 of file `lspg.c`.

5.1.2.9 `#define LS_PG_STATE_SEND_FLUSH 3`

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

5.1.3 Typedef Documentation

5.1.3.1 `typedef struct lspg_lock_detector_struct lspg_lock_detector_t`

lock detector object Implements detector lock for exposure control

5.1.3.2 typedef struct lspg_lock_diffractionmeter_struct lspg_lock_diffractionmeter_t

Object used to impliment locking the diffractometer Critical to exposure timing.

5.1.3.3 typedef struct lspgQueryQueueStruct lspg_query_queue_t

Store each query along with it's callback function. All calls are asynchronous

5.1.3.4 typedef struct lspg_seq_run_prep_struct lspg_seq_run_prep_t

Data collection running object.

5.1.3.5 typedef struct lspg_wait_for_detector_struct lspg_wait_for_detector_t

Object that implements detector / spindle timing We use database locks for exposure control and this implements the md2 portion of this handshake.

5.1.4 Function Documentation

5.1.4.1 void lspg_blight_lut_cb (lspg_query_queue_t * *qqp*, PGresult * *pgr*)

Back Light Lookup Table Callback Install the lookup table for the Back Light.

Parameters:

← *qqp* Our query

← *pgr* The query's result

Definition at line 278 of file lspg.c.

```

281                                     {
282     int i;
283
284     pthread_mutex_lock( &(blight->mutex));
285
286     blight->nlut = PQntuples( pgr)/2;
287     blight->lut = calloc( 2*blight->nlut, sizeof(double));
288     if( blight->lut == NULL) {
289         wprintw( term_output, "\nOut of memmory (lspg_blight_lut_cb)");
290         wnoutrefresh( term_output);
291         wnoutrefresh( term_output);
292         doupdate();
293         pthread_mutex_unlock( &(blight->mutex));
294         return;
295     }
296
297     for( i=0; i<PQntuples( pgr); i++) {
298         blight->lut[i] = strtod( PQgetvalue( pgr, i, 0), NULL);
299     }
300
301     pthread_mutex_unlock( &(blight->mutex));
302
303 }
```

5.1.4.2 void lspg_cmd_cb (lspg_query_queue_t * *qqp*, PGresult * *pgr*)

Send strings directly to PMAC queue.

Parameters:

← *qqp* Our query

← *pgr* Our result

Definition at line 895 of file lspg.c.

```

898             {
899         //
900         // Call back function assumes query results in zero or more commands to send to
           the PMAC
901         //
902         int i;
903         char *sp;
904
905         for( i=0; i<PQntuples( pgr); i++) {
906             sp = PQgetvalue( pgr, i, 0);
907             if( sp != NULL && *sp != 0) {
908                 lspmac_SockSendline( sp);
909                 //
910                 // Keep asking for more until
911                 // there are no commands left
912                 //
913                 // This should solve a potential problem where
914                 // more than one command is put on the queue for a given notify.
915                 //
916                 lspg_query_push( lspg_cmd_cb, "select pmac.md2_queue_next()");
917             }
918         }
919     }

```

5.1.4.3 void lspg_flight_lut_cb (lspg_query_queue_t * *qqp*, PGresult * *pgr*)

Front Light Lookup table query callback Install the lookup table for the Front Light.

Parameters:

← *qqp* Our query

← *pgr* Our result object

Definition at line 247 of file lspg.c.

```

250             {
251         int i;
252
253         pthread_mutex_lock( &(amp;flight->mutex));
254
255         flight->nlut = PQntuples( pgr)/2;
256         flight->lut = calloc( 2*flight->nlut, sizeof(double));
257         if( flight->lut == NULL) {
258             wprintw( term_output, "\nOut of memory (lspg_flight_lut_cb)");
259             wnoutrefresh( term_output);
260             wnoutrefresh( term_output);
261             doupdate();
262             pthread_mutex_unlock( &(amp;flight->mutex));
263             return;

```

```

264     }
265
266     for( i=0; i<PQntuples( pgr); i++) {
267         flight->lut[i] = strtod( PQgetvalue( pgr, i, 0), NULL);
268     }
269
270     pthread_mutex_unlock( &(amp;flight->mutex));
271
272 }

```

5.1.4.4 void lspg_flush ()

Flush psql output buffer (ie, send the query).

Definition at line 924 of file lspg.c.

```

924     {
925     int err;
926
927     err = PQflush( q);
928     switch( err) {
929     case -1:
930         // an error occurred
931
932         pthread_mutex_lock( &ncurses_mutex);
933         wprintw( term_output, "\nflush failed: %s\n", PQerrorMessage( q));
934         wnoutrefresh( term_output);
935         wnoutrefresh( term_input);
936         doupdate();
937         pthread_mutex_unlock( &ncurses_mutex);
938
939         ls_pg_state = LS_PG_STATE_IDLE;
940         //
941         // We should probably reset the connection and start from scratch. Probably
942         // the connection died.
943         break;
944
945     case 0:
946         // goodness and joy.
947         ls_pg_state = LS_PG_STATE_RECV;
948         break;
949
950     case 1:
951         // more sending to do
952         ls_pg_state = LS_PG_STATE_SEND_FLUSH;
953         break;
954     }
955 }

```

5.1.4.5 void lspg_getcenter_cb (lspg_query_queue_t * qqp, PGresult * pgr)

TODO: implement getcenter code.

Definition at line 856 of file lspg.c.

```

856     {
857     int theZoom;
858     double dxp, dyp, z, b;
859     // Need camera pixel height and pixel width!
860
861 }

```

5.1.4.6 void lspg_init ()

Initialize the lspg module.

Definition at line 1451 of file lspg.c.

```

1451     {
1452     pthread_mutex_init( &pg_queue_mutex, NULL);
1453     lspg_nextshot_init();
1454     lspg_wait_for_detector_init();
1455     lspg_lock_diffractionmeter_init();
1456     lspg_lock_detector_init();
1457 }

```

5.1.4.7 void lspg_init_motors_cb (lspg_query_queue_t * *qqp*, PGresult * *pgr*)

Motor initialization callback.

Parameters:

← *qqp* The query queue item used to call us

← *pgr* The postgresql result object

Definition at line 167 of file lspg.c.

```

170     {
171     int i, j;
172     uint32_t motor_number, motor_number_column, max_speed_column, max_accel_column
173     ;
174     uint32_t units_column;
175     uint32_t u2c_column;
176     uint32_t format_column;
177     char *sp;
178     lspmac_motor_t *lsdp;
179     motor_number_column = PQfnumber( pgr, "mm_motor");
180     units_column = PQfnumber( pgr, "mm_unit");
181     u2c_column = PQfnumber( pgr, "mm_u2c");
182     format_column = PQfnumber( pgr, "mm_printf");
183     max_speed_column = PQfnumber( pgr, "mm_max_speed");
184     max_accel_column = PQfnumber( pgr, "mm_max_speed");
185
186     if( motor_number_column == -1 || units_column == -1 || u2c_column == -1 || form
187     at_column == -1)
188         return;
189     for( i=0; i<PQntuples( pgr); i++) {
190
191         motor_number = atoi(PQgetvalue( pgr, i, motor_number_column));
192
193         lsdp = NULL;
194         for( j=0; j<lspmac_nmotors; j++) {
195             if( lspmac_motors[j].motor_num == motor_number) {
196                 lsdp = &(lspmac_motors[j]);
197                 lsdp->units = strdup( PQgetvalue( pgr, i, units_column));
198                 lsdp->format= strdup( PQgetvalue( pgr, i, format_column));
199                 lsdp->u2c = atof(PQgetvalue( pgr, i, u2c_column));
200                 lsdp->max_speed = atof(PQgetvalue( pgr, i, max_speed_column));
201                 lsdp->max_accel = atof(PQgetvalue( pgr, i, max_accel_column));
202                 break;
203             }
204         }

```

```

205     if( lsdp == NULL)
206         continue;
207
208
209     if( fabs(lsdp->u2c) <= 1.0e-9)
210         lsdp->u2c = 1.0;
211
212 }
213 }

```

5.1.4.8 void lspg_lock_detector_all ()

Detector lock convinence function.

Definition at line 768 of file lspg.c.

```

768                                     {
769     lspg_lock_detector_call();
770     lspg_lock_detector_wait();
771     lspg_lock_detector_done();
772 }

```

5.1.4.9 void lspg_lock_detector_call ()

Request (demand) a detector lock.

Definition at line 744 of file lspg.c.

```

744                                     {
745     pthread_mutex_lock( &(lspg_lock_detector.mutex));
746     lspg_lock_detector.new_value_ready = 0;
747     pthread_mutex_unlock( &(lspg_lock_detector.mutex));
748
749     lspg_query_push( lspg_lock_detector_cb, "SELECT px.lock_detector()");
750 }

```

5.1.4.10 void lspg_lock_detector_cb (lspg_query_queue_t * qqp, PGresult * pgr)

Callback for when the detector lock has be grabbed.

Definition at line 735 of file lspg.c.

```

735                                     {
736     pthread_mutex_lock( &(lspg_lock_detector.mutex));
737     lspg_lock_detector.new_value_ready = 1;
738     pthread_cond_signal( &(lspg_lock_detector.cond));
739     pthread_mutex_unlock( &(lspg_lock_detector.mutex));
740 }

```

5.1.4.11 void lspg_lock_detector_done ()

Finish waiting.

Definition at line 762 of file lspg.c.

```
762     {
763     pthread_mutex_unlock( &(lspg_lock_detector.mutex));
764 }
```

5.1.4.12 void lspg_lock_detector_init ()

Initialize detector lock object.

Definition at line 727 of file lspg.c.

```
727     {
728     lspg_lock_detector.new_value_ready = 0;
729     pthread_mutex_init( &(lspg_lock_detector.mutex), NULL);
730     pthread_cond_init( &(lspg_lock_detector.cond), NULL);
731 }
```

5.1.4.13 void lspg_lock_detector_wait ()

Wait for the detector lock.

Definition at line 754 of file lspg.c.

```
754     {
755     pthread_mutex_lock( &(lspg_lock_detector.mutex));
756     while( lspg_lock_detector.new_value_ready == 0)
757         pthread_cond_wait( &(lspg_lock_detector.cond), &(lspg_lock_detector.mutex));
758 }
```

5.1.4.14 void lspg_lock_diffractionmeter_all ()

Convenience function that combines lock diffractionmeter calls.

Definition at line 709 of file lspg.c.

```
709     {
710     lspg_lock_diffractionmeter_call();
711     lspg_lock_diffractionmeter_wait();
712     lspg_lock_diffractionmeter_all();
713 }
```

5.1.4.15 void lspg_lock_diffractionmeter_call ()

Request that the database grab the diffractionmeter lock.

Definition at line 685 of file lspg.c.

```
685     {
686     pthread_mutex_lock( &(lspg_lock_diffractionmeter.mutex));
687     lspg_lock_diffractionmeter.new_value_ready = 0;
688     pthread_mutex_unlock( &(lspg_lock_diffractionmeter.mutex));
689
690     lspg_query_push( lspg_lock_diffractionmeter_cb, "SELECT px.lock_diffractionmeter()");
691 }
```

5.1.4.16 void lspg_lock_diffractionmeter_cb (lspg_query_queue_t * *qqp*, PGresult * *pgr*)

Callback routine for a lock diffractionmeter query.

Definition at line 676 of file lspg.c.

```

676                                     {
677     pthread_mutex_lock( &(lspg_lock_diffractionmeter.mutex));
678     lspg_lock_diffractionmeter.new_value_ready = 1;
679     pthread_cond_signal( &(lspg_lock_diffractionmeter.cond));
680     pthread_mutex_unlock( &(lspg_lock_diffractionmeter.mutex));
681 }
```

5.1.4.17 void lspg_lock_diffractionmeter_done ()

Finish up the lock diffractionmeter call.

Definition at line 703 of file lspg.c.

```

703                                     {
704     pthread_mutex_unlock( &(lspg_lock_diffractionmeter.mutex));
705 }
```

5.1.4.18 void lspg_lock_diffractionmeter_init ()

initialize the diffractionmeter locking object

Definition at line 668 of file lspg.c.

```

668                                     {
669     lspg_lock_diffractionmeter.new_value_ready = 0;
670     pthread_mutex_init( &(lspg_lock_diffractionmeter.mutex), NULL);
671     pthread_cond_init( &(lspg_lock_diffractionmeter.cond), NULL);
672 }
```

5.1.4.19 void lspg_lock_diffractionmeter_wait ()

Wait for the diffractionmeter lock.

Definition at line 695 of file lspg.c.

```

695                                     {
696     pthread_mutex_lock( &(lspg_lock_diffractionmeter.mutex));
697     while( lspg_lock_diffractionmeter.new_value_ready == 0)
698         pthread_cond_wait( &(lspg_lock_diffractionmeter.cond), &(
        lspg_lock_diffractionmeter.mutex));
699 }
```

5.1.4.20 void lspg_next_state ()

Implements our state machine Does not strictly only set the next state as it also calls some functions that, perhaps, alters the state mid-function.

Definition at line 1315 of file lspg.c.


```

1315             {
1316             //
1317             // connect to the database
1318             //
1319             if( q == NULL ||
1320                 ls_pg_state == LS_PG_STATE_INIT ||
1321                 ls_pg_state == LS_PG_STATE_RESET ||
1322                 ls_pg_state == LS_PG_STATE_INIT_POLL ||
1323                 ls_pg_state == LS_PG_STATE_RESET_POLL)
1324                 lspg_pg_connect( lspgfd);
1325
1326
1327             if( ls_pg_state == LS_PG_STATE_IDLE && lspg_query_queue_on !=
lspg_query_queue_off)
1328                 ls_pg_state = LS_PG_STATE_SEND;
1329
1330             switch( ls_pg_state) {
1331             case LS_PG_STATE_INIT_POLL:
1332                 if( lspg_connectPoll_response == PGRES_POLLING_WRITING)
1333                     lspgfd.events = POLLOUT;
1334                 else if( lspg_connectPoll_response == PGRES_POLLING_READING)
1335                     lspgfd.events = POLLIN;
1336                 else
1337                     lspgfd.events = 0;
1338                 break;
1339
1340             case LS_PG_STATE_RESET_POLL:
1341                 if( lspg_resetPoll_response == PGRES_POLLING_WRITING)
1342                     lspgfd.events = POLLOUT;
1343                 else if( lspg_resetPoll_response == PGRES_POLLING_READING)
1344                     lspgfd.events = POLLIN;
1345                 else
1346                     lspgfd.events = 0;
1347                 break;
1348
1349             case LS_PG_STATE_IDLE:
1350             case LS_PG_STATE_RECV:
1351                 lspgfd.events = POLLIN;
1352                 break;
1353
1354             case LS_PG_STATE_SEND:
1355             case LS_PG_STATE_SEND_FLUSH:
1356                 lspgfd.events = POLLOUT;
1357                 break;
1358
1359             default:
1360                 lspgfd.events = 0;
1361             }
1362 }

```

5.1.4.21 void lspg_nextaction_cb (lspg_query_queue_t * *qqp*, PGresult * *pgr*)

Queue the next MD2 instruction.

Parameters:

- ← *qqp* The query that generated this result
- ← *pgr* The result

Definition at line 865 of file lspg.c.

```

868             {
869             char *action;

```

```

870
871     if( PQntuples( pgr) <= 0)
872         return;          // Note: nextaction should always return at least "noAction", so this branch should never be taken
873
874     action = PQgetvalue( pgr, 0, 0);          // next action only returns one row
875
876     if( strcmp( action, "noAction") == 0)
877         return;
878
879     if( pthread_mutex_trylock( &md2cmds_mutex) == 0) {
880         strncpy( md2cmds_cmd, action, MD2CMDS_CMD_LENGTH-1);
881         md2cmds_cmd[MD2CMDS_CMD_LENGTH-1] = 0;
882         pthread_cond_signal( &md2cmds_cond);
883         pthread_mutex_unlock( &md2cmds_mutex);
884     } else {
885         //
886         // TODO:
887         // We should probably report that we aren't going to act
888         // on the requested action. That code would go here.
889         //
890     }
891 }

```

5.1.4.22 void lspg_nextshot_call ()

Queue up a nextshot query.

Definition at line 568 of file lspg.c.

```

568     {
569         pthread_mutex_lock( &(lspg_nextshot.mutex));
570         lspg_nextshot.new_value_ready = 0;
571         pthread_mutex_unlock( &(lspg_nextshot.mutex));
572
573         lspg_query_push( lspg_nextshot_cb, "SELECT * FROM px.nextshot()");
574     }

```

5.1.4.23 void lspg_nextshot_cb (lspg_query_queue_t * qqp, PGresult * pgr)

Next Shot Callback. This is a long and tedious routine as there are a large number of variables returned. Suck it up. Return with the global variable lspg_nextshot set.

Parameters:

← **qqp** Our nextshot query

← **pgr** result of the query

Definition at line 313 of file lspg.c.

```

316     {
317         static int got_col_nums=0;
318         static int
319             dsdir_c, dspid_c, dsowidth_c, dsoscaxis_c, dsexp_c, skey_c, sstart_c, sfnc_c,
320             dsphi_c,
321             dsomega_c, dskappa_c, dsdist_c, dsnrg_c, dshpid_c, cx_c, cy_c, ax_c, ay_c, az_c,
322             active_c, sindex_c, stype_c,
323             dsowidth2_c, dsoscaxis2_c, dsexp2_c, sstart2_c, dsphi2_c, dsomega2_c, dskappa

```

```

2_c, dsdist2_c, dsnr2_c,
323     cx2_c, cy2_c, ax2_c, ay2_c, az2_c, active2_c, sindex2_c, stype2_c;
324
325 pthread_mutex_lock( &(lspg_nextshot.mutex));
326
327 lspg_nextshot.no_rows_returned = PQntuples( pgr) <= 0;
328 if( lspg_nextshot.no_rows_returned) {
329     lspg_nextshot.new_value_ready = 1;
330     pthread_cond_signal( &(lspg_nextshot.cond));
331     pthread_mutex_unlock( &(lspg_nextshot.mutex));
332     return; // I guess there was no shot after all
333 }
334
335 if( got_col_nums == 0) {
336     dsdir_c = PQfnumber( pgr, "dsdir");
337     dspid_c = PQfnumber( pgr, "dspid");
338     dsowidth_c = PQfnumber( pgr, "dsowidth");
339     dsoscaxis_c = PQfnumber( pgr, "dsoscaxis");
340     dsexp_c = PQfnumber( pgr, "dsexp");
341     skey_c = PQfnumber( pgr, "skey");
342     sstart_c = PQfnumber( pgr, "sstart");
343     sfnc_c = PQfnumber( pgr, "sfnc");
344     dsphi_c = PQfnumber( pgr, "dsphi");
345     dsomega_c = PQfnumber( pgr, "dsomega");
346     dskappa_c = PQfnumber( pgr, "dskappa");
347     dsdist_c = PQfnumber( pgr, "dsdist");
348     dsnr2_c = PQfnumber( pgr, "dsnr2");
349     dshpid_c = PQfnumber( pgr, "dshpid");
350     cx_c = PQfnumber( pgr, "cx");
351     cy_c = PQfnumber( pgr, "cy");
352     ax_c = PQfnumber( pgr, "ax");
353     ay_c = PQfnumber( pgr, "ay");
354     az_c = PQfnumber( pgr, "az");
355     active_c = PQfnumber( pgr, "active");
356     sindex_c = PQfnumber( pgr, "sindex");
357     stype_c = PQfnumber( pgr, "stype");
358     dsowidth2_c = PQfnumber( pgr, "dsowidth2");
359     dsoscaxis2_c = PQfnumber( pgr, "dsoscaxis2");
360     dsexp2_c = PQfnumber( pgr, "dsexp2");
361     sstart2_c = PQfnumber( pgr, "sstart2");
362     dsphi2_c = PQfnumber( pgr, "dsphi2");
363     dsomega2_c = PQfnumber( pgr, "dsomega2");
364     dskappa2_c = PQfnumber( pgr, "dskappa2");
365     dsdist2_c = PQfnumber( pgr, "dsdist2");
366     dsnr2_c = PQfnumber( pgr, "dsnr2");
367     cx2_c = PQfnumber( pgr, "cx2");
368     cy2_c = PQfnumber( pgr, "cy2");
369     ax2_c = PQfnumber( pgr, "ax2");
370     ay2_c = PQfnumber( pgr, "ay2");
371     az2_c = PQfnumber( pgr, "az2");
372     active2_c = PQfnumber( pgr, "active2");
373     sindex2_c = PQfnumber( pgr, "sindex2");
374     stype2_c = PQfnumber( pgr, "stype2");
375
376     got_col_nums = 1;
377 }
378
379
380 //
381 // NULL string values come back as empty strings
382 // Mark the null flag but allocate the empty string anyway
383 //
384
385 lspg_nextshot.dsdir_isnull = PQgetisnull( pgr, 0, dsdir_c);
386 if( lspg_nextshot.dsdir != NULL)
387     free( lspg_nextshot.dsdir);
388 lspg_nextshot.dsdir = strdup( PQgetvalue( pgr, 0, dsdir_c));

```

```

389
390  lspg_nextshot.dspid_isnull = PQgetisnull( pgr, 0, dspid_c);
391  if( lspg_nextshot.dspid != NULL)
392      free( lspg_nextshot.dspid);
393  lspg_nextshot.dspid = strdup( PQgetvalue( pgr, 0, dspid_c));
394
395  lspg_nextshot.dsoscaxis_isnull = PQgetisnull( pgr, 0, dsoscaxis_c);
396  if( lspg_nextshot.dsoscaxis != NULL)
397      free( lspg_nextshot.dsoscaxis);
398  lspg_nextshot.dsoscaxis = strdup( PQgetvalue( pgr, 0, dsoscaxis_c));
399
400  lspg_nextshot.dsoscaxis2_isnull = PQgetisnull( pgr, 0, dsoscaxis2_c);
401  if( lspg_nextshot.dsoscaxis2 != NULL)
402      free( lspg_nextshot.dsoscaxis2);
403  lspg_nextshot.dsoscaxis2 = strdup( PQgetvalue( pgr, 0, dsoscaxis2_c));
404
405  lspg_nextshot.sfn_isnull = PQgetisnull(pgr, 0, sfn_c);
406  if( lspg_nextshot.sfn != NULL)
407      free( lspg_nextshot.sfn);
408  lspg_nextshot.sfn = strdup( PQgetvalue( pgr, 0, sfn_c));
409
410  lspg_nextshot.stype_isnull = PQgetisnull( pgr, 0, stype_c);
411  if( lspg_nextshot.stype != NULL)
412      free( lspg_nextshot.stype);
413  lspg_nextshot.stype = strdup( PQgetvalue( pgr, 0, stype_c));
414
415  lspg_nextshot.stype2_isnull = PQgetisnull( pgr, 0, stype2_c);
416  if( lspg_nextshot.stype2 != NULL)
417      free( lspg_nextshot.stype2);
418  lspg_nextshot.stype2 = strdup( PQgetvalue( pgr, 0, stype2_c));
419
420  //
421  // Probably shouldn't try to convert null number values
422  //
423  lspg_nextshot.dsowidth_isnull = PQgetisnull( pgr, 0, dsowidth_c);
424  if( lspg_nextshot.dsowidth_isnull == 0)
425      lspg_nextshot.dsowidth = atof( PQgetvalue( pgr,0, dsowidth_c));
426
427  lspg_nextshot.dsexp_isnull = PQgetisnull( pgr, 0, dsexp_c);
428  if( lspg_nextshot.dsexp_isnull == 0)
429      lspg_nextshot.dsexp = atof( PQgetvalue( pgr,0, dsexp_c));
430
431  lspg_nextshot.sstart_isnull = PQgetisnull( pgr, 0, sstart_c);
432  if( lspg_nextshot.sstart_isnull == 0)
433      lspg_nextshot.sstart = atof( PQgetvalue( pgr,0, sstart_c));
434
435  lspg_nextshot.dsphi_isnull = PQgetisnull( pgr, 0, dsphi_c);
436  if( lspg_nextshot.dsphi_isnull == 0)
437      lspg_nextshot.dsphi = atof( PQgetvalue( pgr,0, dsphi_c));
438
439  lspg_nextshot.dsomega_isnull = PQgetisnull( pgr, 0, dsomega_c);
440  if( lspg_nextshot.dsomega_isnull == 0)
441      lspg_nextshot.dsomega = atof( PQgetvalue( pgr,0, dsomega_c));
442
443  lspg_nextshot.dskappa_isnull = PQgetisnull( pgr, 0, dskappa_c);
444  if( lspg_nextshot.dskappa_isnull == 0)
445      lspg_nextshot.dskappa = atof( PQgetvalue( pgr,0, dskappa_c));
446
447  lspg_nextshot.dsdist_isnull = PQgetisnull( pgr, 0, dsdist_c);
448  if( lspg_nextshot.dsdist_isnull == 0)
449      lspg_nextshot.dsdist = atof( PQgetvalue( pgr,0, dsdist_c));
450
451  lspg_nextshot.dsnrg_isnull = PQgetisnull( pgr, 0, dsnrg_c);
452  if( lspg_nextshot.dsnrg_isnull == 0)
453      lspg_nextshot.dsnrg = atof( PQgetvalue( pgr,0, dsnrg_c));
454
455  lspg_nextshot.cx_isnull = PQgetisnull( pgr, 0, cx_c);

```

```
456 if( lspg_nextshot.cx_isnull == 0)
457     lspg_nextshot.cx      = atof( PQgetvalue( pgr,0, cx_c));
458
459 lspg_nextshot.cy_isnull = PQgetisnull( pgr, 0, cy_c);
460 if( lspg_nextshot.cy_isnull == 0)
461     lspg_nextshot.cy      = atof( PQgetvalue( pgr,0, cy_c));
462
463 lspg_nextshot.ax_isnull = PQgetisnull( pgr, 0, ax_c);
464 if( lspg_nextshot.ax_isnull == 0)
465     lspg_nextshot.ax      = atof( PQgetvalue( pgr,0, ax_c));
466
467 lspg_nextshot.ay_isnull = PQgetisnull( pgr, 0, ay_c);
468 if( lspg_nextshot.ay_isnull == 0)
469     lspg_nextshot.ay      = atof( PQgetvalue( pgr,0, ay_c));
470
471 lspg_nextshot.az_isnull = PQgetisnull( pgr, 0, az_c);
472 if( lspg_nextshot.az_isnull == 0)
473     lspg_nextshot.az      = atof( PQgetvalue( pgr,0, az_c));
474
475 lspg_nextshot.active_isnull = PQgetisnull( pgr, 0, active_c);
476 if( lspg_nextshot.active_isnull == 0)
477     lspg_nextshot.active = atoi( PQgetvalue( pgr, 0, active_c));
478
479 lspg_nextshot.sindex_isnull = PQgetisnull( pgr, 0, sindex_c);
480 if( lspg_nextshot.sindex_isnull == 0)
481     lspg_nextshot.sindex = atoi( PQgetvalue( pgr, 0, sindex_c));
482
483 lspg_nextshot.dshpid_isnull = PQgetisnull( pgr, 0, dshpid_c);
484 if( lspg_nextshot.dshpid_isnull == 0)
485     lspg_nextshot.dshpid = atoi( PQgetvalue( pgr, 0, dshpid_c));
486
487 lspg_nextshot.skey_isnull = PQgetisnull( pgr, 0, skey_c);
488 if( lspg_nextshot.skey_isnull == 0)
489     lspg_nextshot.skey   = atoll( PQgetvalue( pgr, 0, skey_c));
490
491 lspg_nextshot.dsowidth2_isnull = PQgetisnull( pgr, 0, dsowidth2_c);
492 if( lspg_nextshot.dsowidth2_isnull == 0)
493     lspg_nextshot.dsowidth2 = atof( PQgetvalue( pgr,0, dsowidth2_c));
494
495 lspg_nextshot.dsexp2_isnull = PQgetisnull( pgr, 0, dsexp2_c);
496 if( lspg_nextshot.dsexp2_isnull == 0)
497     lspg_nextshot.dsexp2   = atof( PQgetvalue( pgr,0, dsexp2_c));
498
499 lspg_nextshot.sstart2_isnull = PQgetisnull( pgr, 0, sstart2_c);
500 if( lspg_nextshot.sstart2_isnull == 0)
501     lspg_nextshot.sstart2   = atof( PQgetvalue( pgr,0, sstart2_c));
502
503 lspg_nextshot.dsphi2_isnull = PQgetisnull( pgr, 0, dsphi2_c);
504 if( lspg_nextshot.dsphi2_isnull == 0)
505     lspg_nextshot.dsphi2    = atof( PQgetvalue( pgr,0, dsphi2_c));
506
507 lspg_nextshot.dsomega2_isnull = PQgetisnull( pgr, 0, dsomega2_c);
508 if( lspg_nextshot.dsomega2_isnull == 0)
509     lspg_nextshot.dsomega2  = atof( PQgetvalue( pgr,0, dsomega2_c));
510
511 lspg_nextshot.dskappa2_isnull = PQgetisnull( pgr, 0, dskappa2_c);
512 if( lspg_nextshot.dskappa2_isnull == 0)
513     lspg_nextshot.dskappa2  = atof( PQgetvalue( pgr,0, dskappa2_c));
514
515 lspg_nextshot.dsdist2_isnull = PQgetisnull( pgr, 0, dsdist2_c);
516 if( lspg_nextshot.dsdist2_isnull == 0)
517     lspg_nextshot.dsdist2   = atof( PQgetvalue( pgr,0, dsdist2_c));
518
519 lspg_nextshot.dsnrg2_isnull = PQgetisnull( pgr, 0, dsnrg2_c);
520 if( lspg_nextshot.dsnrg2_isnull == 0)
521     lspg_nextshot.dsnrg2    = atof( PQgetvalue( pgr,0, dsnrg2_c));
522
```

```

523  lspg_nextshot.cx2_isnull = PQgetisnull( pgr, 0, cx2_c);
524  if( lspg_nextshot.cx2_isnull == 0)
525      lspg_nextshot.cx2      = atof( PQgetvalue( pgr,0, cx2_c));
526
527  lspg_nextshot.cy2_isnull = PQgetisnull( pgr, 0, cy2_c);
528  if( lspg_nextshot.cy2_isnull == 0)
529      lspg_nextshot.cy2      = atof( PQgetvalue( pgr,0, cy2_c));
530
531  lspg_nextshot.ax2_isnull = PQgetisnull( pgr, 0, ax2_c);
532  if( lspg_nextshot.ax2_isnull == 0)
533      lspg_nextshot.ax2      = atof( PQgetvalue( pgr,0, ax2_c));
534
535  lspg_nextshot.ay2_isnull = PQgetisnull( pgr, 0, ay2_c);
536  if( lspg_nextshot.ay2_isnull == 0)
537      lspg_nextshot.ay2      = atof( PQgetvalue( pgr,0, ay2_c));
538
539  lspg_nextshot.az2_isnull = PQgetisnull( pgr, 0, az2_c);
540  if( lspg_nextshot.az2_isnull == 0)
541      lspg_nextshot.az2      = atof( PQgetvalue( pgr,0, az2_c));
542
543  lspg_nextshot.active2_isnull = PQgetisnull( pgr, 0, active2_c);
544  if( lspg_nextshot.active2_isnull == 0)
545      lspg_nextshot.active2 = atoi( PQgetvalue( pgr, 0, active2_c));
546
547  lspg_nextshot.sindex2_isnull = PQgetisnull( pgr, 0, sindex2_c);
548  if( lspg_nextshot.sindex2_isnull == 0)
549      lspg_nextshot.sindex2 = atoi( PQgetvalue( pgr, 0, sindex2_c));
550
551  lspg_nextshot.new_value_ready = 1;
552
553  pthread_cond_signal( &(lspg_nextshot.cond));
554  pthread_mutex_unlock( &(lspg_nextshot.mutex));
555
556 }

```

5.1.4.24 void lspg_nextshot_done ()

Called when the next shot query has been processed.

Definition at line 586 of file lspg.c.

```

586      {
587  pthread_mutex_unlock( &(lspg_nextshot.mutex));
588 }

```

5.1.4.25 void lspg_nextshot_init ()

Initialize the nextshot variable, mutex, and condition.

Definition at line 560 of file lspg.c.

```

560      {
561  memset( &lspg_nextshot, 0, sizeof( lspg_nextshot));
562  pthread_mutex_init( &(lspg_nextshot.mutex), NULL);
563  pthread_cond_init( &(lspg_nextshot.cond), NULL);
564 }

```

5.1.4.26 void lspg_nextshot_wait ()

Wait for the next shot query to get processed.

Definition at line 578 of file lspg.c.

```

578         {
579     pthread_mutex_lock( &(lspg_nextshot.mutex));
580     while( lspg_nextshot.new_value_ready == 0)
581         pthread_cond_wait( &(lspg_nextshot.cond), &(lspg_nextshot.mutex));
582 }
```

5.1.4.27 void lspg_pg_connect ()

Connect to the pg server.

Definition at line 1214 of file lspg.c.

```

1214         {
1215     PGresult *pgr;
1216     int wait_interval = 1;
1217     int connection_init = 0;
1218     int i, err;
1219
1220     if( q == NULL)
1221         ls_pg_state = LS_PG_STATE_INIT;
1222
1223     switch( ls_pg_state) {
1224     case LS_PG_STATE_INIT:
1225         q = PQconnectStart( "dbname=ls user=lsuser hostaddr=10.1.0.3");
1226         if( q == NULL) {
1227             pthread_mutex_lock( &ncurses_mutex);
1228             wprintw( term_output, "Out of memory (lspg_pg_connect)\n");
1229             wnoutrefresh( term_output);
1230             wnoutrefresh( term_input);
1231             doupdate();
1232             pthread_mutex_unlock( &ncurses_mutex);
1233             exit( -1);
1234         }
1235
1236         err = PQstatus( q);
1237         if( err == CONNECTION_BAD) {
1238             pthread_mutex_lock( &ncurses_mutex);
1239             wprintw( term_output, "Trouble connecting to database\n");
1240             wnoutrefresh( term_output);
1241             wnoutrefresh( term_input);
1242             doupdate();
1243             pthread_mutex_unlock( &ncurses_mutex);
1244             //
1245             // TODO: save time of day so we can check that we are not retrying the conn
1246             //
1247             return;
1248         }
1249         err = PQsetnonblocking( q, 1);
1250         if( err != 0) {
1251             pthread_mutex_lock( &ncurses_mutex);
1252             wprintw( term_output, "Odd, could not set database connection to nonblockin
1253             g\n");
1254             wnoutrefresh( term_output);
1255             wnoutrefresh( term_input);
1256             doupdate();
1257             pthread_mutex_unlock( &ncurses_mutex);

```

```

1257     }
1258
1259     lspg_pg_state = LS_PG_STATE_INIT_POLL;
1260     lspg_connectPoll_response = PGRES_POLLING_WRITING;
1261     //
1262     // set up the connection for poll
1263     //
1264     lspgfd.fd = PQsocket( q);
1265     break;
1266
1267 case LS_PG_STATE_INIT_POLL:
1268     if( lspg_connectPoll_response == PGRES_POLLING_FAILED) {
1269         PQfinish( q);
1270         q = NULL;
1271         lspg_pg_state = LS_PG_STATE_INIT;
1272     } else if( lspg_connectPoll_response == PGRES_POLLING_OK) {
1273         lspg_query_push( lspg_init_motors_cb, "select * from pmac.md2_getmotors()")
1274     ;
1275     lspg_query_push( NULL, "select pmac.md2_init()");
1276     lspg_query_push( lspg_zoom_lut_cb, "SELECT * FROM pmac.md2_zoom_lut()");
1277     lspg_query_push( lspg_flight_lut_cb, "SELECT * FROM pmac.md2_flight_lut()")
1278 ;
1279     lspg_query_push( lspg_blight_lut_cb, "SELECT * FROM pmac.md2_blight_lut()")
1280 ;
1281     lspg_pg_state = LS_PG_STATE_IDLE;
1282     }
1283     break;
1284
1285 case LS_PG_STATE_RESET:
1286     err = PQresetStart( q);
1287     if( err == 0) {
1288         PQfinish( q);
1289         q = NULL;
1290         lspg_pg_state = LS_PG_STATE_INIT;
1291     } else {
1292         lspg_pg_state = LS_PG_STATE_RESET_POLL;
1293         lspg_resetPoll_response = PGRES_POLLING_WRITING;
1294     }
1295     break;
1296
1297 case LS_PG_STATE_RESET_POLL:
1298     if( lspg_resetPoll_response == PGRES_POLLING_FAILED) {
1299         PQfinish( q);
1300         q = NULL;
1301         lspg_pg_state = LS_PG_STATE_INIT;
1302     } else if( lspg_resetPoll_response == PGRES_POLLING_OK) {
1303         lspg_query_push( lspg_init_motors_cb, "select * from pmac.md2_getmotors()")
1304     ;
1305     lspg_query_push( NULL, "select pmac.md2_init()");
1306     lspg_pg_state = LS_PG_STATE_IDLE;
1307     }
1308     break;
1309 }

```

5.1.4.28 void lspg_pg_service (struct pollfd * evt)

I/O control to/from the postgresql server.

Parameters:

← *evt* The pollfd object that we are responding to

Definition at line 1109 of file lspg.c.


```

1111         {
1112         //
1113         // Currently just used to check for notifies
1114         // Other socket communication is done synchronously
1115         // Reconsider this if we start using the pmac gather functions
1116         // since we'll want to be servicing those sockets ASAP
1117         //
1118
1119         if( evt->revents & POLLIN) {
1120             int err;
1121
1122             if( ls_pg_state == LS_PG_STATE_INIT_POLL) {
1123                 lspg_connectPoll_response = PQconnectPoll( q);
1124                 if( lspg_connectPoll_response == PGRES_POLLING_FAILED) {
1125                     ls_pg_state = LS_PG_STATE_RESET;
1126                 }
1127                 return;
1128             }
1129
1130             if( ls_pg_state == LS_PG_STATE_RESET_POLL) {
1131                 lspg_resetPoll_response = PQresetPoll( q);
1132                 if( lspg_resetPoll_response == PGRES_POLLING_FAILED) {
1133                     ls_pg_state = LS_PG_STATE_RESET;
1134                 }
1135                 return;
1136             }
1137
1138
1139             //
1140             // if in IDLE or RECV we need to call consumeInput first
1141             //
1142             if( ls_pg_state == LS_PG_STATE_IDLE) {
1143                 err = PQconsumeInput( q);
1144                 if( err != 1) {
1145                     pthread_mutex_lock( &ncurses_mutex);
1146                     wprintw( term_output, "\nconsume input failed: %s\n", PQerrorMessage( q))
;
1147                     wnoutrefresh( term_output);
1148                     wnoutrefresh( term_input);
1149                     doupdate();
1150                     pthread_mutex_unlock( &ncurses_mutex);
1151                     ls_pg_state == LS_PG_STATE_RESET;
1152                     return;
1153                 }
1154             }
1155
1156             if( ls_pg_state == LS_PG_STATE_RECV) {
1157                 lspg_receive();
1158             }
1159
1160             //
1161             // Check for notifies regardless of our state
1162             // Push as many requests as we have notifies.
1163             //
1164             {
1165                 PGnotify *pgn;
1166
1167                 while( 1) {
1168                     pgn = PQnotifies( q);
1169                     if( pgn == NULL)
1170                         break;
1171
1172                     if( strstr( pgn->relname, "_pmac") != NULL) {
1173                         lspg_query_push( lspg_cmd_cb, "SELECT pmac.md2_queue_next()");
1174                     } else {
1175                         lspg_query_push( lspg_nextaction_cb, "SELECT action FROM px.nextaction(
");

```

```

1176     }
1177     PQfreemem( pgn);
1178 }
1179 }
1180 }
1181
1182 if( evt->revents & POLLOUT) {
1183
1184     if( ls_pg_state == LS_PG_STATE_INIT_POLL) {
1185         lspg_connectPoll_response = PQconnectPoll( q);
1186         if( lspg_connectPoll_response == PGRES_POLLING_FAILED) {
1187             ls_pg_state = LS_PG_STATE_RESET;
1188         }
1189         return;
1190     }
1191
1192     if( ls_pg_state == LS_PG_STATE_RESET_POLL) {
1193         lspg_resetPoll_response = PQresetPoll( q);
1194         if( lspg_resetPoll_response == PGRES_POLLING_FAILED) {
1195             ls_pg_state = LS_PG_STATE_RESET;
1196         }
1197         return;
1198     }
1199
1200
1201     if( ls_pg_state == LS_PG_STATE_SEND) {
1202         lspg_send_next_query();
1203     }
1204
1205     if( ls_pg_state == LS_PG_STATE_SEND_FLUSH) {
1206         lspg_flush();
1207     }
1208 }
1209 }

```

5.1.4.29 `lspg_query_queue_t* lspg_query_next()`

Return the next item in the postgresql queue. If there is an item left in the queue then it is returned. Otherwise, NULL is returned.

Definition at line 80 of file `lspg.c`.

```

80                                     {
81     lspg_query_queue_t *rtn;
82
83     pthread_mutex_lock( &pg_queue_mutex);
84
85     if( lspg_query_queue_off == lspg_query_queue_on)
86         // Queue is empty
87         rtn = NULL;
88     else
89         rtn = &(lspg_query_queue[(lspg_query_queue_off++) % LS_PG_QUERY_QUEUE_LENGTH]
90     );
91     pthread_mutex_unlock( &pg_queue_mutex);
92     return rtn;
93 }

```

5.1.4.30 `void lspg_query_push (void(*) (lspg_query_queue_t *, PGresult *) cb, char *fmt, ...)`

Place a query on the queue.

Parameters:

← *cb* Our callback function that deals with the response

← *fmt* Printf style function to generate the query

Definition at line 131 of file lspg.c.

```

135             {
136     int idx;
137     va_list arg_ptr;
138
139     pthread_mutex_lock( &pg_queue_mutex);
140
141     //
142     // TODO
143     //
144     // Should really wait until there is enough room on the queue.
145     // Although the queue is big it is not infinite, so one day we'll over run it.
146     // Should really test to see if (on + 1) == off. If so, then use pg_queue_cond
147     // to
148     // wait until some room has been cleared.
149     //
150     idx = lspg_query_queue_on % LS_PG_QUERY_QUEUE_LENGTH;
151
152     va_start( arg_ptr, fmt);
153     vsnprintf( lspg_query_queue[idx].qs, LS_PG_QUERY_STRING_LENGTH-1, fmt, arg_ptr)
154     ;
155     va_end( arg_ptr);
156
157     lspg_query_queue[idx].qs[LS_PG_QUERY_STRING_LENGTH - 1] = 0;
158     lspg_query_queue[idx].onResponse = cb;
159     lspg_query_queue_on++;
160
161     pthread_kill( lspg_thread, SIGUSR1);
162     pthread_mutex_unlock( &pg_queue_mutex);
163 };

```

5.1.4.31 void lspg_query_reply_next ()

Remove the oldest item in the queue. this is called only when there is nothing else to service the reply: this pop does not return anything. We use the ...reply_peek function to return the next item in the reply queue

Definition at line 102 of file lspg.c.

```

102             {
103
104     pthread_mutex_lock( &pg_queue_mutex);
105
106     if( lspg_query_queue_reply != lspg_query_queue_on)
107         lspg_query_queue_reply++;
108
109     pthread_mutex_unlock( &pg_queue_mutex);
110 }

```

5.1.4.32 lspg_query_queue_t* lspg_query_reply_peek ()

Return the next item in the reply queue but don't pop it since we may need it more than once. Call [lspg_query_reply_next\(\)](#) when done.

Definition at line 115 of file lspg.c.

```

115                                     {
116     lspg_query_queue_t *rtn;
117
118     pthread_mutex_lock( &pg_queue_mutex);
119
120     if( lspg_query_queue_reply == lspg_query_queue_on)
121         rtn = NULL;
122     else
123         rtn = &(lspg_query_queue[(lspg_query_queue_reply) % LS_PG_QUERY_QUEUE_LENGTH]
124         );
125     pthread_mutex_unlock( &pg_queue_mutex);
126     return rtn;
127 }

```

5.1.4.33 void lspg_receive ()

Receive a result of a query.

Definition at line 1016 of file lspg.c.

```

1016                                     {
1017     PGresult *pgr;
1018     lspg_query_queue_t *qgp;
1019     int err;
1020
1021     err = PQconsumeInput( q);
1022     if( err != 1) {
1023         pthread_mutex_lock( &ncurses_mutex);
1024         wprintw( term_output, "\nconsume input failed: %s\n", PQerrorMessage( q));
1025         wnoutrefresh( term_output);
1026         wnoutrefresh( term_input);
1027         doupdate();
1028         pthread_mutex_unlock( &ncurses_mutex);
1029         ls_pg_state == LS_PG_STATE_RESET;
1030         return;
1031     }
1032
1033     //
1034     // We must call PQgetResult until it returns NULL before sending the next query
1035
1036     // This implies that only one query can ever be active at a time and our queue
1037     // management should be simple
1038     // We should be in the LS_PG_STATE_RECV here
1039     //
1040
1041     while( !PQisBusy( q)) {
1042         pgr = PQgetResult( q);
1043         if( pgr == NULL) {
1044             lspg_query_reply_next();
1045             //
1046             // we are now done reading the response from the database
1047             //
1048             ls_pg_state = LS_PG_STATE_IDLE;
1049             break;
1050         } else {
1051             ExecStatusType es;
1052
1053             qgp = lspg_query_reply_peek();
1054             es = PQresultStatus( pgr);
1055
1056             if( es != PGRES_COMMAND_OK && es != PGRES_TUPLES_OK) {
1057                 char *emess;

```

```

1058         emess = PQresultErrorMessage( pgr);
1059         if( emess != NULL && emess[0] != 0) {
1060             pthread_mutex_lock( &ncurses_mutex);
1061             wprintw( term_output, "\nError from query '%s':\n%s\n", qqp->qs, emess)
1062         ;
1063             wnoutrefresh( term_output);
1064             wnoutrefresh( term_input);
1065             doupdate();
1066             pthread_mutex_unlock( &ncurses_mutex);
1067         } else {
1068             //
1069             // Deal with the response
1070             //
1071             // If the response is likely to take awhile we should probably
1072             // add a new state and put something in the main loop to run the onRespon
se
1073             // routine in the main loop. For now, though, we only expect very brief
onResponse routines
1074             //
1075             if( qqp != NULL && qqp->onResponse != NULL)
1076                 qqp->onResponse( qqp, pgr);
1077         }
1078         PQclear( pgr);
1079     }
1080 }
1081 }

```

5.1.4.34 void lspg_run ()

Start 'er runnin'.

Definition at line 1461 of file lspg.c.

```

1461     {
1462     pthread_create( &lspg_thread, NULL, lspg_worker, NULL);
1463 }

```

5.1.4.35 void lspg_send_next_query ()

send the next queued query to the DB server

Definition at line 959 of file lspg.c.

```

959     {
960     //
961     // Normally we should be in the "send" state
962     // but we can also send if we are servicing
963     // a reply
964     //
965     //
966     lspg_query_queue_t *qqp;
967     int err;
968     //
969     qqp = lspg_query_next();
970     if( qqp == NULL) {
971         //
972         // A send without a query? Should never happen.
973         // But at least we shouldn't segfault if it does.
974         //
975         return;

```

```

976     }
977
978     if( qqp->qs[0] == 0) {
979         //
980         // Do we really have to check this case?
981         // It would only come up if we stupidly pushed an empty query string
982         // or ran off the end of the queue
983         //
984         pthread_mutex_lock( &ncurses_mutex);
985         wprintw( term_output, "\nPopped empty query string.  Probably bad things are
going on.\n");
986         wnoutrefresh( term_output);
987         wnoutrefresh( term_input);
988         doupdate();
989         pthread_mutex_unlock( &ncurses_mutex);
990
991         lspg_query_reply_next();
992         ls_pg_state = LS_PG_STATE_IDLE;
993     } else {
994         err = PQsendQuery( q, qqp->qs);
995         if( err == 0) {
996             pthread_mutex_lock( &ncurses_mutex);
997             wprintw( term_output, "\nquery failed: %s\n", PQerrorMessage( q));
998             wnoutrefresh( term_output);
999             wnoutrefresh( term_input);
1000             doupdate();
1001             pthread_mutex_unlock( &ncurses_mutex);
1002
1003             //
1004             // Don't wait for a reply, just reset the connection
1005             //
1006             lspg_query_reply_next();
1007             ls_pg_state == LS_PG_STATE_RESET;
1008         } else {
1009             ls_pg_state = LS_PG_STATE_SEND_FLUSH;
1010         }
1011     }
1012 }

```

5.1.4.36 void lspg_seq_run_prep_all (long long *skey*, double *kappa*, double *phi*, double *cx*, double *cy*, double *ax*, double *ay*, double *az*)

Convenience function to call seq run prep.

Parameters:

- ← *skey* px.shots key for this image
- ← *kappa* current kappa postion
- ← *phi* current phi postition
- ← *cx* current center table x
- ← *cy* current center table y
- ← *ax* current alignment table x
- ← *ay* current alignment table y
- ← *az* current alignment table z

Definition at line 839 of file lspg.c.

```

848                                     {

```

```

849  lspg_seq_run_prep_call( skey, kappa, phi, cx, cy, ax, ay, az);
850  lspg_seq_run_prep_wait();
851  lspg_seq_run_prep_done();
852 }

```

5.1.4.37 void `lspg_seq_run_prep_call` (long long *skey*, double *kappa*, double *phi*, double *cx*, double *cy*, double *ax*, double *ay*, double *az*)

queue up the `seq_run_prep` query

Parameters:

- ← *skey* px.shots key for this image
- ← *kappa* current kappa postion
- ← *phi* current phi postition
- ← *cx* current center table x
- ← *cy* current center table y
- ← *ax* current alignment table x
- ← *ay* current alignment table y
- ← *az* current alignment table z

Definition at line 805 of file `lspg.c`.

```

814  {
815  pthread_mutex_lock( &(lspg_seq_run_prep.mutex));
816  lspg_seq_run_prep.new_value_ready = 0;
817  pthread_mutex_unlock( &(lspg_seq_run_prep.mutex));
818
819  lspg_query_push( lspg_seq_run_prep_cb, "SELECT px.seq_run_prep( %lld, %.3f, %.3f, %.3f, %.3f, %.3f, %.3f, %.3f)",
820  skey, kappa, phi, cx, cy, ax, ay, az);
821 }

```

5.1.4.38 void `lspg_seq_run_prep_cb` (`lspg_query_queue_t` * *qqp*, `PGresult` * *pgr*)

Callback for the `seq_run_prep` query.

Parameters:

- ← *qqp* The query item that generated this callback
- ← *pgr* The result of the query

Definition at line 793 of file `lspg.c`.

```

796  {
797  pthread_mutex_lock( &(lspg_seq_run_prep.mutex));
798  lspg_seq_run_prep.new_value_ready = 1;
799  pthread_cond_signal( &(lspg_seq_run_prep.cond));
800  pthread_mutex_unlock( &(lspg_seq_run_prep.mutex));
801 }

```

5.1.4.39 void lspg_seq_run_prep_done ()

Indicate we are done waiting.

Definition at line 833 of file lspg.c.

```
833     {
834     pthread_mutex_unlock( &(lspg_seq_run_prep.mutex));
835 }
```

5.1.4.40 void lspg_seq_run_prep_init ()

Initialize the data collection object.

Definition at line 785 of file lspg.c.

```
785     {
786     lspg_seq_run_prep.new_value_ready = 0;
787     pthread_mutex_init( &(lspg_seq_run_prep.mutex), NULL);
788     pthread_cond_init( &(lspg_seq_run_prep.cond), NULL);
789 }
```

5.1.4.41 void lspg_seq_run_prep_wait ()

Wait for seq run prep query to return.

Definition at line 825 of file lspg.c.

```
825     {
826     pthread_mutex_lock( &(lspg_seq_run_prep.mutex));
827     while( lspg_seq_run_prep.new_value_ready == 0)
828         pthread_cond_wait( &(lspg_seq_run_prep.cond), &(lspg_seq_run_prep.mutex));
829 }
```

5.1.4.42 void lspg_sig_service (struct pollfd * *evt*)

Service a signal Signals here are treated as file descriptors and fits into our poll scheme.

Parameters:

← *evt* The pollfd object that triggered this call

Definition at line 1087 of file lspg.c.

```
1089     {
1090     struct signalfd_siginfo fdsi;
1091
1092     //
1093     // Really, we don't care about the signal,
1094     // it's just used to drop out of the poll
1095     // function when there is something for us
1096     // to do that didn't involve something coming
1097     // from our postgresql server.
1098     //
1099     // This is accomplished by the query_push function
```



```
1100 // to notify us that a new query is ready.
1101 //
1102
1103 read( evt->fd, &fdsi, sizeof( struct signalfd_siginfo));
1104
1105 }
```

5.1.4.43 void lspg_wait_for_detector_all ()

Combined call to wait for the detector.

Definition at line 649 of file lspg.c.

```
649 {
650     lspg_wait_for_detector_call();
651     lspg_wait_for_detector_wait();
652     lspg_wait_for_detector_done();
653 }
```

5.1.4.44 void lspg_wait_for_detector_call ()

initiate the wait for detector query

Definition at line 623 of file lspg.c.

```
623 {
624     pthread_mutex_lock( &(lspg_wait_for_detector.mutex));
625     lspg_wait_for_detector.new_value_ready = 0;
626     pthread_mutex_unlock( &(lspg_wait_for_detector.mutex));
627
628     lspg_query_push( lspg_wait_for_detector_cb, "SELECT px.lock_detector_test_block
629     ()");
629 }
```

5.1.4.45 void lspg_wait_for_detector_cb (lspg_query_queue_t * qqp, PGresult * pgr)

Callback for the wait for detector query.

Definition at line 614 of file lspg.c.

```
614 {
615     pthread_mutex_lock( &(lspg_wait_for_detector.mutex));
616     lspg_wait_for_detector.new_value_ready = 1;
617     pthread_cond_signal( &(lspg_wait_for_detector.cond));
618     pthread_mutex_unlock( &(lspg_wait_for_detector.mutex));
619 }
```

5.1.4.46 void lspg_wait_for_detector_done ()

Done waiting for the detector.

Definition at line 642 of file lspg.c.

```
642 {
643     pthread_mutex_unlock( &(lspg_wait_for_detector.mutex));
644 }
```

5.1.4.47 void lspg_wait_for_detector_init ()

initialize the detector timing object

Definition at line 606 of file lspg.c.

```

606         {
607     lspg_wait_for_detector.new_value_ready = 0;
608     pthread_mutex_init( &(lspg_wait_for_detector.mutex), NULL);
609     pthread_cond_init( &(lspg_wait_for_detector.cond), NULL);
610 }
```

5.1.4.48 void lspg_wait_for_detector_wait ()

Pause the calling thread until the detector is ready Called by the MD2 thread.

Definition at line 634 of file lspg.c.

```

634         {
635     pthread_mutex_lock( &(lspg_wait_for_detector.mutex));
636     while( lspg_wait_for_detector.new_value_ready == 0)
637         pthread_cond_wait( &(lspg_wait_for_detector.cond), &(lspg_wait_for_detector.
        mutex));
638 }
```

5.1.4.49 void* lspg_worker (void * *dummy*)

The main loop for the lspg thread.

Parameters:

← *dummy* Required by pthreads but unused

Definition at line 1366 of file lspg.c.

```

1368     {
1369     static struct pollfd fda[2]; // 0=signal handler, 1=pg socket
1370     static int nfda = 0;
1371     static sigset_t our_sigset;
1372     int sigfd;
1373
1374     sigemptyset( &our_sigset);
1375     sigaddset( &our_sigset, SIGUSR1);
1376
1377
1378     //
1379     // block ordinary signal mechanism
1380     //
1381     sigprocmask(SIG_BLOCK, &our_sigset, NULL);
1382
1383
1384     fda[0].fd = signalfd( -1, &our_sigset, SFD_NONBLOCK);
1385     if( fda[0].fd == -1) {
1386         char *es;
1387
1388         es = strerror( errno);
1389         pthread_mutex_lock( &ncurses_mutex);
1390         wprintw( term_output, "Signalfd trouble: %s", es);
1391         wnoutrefresh( term_output);
```

```

1392     wnoutrefresh( term_input);
1393     doupdate();
1394     pthread_mutex_unlock( &ncurses_mutex);
1395 }
1396 fda[0].events = POLLIN;
1397
1398 //
1399 // make sure file descriptor is not legal until it's been conneceted
1400 //
1401 lspgfd.fd = -1;
1402
1403
1404 while( 1) {
1405     int pollrtn;
1406     int poll_timeout_ms;
1407
1408     lspg_next_state();
1409
1410     if( lspgfd.fd == -1) {
1411         //
1412         // Here a connection to the database is not established.
1413         // Periodicaly try again. Should possibly arrange to reconnect
1414         // to signalfd but that's unlikely to be nessesary.
1415         //
1416         nfda = 1;
1417         poll_timeout_ms = 10000;
1418         fda[1].revents = 0;
1419     } else {
1420         //
1421         // Arrange to peacefully do nothing until either the pg server sends us some
thing
1422         // or someone pushes something onto our queue
1423         //
1424         nfda = 2;
1425         fda[1].fd = lspgfd.fd;
1426         fda[1].events = lspgfd.events;
1427         fda[1].revents = 0;
1428         poll_timeout_ms = -1;
1429     }
1430
1431     pollrtn = poll( fda, nfda, poll_timeout_ms);
1432
1433     if( pollrtn && fda[0].revents) {
1434         lspg_sig_service( &(fda[0]));
1435         pollrtn--;
1436     }
1437     if( pollrtn && fda[1].revents) {
1438         lspg_pg_service( &(fda[1]));
1439         pollrtn--;
1440     }
1441
1442
1443
1444
1445 }
1446 }

```

5.1.4.50 void lspg_zoom_lut_cb (lspg_query_queue_t * *qqp*, PGresult * *pgr*)

Zoom motor look up table callback.

Parameters:

← *qqp* the queue item responsible for calling us

← **pgr** The Postgresql result object

Definition at line 217 of file lspg.c.

```

220             {
221     int i;
222
223     pthread_mutex_lock( &(amp;zoom->mutex));
224
225     zoom->nlut = PQntuples( pgr)/2;
226     zoom->lut = calloc( 2*zoom->nlut, sizeof(double));
227     if( zoom->lut == NULL) {
228         wprintw( term_output, "\nOut of memory (lspg_zoom_lut_cb)");
229         wnoutrefresh( term_output);
230         wnoutrefresh( term_output);
231         douupdate();
232         pthread_mutex_unlock( &(amp;zoom->mutex));
233         return;
234     }
235
236     for( i=0; i<PQntuples( pgr); i++) {
237         zoom->lut[i] = strtod( PQgetvalue( pgr, i, 0), NULL);
238     }
239
240     pthread_mutex_unlock( &(amp;zoom->mutex));
241
242 }
```

5.1.5 Variable Documentation

5.1.5.1 `int ls_pg_state = LS_PG_STATE_INIT` [**static**]

State of the lspg state machine.

Definition at line 39 of file lspg.c.

5.1.5.2 `PostgresPollingStatusType lspg_connectPoll_response` [**static**]

Used to determine state while connecting.

Definition at line 70 of file lspg.c.

5.1.5.3 `lspg_lock_detector_t lspg_lock_detector` [**static**]

Definition at line 723 of file lspg.c.

5.1.5.4 `lspg_lock_diffractionmeter_t lspg_lock_diffractionmeter` [**static**]

Definition at line 664 of file lspg.c.

5.1.5.5 `lspg_nextshot_t lspg_nextshot`

the nextshot object

Definition at line 73 of file lspg.c.

5.1.5.6 `lspg_query_queue_t lspg_query_queue[LS_PG_QUERY_QUEUE_LENGTH]` `[static]`

Our query queue.

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

5.1.5.7 `unsigned int lspg_query_queue_off = 0` `[static]`

The last item still being used (on == off means nothing in queue).

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

5.1.5.8 `unsigned int lspg_query_queue_on = 0` `[static]`

Next position to add something to the queue.

Definition at line 63 of file `lspg.c`.

5.1.5.9 `unsigned int lspg_query_queue_reply = 0` `[static]`

The current item being digested. Normally `off <= reply <= on`. Corner case of queue wrap around works because we only increment and compare for equality.

Definition at line 65 of file `lspg.c`.

5.1.5.10 `PostgresPollingStatusType lspg_resetPoll_response` `[static]`

Used to determine state while reconnecting.

Definition at line 71 of file `lspg.c`.

5.1.5.11 `lspg_seq_run_prep_t lspg_seq_run_prep` `[static]`

Definition at line 781 of file `lspg.c`.

5.1.5.12 `pthread_t lspg_thread` `[static]`

our worker thread

Definition at line 41 of file `lspg.c`.

5.1.5.13 `lspg_wait_for_detector_t lspg_wait_for_detector` `[static]`

Instance of the detector timing object.

Definition at line 602 of file `lspg.c`.

5.1.5.14 `struct pollfd lspgfd` `[static]`

our poll info

Definition at line 43 of file lspg.c.

5.1.5.15 pthread_mutex_t pg_queue_mutex [static]

keep the queue from getting tangled

Definition at line 42 of file lspg.c.

5.1.5.16 PGconn* q = NULL [static]

Database connector.

Definition at line 69 of file lspg.c.

5.2 lspmacc File Reference

Routines concerned with communication with PMAC. `#include "pgpmac.h"`

Data Structures

- struct [md2StatusStruct](#)

The block of memory retrieved in a status request.

Defines

- `#define LS_PMAC_STATE_RESET -1`
- `#define LS_PMAC_STATE_DETACHED 0`
- `#define LS_PMAC_STATE_IDLE 1`
- `#define LS_PMAC_STATE_SC 2`
- `#define LS_PMAC_STATE_WACK_NFR 3`
- `#define LS_PMAC_STATE_WACK_CC 4`
- `#define LS_PMAC_STATE_WACK 5`
- `#define LS_PMAC_STATE_GMR 6`
- `#define LS_PMAC_STATE_CR 7`
- `#define LS_PMAC_STATE_RR 8`
- `#define LS_PMAC_STATE_WACK_RR 9`
- `#define LS_PMAC_STATE_GB 10`
- `#define LS_PMAC_STATE_WCR 11`
- `#define LS_PMAC_STATE_WGB 12`
- `#define PMACPORT 1025`
The PMAC (only) listens on this port.
- `#define pmac_cmd_size 8`
PMAC command size in bytes.
- `#define VR_UPLOAD 0xc0`
- `#define VR_DOWNLOAD 0x40`
- `#define VR_PMAC_SENDLINE 0xb0`
- `#define VR_PMAC_GETLINE 0xb1`
- `#define VR_PMAC_FLUSH 0xb3`
- `#define VR_PMAC_GETMEM 0xb4`
- `#define VR_PMAC_SETMEM 0xb5`
- `#define VR_PMAC_SENDCTRLCHAR 0xb6`
- `#define VR_PMAC_SETBIT 0xba`
- `#define VR_PMAC_SETBITS 0xbb`
- `#define VR_PMAC_PORT 0xbe`
- `#define VR_PMAC_GETRESPONSE 0xbf`
- `#define VR_PMAC_READREADY 0xc2`
- `#define VR_CTRL_RESPONSE 0xc4`
- `#define VR_PMAC_GETBUFFER 0xc5`
- `#define VR_PMAC_WRITEBUFFER 0xc6`
- `#define VR_PMAC_WRITEERROR 0xc7`

- #define [VR_FWDOWNLOAD](#) 0xcb
- #define [VR_IPADDRESS](#) 0xe0
- #define [PMAC_MIN_CMD_TIME](#) 20000.0
Minimum time between commands to the pmac.
- #define [PMAC_CMD_QUEUE_LENGTH](#) 2048
Size of the PMAC command queue.

Typedefs

- typedef struct [md2StatusStruct](#) [md2_status_t](#)
The block of memory retrieved in a status request.

Functions

- void [hex_dump](#) (int n, unsigned char *s)
Prints a hex dump of the given data.
- void [cleanstr](#) (char *s)
*Replace with
in null terminated string and print result to terminal.*
- void [lsConnect](#) (char *ipaddr)
Connect to the PMAC socket.
- [pmac_cmd_queue_t](#) * [lspmac_push_queue](#) ([pmac_cmd_queue_t](#) *cmd)
Put a new command on the queue.
- [pmac_cmd_queue_t](#) * [lspmac_pop_queue](#) ()
Remove the oldest queue item.
- [pmac_cmd_queue_t](#) * [lspmac_pop_reply](#) ()
Remove the next command queue item that is waiting for a reply.
- [pmac_cmd_queue_t](#) * [lspmac_send_command](#) (int rqType, int rq, int wValue, int wIndex, int wLength, unsigned char *data, void(*responseCB)([pmac_cmd_queue_t](#) *, int, unsigned char *), int no_reply)
Compose a packet and send it to the PMAC.
- void [lspmac_SockFlush](#) ()
Reset the PMAC socket from the PMAC side.
- void [lspmac_Reset](#) ()
Clear the queue and put the PMAC into a known state.
- void [lspmac_Error](#) (unsigned char *buff)
The service routing detected an error condition.

- void [lspmac_Service](#) (struct pollfd *evt)
Service routine for packet coming from the PMAC.
- void [lspmac_GetShortReplyCB](#) (pmac_cmd_queue_t *cmd, int nreceived, unsigned char *buff)
Receive a reply that does not require multiple buffers.
- void [lspmac_SendControlReplyPrintCB](#) (pmac_cmd_queue_t *cmd, int nreceived, unsigned char *buff)
Receive a reply to a control character Print a "printable" version of the character to the terminal Followed by a hex dump of the response.
- void [lspmac_GetmemReplyCB](#) (pmac_cmd_queue_t *cmd, int nreceived, unsigned char *buff)
Service a reply to the getmem command.
- pmac_cmd_queue_t * [lspmac_SockGetmem](#) (int offset, int nbytes)
Request a chunk of memory to be returned.
- pmac_cmd_queue_t * [lspmac_SockSendline](#) (char *fmt,...)
Send a one line command.
- pmac_cmd_queue_t * [lspmac_SockSendline_nr](#) (char *fmt,...)
Send a command and ignore the response.
- pmac_cmd_queue_t * [lspmac_SockSendControlCharPrint](#) (char c)
Send a control character.
- void [lspmac_Getmem](#) ()
Request a block of double buffer memory.
- void [lspmac_bio_read](#) (lspmac_motor_t *mp)
Read the state of a binary i/o motor This is the read method for the binary i/o motor class.
- void [lspmac_dac_read](#) (lspmac_motor_t *mp)
Read a DAC motor position.
- void [lspmac_shutter_read](#) (lspmac_motor_t *mp) pthread_mutex_lock(&lspmac_shutter_mutex)
Fast shutter read routine The shutter is mildly complicated in that we need to take into account the fact that the shutter can open and close again between status updates.
- if (md2_status.fs_has_opened &&!lspmac_shutter_has_opened &&!md2_status.fs_is_open)
- if (lspmac_shutter_state!=md2_status.fs_is_open)
- if (md2_status.fs_is_open)
- void [lspmac_pmacmotor_read](#) (lspmac_motor_t *mp)
Read the position and status of a normal PMAC motor.
- void [lspmac_get_status_cb](#) (pmac_cmd_queue_t *cmd, int nreceived, unsigned char *buff)
Service routing for status upate This updates positions and status information.
- void [lspmac_get_status](#) ()

Request a status update from the PMAC.

- void `lspmac_GetAllIVarsCB` (`pmac_cmd_queue_t` *cmd, int nreceived, unsigned char *buff)
Receive the values of all the I variables Update our Postgresql database with the results.
- void `lspmac_GetAllIVars` ()
Request the values of all the I variables.
- void `lspmac_GetAllMVarsCB` (`pmac_cmd_queue_t` *cmd, int nreceived, unsigned char *buff)
Receive the values of all the M variables Update our database with the results.
- void `lspmac_GetAllMVars` ()
Request the values of all the M variables.
- void `lspmac_sendcmd_nocb` (char *fmt,...)
Send a command that does not need to deal with the reply.
- void `lspmac_next_state` ()
State machine logic.
- void * `lspmac_worker` (void *dummy)
Our lspmac worker thread.
- double `lspmac_lut` (int nlut, double *lut, double x)
Look up table support for motor positions (think x=zoom, y=light intensity) use a lookup table to find the "counts" to move the motor to the requested position The look up table is a simple one dimensional array with the x values as even indices and the y values as odd indices.
- void `lspmac_movedac_queue` (`lspmac_motor_t` *mp, double requested_position)
Move method for dac motor objects (ie, lights).
- void `lspmac_movezoom_queue` (`lspmac_motor_t` *mp, double requested_position)
Move method for the zoom motor.
- void `lspmac_moveabs_fshut_queue` (`lspmac_motor_t` *mp, double requested_position)
Move method for the fast shutter.
- void `lspmac_moveabs_bio_queue` (`lspmac_motor_t` *mp, double requested_position)
Move method for binary i/o motor objects.
- void `lspmac_moveabs_queue` (`lspmac_motor_t` *mp, double requested_position)
Move method for normal stepper and servo motor objects.
- void `lspmac_moveabs_wait` (`lspmac_motor_t` *mp)
Wait for motor to finish moving.
- `lspmac_motor_t` * `lspmac_motor_init` (`lspmac_motor_t` *d, int motor_number, int wy, int wx, int *posp, int *stat1p, int *stat2p, char *wtitle, char *name, void(*moveAbs)(`lspmac_motor_t` *, double))
Initialize a pmac stepper or servo motor.

- `lspmac_motor_t * lspmac_fshut_init (lspmac_motor_t *d)`
Initialize the fast shutter motor.
- `lspmac_motor_t * lspmac_bio_init (lspmac_motor_t *d, char *name, char *write_fmt, int *read_ptr, int read_mask)`
Initialize binary i/o motor.
- `lspmac_motor_t * lspmac_dac_init (lspmac_motor_t *d, int *posp, double scale, char *mvar, char *name)`
Initialize DAC motor Note that some motors require further initialization from a database query.
- `void lspmac_init (int ivarsflag, int mvarsflag)`
Initialize this module.
- `void lspmac_run ()`
Start up the lspmac thread.

Variables

- `static int ls_pmac_state = LS_PMAC_STATE_DETACHED`
Current state of the PMAC communications state machine.
- `int lspmac_shutter_state`
State of the shutter; used to detect changes.
- `int lspmac_shutter_has_opened = md2_status.fs_has_opened`
Indicates that the shutter had opened, perhaps briefly even if the state did not change.
- `pthread_mutex_t lspmac_shutter_mutex`
Coordinates threads reading shutter status.
- `pthread_cond_t lspmac_shutter_cond`
Allows waiting for the shutter status to change.
- `pthread_mutex_t lspmac_moving_mutex`
Coordinate moving motors between threads.
- `pthread_cond_t lspmac_moving_cond`
Wait for motor(s) to finish moving condition.
- `int lspmac_moving_flags`
Flag used to implement motor moving condition.
- `static pthread_t pmac_thread`
our thread to manage access and communication to the pmac
- `static pthread_mutex_t pmac_queue_mutex`

manage access to the pmac command queue

- static pthread_cond_t [pmac_queue_cond](#)
wait for a command to be sent to PMAC before continuing
- static struct pollfd [pmacfd](#)
our poll structure
- static int [getivars](#) = 0
flag set at initialization to send i vars to db
- static int [getmvars](#) = 0
flag set at initialization to send m vars to db
- [lspmac_motor_t](#) [lspmac_motors](#) [32]
All our motors.
- int [lspmac_nmotors](#) = 0
The number of motors we manage.
- [lspmac_motor_t](#) * [omega](#)
MD2 omega axis (the air bearing).
- [lspmac_motor_t](#) * [alignx](#)
Alignment stage X.
- [lspmac_motor_t](#) * [aligny](#)
Alignment stage Y.
- [lspmac_motor_t](#) * [alignz](#)
Alignment stage X.
- [lspmac_motor_t](#) * [anal](#)
Polaroid analyzer motor.
- [lspmac_motor_t](#) * [zoom](#)
Optical zoom.
- [lspmac_motor_t](#) * [apery](#)
Aperture Y.
- [lspmac_motor_t](#) * [aperz](#)
Aperture Z.
- [lspmac_motor_t](#) * [capy](#)
Capillary Y.
- [lspmac_motor_t](#) * [capz](#)
Capillary Z.

- `lspmac_motor_t * scinz`
Scintillator Z.
- `lspmac_motor_t * cenx`
Centering Table X.
- `lspmac_motor_t * ceny`
Centering Table Y.
- `lspmac_motor_t * kappa`
Kappa.
- `lspmac_motor_t * phi`
Phi (not data collection axis).
- `lspmac_motor_t * fshut`
Fast shutter.
- `lspmac_motor_t * flight`
Front Light DAC.
- `lspmac_motor_t * blight`
Back Light DAC.
- `lspmac_motor_t * fscint`
Scintillator Piezo DAC.
- `lspmac_motor_t * blight_ud`
Back Light Up/Down actuator.
- `static int linesReceived = 0`
current number of lines received
- `static unsigned char dbmem [64 * 1024]`
double buffered memory
- `static int dbmemIn = 0`
next location
- `static struct timeval pmac_time_sent now`
used to ensure we do not send commands to the pmac too often. Only needed for non-DB commands.
- `static pmac_cmd_t rr_cmd`
- `static pmac_cmd_t gb_cmd`
- `static pmac_cmd_t cr_cmd`
commands to send out "readready", "getbuffer", controlresponse (initialized in main)
- `static pmac_cmd_queue_t ethCmdQueue [PMAC_CMD_QUEUE_LENGTH]`
PMAC command queue.

- static unsigned int `ethCmdOn` = 0
points to next empty PMAC command queue position
- static unsigned int `ethCmdOff` = 0
points to current command (or none if == ethCmdOn)
- static unsigned int `ethCmdReply` = 0
Used like ethCmdOff only to deal with the pmac reply to a command.
- static char * `pmac_error_strs` []
Decode the errors perhaps returned by the PMAC.
- static `md2_status_t` `md2_status`
Buffer for MD2 Status.
- `pthread_mutex_t` `md2_status_mutex`
Synchronize reading/writing status buffer.

5.2.1 Detailed Description

Routines concerned with communication with PMAC.

Date:

2012

Author:

Keith Brister All Rights Reserved

This is a state machine (surprise!) Lacking is support for writingbuffer, control writing and reading, as well as double buffered memory It looks like several different methods of managing PMAC communications are possible. Here is set up a queue of outgoing commands and deal completely with the result before sending the next. A full handshake of acknowledgements and "readready" is expected.

State	Description
-1	Reset the connection
0	Detached: need to connect to tcp port
1	Idle (waiting for a command to send to the pmac)
2	Send command
3	Waiting for command acknowledgement (no further response expected)
4	Waiting for control character acknowledgement (further response expected)
5	Waiting for command acknowledgement (further response expected)
6	Waiting for get memory response
7	Send controlresponse
8	Send readready
9	Waiting for acknowledgement of "readready"
10	Send readbuffer
11	Waiting for control response
12	Waiting for readbuffer response

Definition in file [lspmac.c](#).

5.2.2 Define Documentation

5.2.2.1 **#define LS_PMAC_STATE_CR 7**

Definition at line 45 of file lspmac.c.

5.2.2.2 **#define LS_PMAC_STATE_DETACHED 0**

Definition at line 38 of file lspmac.c.

5.2.2.3 **#define LS_PMAC_STATE_GB 10**

Definition at line 48 of file lspmac.c.

5.2.2.4 **#define LS_PMAC_STATE_GMR 6**

Definition at line 44 of file lspmac.c.

5.2.2.5 **#define LS_PMAC_STATE_IDLE 1**

Definition at line 39 of file lspmac.c.

5.2.2.6 **#define LS_PMAC_STATE_RESET -1**

Definition at line 37 of file lspmac.c.

5.2.2.7 **#define LS_PMAC_STATE_RR 8**

Definition at line 46 of file lspmac.c.

5.2.2.8 **#define LS_PMAC_STATE_SC 2**

Definition at line 40 of file lspmac.c.

5.2.2.9 **#define LS_PMAC_STATE_WACK 5**

Definition at line 43 of file lspmac.c.

5.2.2.10 **#define LS_PMAC_STATE_WACK_CC 4**

Definition at line 42 of file lspmac.c.

5.2.2.11 #define LS_PMAC_STATE_WACK_NFR 3

Definition at line 41 of file lspmac.c.

5.2.2.12 #define LS_PMAC_STATE_WACK_RR 9

Definition at line 47 of file lspmac.c.

5.2.2.13 #define LS_PMAC_STATE_WCR 11

Definition at line 49 of file lspmac.c.

5.2.2.14 #define LS_PMAC_STATE_WGB 12

Definition at line 50 of file lspmac.c.

5.2.2.15 #define PMAC_CMD_QUEUE_LENGTH 2048

Size of the PMAC command queue.

Definition at line 137 of file lspmac.c.

5.2.2.16 #define pmac_cmd_size 8

PMAC command size in bytes.

Definition at line 103 of file lspmac.c.

5.2.2.17 #define PMAC_MIN_CMD_TIME 20000.0

Minimum time between commands to the pmac.

Definition at line 133 of file lspmac.c.

5.2.2.18 #define PMACPORT 1025

The PMAC (only) listens on this port.

Definition at line 97 of file lspmac.c.

5.2.2.19 #define VR_CTRL_RESPONSE 0xc4

Definition at line 119 of file lspmac.c.

5.2.2.20 #define VR_DOWNLOAD 0x40

Definition at line 106 of file lspmac.c.

5.2.2.21 #define VR_FWDOWNLOAD 0xcb

Definition at line 123 of file lspmac.c.

5.2.2.22 #define VR_IPADDRESS 0xe0

Definition at line 124 of file lspmac.c.

5.2.2.23 #define VR_PMAC_FLUSH 0xb3

Definition at line 110 of file lspmac.c.

5.2.2.24 #define VR_PMAC_GETBUFFER 0xc5

Definition at line 120 of file lspmac.c.

5.2.2.25 #define VR_PMAC_GETLINE 0xb1

Definition at line 109 of file lspmac.c.

5.2.2.26 #define VR_PMAC_GETMEM 0xb4

Definition at line 111 of file lspmac.c.

5.2.2.27 #define VR_PMAC_GETRESPONSE 0xbf

Definition at line 117 of file lspmac.c.

5.2.2.28 #define VR_PMAC_PORT 0xbe

Definition at line 116 of file lspmac.c.

5.2.2.29 #define VR_PMAC_READREADY 0xc2

Definition at line 118 of file lspmac.c.

5.2.2.30 #define VR_PMAC_SENDCTRLCHAR 0xb6

Definition at line 113 of file lspmac.c.

5.2.2.31 #define VR_PMAC_SENDLINE 0xb0

Definition at line 108 of file lspmac.c.

5.2.2.32 `#define VR_PMAC_SETBIT 0xba`

Definition at line 114 of file `lspmac.c`.

5.2.2.33 `#define VR_PMAC_SETBITS 0xbb`

Definition at line 115 of file `lspmac.c`.

5.2.2.34 `#define VR_PMAC_SETMEM 0xb5`

Definition at line 112 of file `lspmac.c`.

5.2.2.35 `#define VR_PMAC_WRITEBUFFER 0xc6`

Definition at line 121 of file `lspmac.c`.

5.2.2.36 `#define VR_PMAC_WRITEERROR 0xc7`

Definition at line 122 of file `lspmac.c`.

5.2.2.37 `#define VR_UPLOAD 0xc0`

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

5.2.3 Typedef Documentation

5.2.3.1 `typedef struct md2StatusStruct md2_status_t`

The block of memory retrieved in a status request.

5.2.4 Function Documentation

5.2.4.1 `void cleanstr (char * s)`

Replace with

in null terminated string and print result to terminal. Needed to turn PMAC messages into something printable.

Parameters:

← *s* String to print to terminal.

Definition at line 312 of file `lspmac.c`.

```
314         {
315     int i;
316
317     pthread_mutex_lock( &ncurses_mutex);
318
```

```

319     for( i=0; i<strlen( s); i++) {
320         if( s[i] == '\r')
321             wprintw( term_output, "\n");
322         else
323             wprintw( term_output, "%c", s[i]);
324     }
325
326     pthread_mutex_unlock( &ncurses_mutex);
327 }

```

5.2.4.2 void hex_dump(int *n*, unsigned char * *s*)

Prints a hex dump of the given data. Used to debug packet data.

Parameters:

- ← *n* Number of bytes passed in *s*
- ← *s* Data to dump

Definition at line 284 of file lspmac.c.

```

287     {
288
289     int i;          // row counter
290     int j;          // column counter
291
292     pthread_mutex_lock( &ncurses_mutex);
293
294     for( i=0; n > 0; i++) {
295         for( j=0; j<16 && n > 0; j++) {
296             if( j==8)
297                 wprintw( term_output, " ");
298             wprintw( term_output, " %02x", *(s + 16*i + j));
299             n--;
300         }
301         wprintw( term_output, "\n");
302     }
303     wprintw( term_output, "\n");
304
305     pthread_mutex_unlock( &ncurses_mutex);
306 }

```

5.2.4.3 if (md2_status.fs_is_open)

Definition at line 968 of file lspmac.c.

```

968     {
969         mvwprintw( term_status2, 1, 1, "Shutter Open ");
970         mp->position = 1;
971     } else {

```

5.2.4.4 if (lspmac_shutter_state! = md2_status.fs_is_open)

Definition at line 963 of file lspmac.c.

```

963                                     {
964     lspmac_shutter_state = md2_status.fs_is_open;
965     pthread_cond_signal( &lspmac_shutter_cond);
966 }

```

5.2.4.5 if (md2_status.fs_has_opened &&!lspmac_shutter_has_opened &&!md2_status.fs_is_open)

Definition at line 954 of file lspmac.c.

```

954     {
955     //
956     // Here the shutter opened and closed again before we got the memo
957     // Treat it as a shutter closed event
958     //
959     pthread_cond_signal( &lspmac_shutter_cond);
960 }

```

5.2.4.6 void lsConnect (char *ipaddr)

Connect to the PMAC socket. Establish or reestablish communications.

Parameters:

← *ipaddr* String representation of the IP address (dot quad or FQN)

Definition at line 333 of file lspmac.c.

```

335     {
336     int psock;                // our socket: value stored in pmacfda.fd
337     int err;                  // error code from some system calls
338     struct sockaddr_in *addrP; // our address structure to connect to
339     struct addrinfo ai_hints;  // required for getaddrinfo
340     struct addrinfo *ai_resultP; // linked list of address structures (we'll always
    s pick the first)
341
342     pmacfd.fd = -1;
343     pmacfd.events = 0;
344
345     // Initial buffer(s)
346     memset( &ai_hints, 0, sizeof( ai_hints));
347
348     ai_hints.ai_family = AF_INET;
349     ai_hints.ai_socktype = SOCK_STREAM;
350
351
352     //
353     // get address
354     //
355     err = getaddrinfo( ipaddr, NULL, &ai_hints, &ai_resultP);
356     if( err != 0) {
357
358         pthread_mutex_lock( &ncurses_mutex);
359
360         wprintw( term_output, "Could not find address: %s\n", gai_strerror( err));
361
362         wnoutrefresh( term_output);
363         wnoutrefresh( term_input);

```

```

364     doupdate();
365
366     pthread_mutex_unlock( &ncurses_mutex);
367
368     return;
369 }
370
371
372 addrP = (struct sockaddr_in *)ai_resultP->ai_addr;
373 addrP->sin_port = htons( PMACPORT);
374
375
376 psock = socket( PF_INET, SOCK_STREAM, 0);
377 if( psock == -1) {
378
379     pthread_mutex_lock( &ncurses_mutex);
380     wprintw( term_output, "Could not create socket\n");
381
382     wnoutrefresh( term_output);
383     wnoutrefresh( term_input);
384     doupdate();
385     pthread_mutex_unlock( &ncurses_mutex);
386     return;
387 }
388
389 err = connect( psock, (const struct sockaddr *)addrP, sizeof( *addrP));
390 if( err != 0) {
391     pthread_mutex_lock( &ncurses_mutex);
392     wprintw( term_output, "Could not connect socket: %s\n", strerror( errno));
393
394     wnoutrefresh( term_output);
395     wnoutrefresh( term_input);
396     doupdate();
397     pthread_mutex_unlock( &ncurses_mutex);
398     return;
399 }
400
401 ls_pmac_state = LS_PMAC_STATE_IDLE;
402 pmacfd.fd      = psock;
403 pmacfd.events  = POLLIN;
404
405 }

```

5.2.4.7 `lspmac_motor_t* lspmac_bio_init(lspmac_motor_t *d, char *name, char *write_fmt, int *read_ptr, int read_mask)`

Initialize binary i/o motor.

Parameters:

- ← **d** Our uninitialized motor object
- ← **name** Name of motor to coordinate with DB
- ← **write_fmt** Format string used to generate PMAC command to move motor
- ← **read_ptr** Pointer to byte in md2_status to find position
- ← **read_mask** Bitmask to find position in *read_ptr

Definition at line 1722 of file lspmac.c.

```

1728                                     {
1729     lspmac_nmotors++;

```

```

1730
1731     d->name           = strdup( name);
1732     d->moveAbs        = lspmac_moveabs_bio_queue;
1733     d->read           = lspmac_bio_read;
1734     d->lut            = NULL;
1735     d->nlut           = 0;
1736     d->actual_pos_cnts_p = NULL;
1737     d->status1        = NULL;
1738     d->status2        = NULL;
1739     d->motor_num      = -1;
1740     d->dac_mvar        = NULL;
1741     d->win            = NULL;
1742     d->write_fmt       = strdup( write_fmt);
1743     d->read_ptr        = read_ptr;
1744     d->read_mask       = read_mask;
1745     d->win            = NULL;
1746     d->u2c            = 1.0;
1747 }

```

5.2.4.8 void lspmac_bio_read (lspmac_motor_t * mp)

Read the state of a binary i/o motor This is the read method for the binary i/o motor class.

Parameters:

← *mp* The motor

Definition at line 911 of file lspmac.c.

```

913     {
914     char s[512];
915     int pos;
916
917     pthread_mutex_lock( &(mp->mutex));
918
919     pos = (*mp->read_ptr) & mp->read_mask == 0 ? 0 : 1;
920     mp->position = pos;
921
922     if( mp->u2c != 0.0) {
923         mp->position = *mp->actual_pos_cnts_p/mp->u2c;
924         snprintf( s, sizeof(s)-1, mp->format, 8, pos/mp->u2c);
925     } else {
926         mp->position = 1.0* (*mp->actual_pos_cnts_p);
927         snprintf( s, sizeof(s)-1, mp->format, 8, 1.0* (pos));
928     }
929
930     pthread_mutex_unlock( &(mp->mutex));
931 }

```

5.2.4.9 lspmac_motor_t* lspmac_dac_init (lspmac_motor_t * d, int * posp, double scale, char * mvar, char * name)

Initialize DAC motor Note that some motors require further initialization from a database query. For this reason this initialization code must be run before the database queue is allowed to be processed.

Parameters:

→ *d* Returns the (almost) initialized motor object [in,out] unitintialized motor

← *posp* Location of current position

- ← *scale* Scale factor (units)
- ← *mvar* M variable, ie, "M1200"
- ← *name* name to coordinate with DB

Definition at line 1756 of file lspmac.c.

```

1763                                     {
1764     lspmac_nmotors++;
1765     d->name      = strdup( name);
1766     d->moveAbs   = lspmac_movedac_queue;
1767     d->read      = lspmac_dac_read;
1768     d->lut       = NULL;
1769     d->nlut      = 0;
1770     d->actual_pos_cnts_p = posp;
1771     d->status1   = NULL;
1772     d->status2   = NULL;
1773     d->motor_num = -1;
1774     d->dac_mvar  = strdup(mvar);
1775     d->u2c       = scale;
1776     d->win       = NULL;
1777 }
```

5.2.4.10 void lspmac_dac_read (lspmac_motor_t * *mp*)

Read a DAC motor position.

Parameters:

- ← *mp* The motor

Definition at line 935 of file lspmac.c.

```

937                                     {
938     // TODO: impliement
939 }
```

5.2.4.11 void lspmac_Error (unsigned char * *buff*)

The service routing detected an error condition. Scan the response buffer for an error code and print it out.

Parameters:

- ← *buff* Buffer returned by PMAC perhaps containing a NULL terminated message.

Definition at line 557 of file lspmac.c.

```

559                                     {
560     int err;
561     //
562     // assume buff points to a 1400 byte array of stuff read from the pmac
563     //
564
565     if( buff[0] == 7 && buff[1] == 'E' && buff[2] == 'R' && buff[3] == 'R') {
566         buff[7] = 0; // For null termination
567         err = atoi( &(buff[4]));
568         if( err > 0 && err < 20) {
```

```

569     pthread_mutex_lock( &ncurses_mutex);
570     wprintw( term_output, "\n%s\n", pmac_error_strs[err]);
571     wnoutrefresh( term_output);
572     wnoutrefresh( term_input);
573     doupdate();
574     pthread_mutex_unlock( &ncurses_mutex);
575 }
576 }
577 lspmac_Reset();
578 }

```

5.2.4.12 `lspmac_motor_t* lspmac_fshut_init (lspmac_motor_t * d)`

Initialize the fast shutter motor.

Parameters:

← *d* Our uninitialized motor object

Definition at line 1698 of file `lspmac.c`.

```

1700                                     {
1701     lspmac_nmotors++;
1702     d->name = strdup("fastShutter");
1703     d->moveAbs = lspmac_moveabs_fshut_queue;
1704     d->read = lspmac_shutter_read;
1705     d->lut = NULL;
1706     d->nlut = 0;
1707     d->actual_pos_cnts_p = NULL;
1708     d->status1 = NULL;
1709     d->status2 = NULL;
1710     d->motor_num = -1;
1711     d->dac_mvar = NULL;
1712     d->win = NULL;
1713 }

```

5.2.4.13 `void lspmac_get_status ()`

Request a status update from the PMAC.

Definition at line 1179 of file `lspmac.c`.

```

1179                                     {
1180     lspmac_send_command( VR_UPLOAD, VR_PMAC_GETMEM, 0x400, 0, sizeof(md2_status_t),
1181     NULL, lspmac_get_status_cb, 0);
1181 }

```

5.2.4.14 `void lspmac_get_status_cb (pmac_cmd_queue_t * cmd, int nreceived, unsigned char * buff)`

Service routing for status update This updates positions and status information.

Parameters:

← *cmd* The command that generated this reply

← *nreceived* Number of bytes received

← *buff* The Big Byte Buffer

Definition at line 1057 of file lspmac.c.

```
1061     {
1062     static int cnt = 0;
1063     static char s[256];
1064
1065     char *sp;
1066     int i, pos;
1067     lspmac_motor_t *mp;
1068
1069     pthread_mutex_lock( &md2_status_mutex);
1070     memcpy( &md2_status, buff, sizeof(md2_status));
1071     pthread_mutex_unlock( &md2_status_mutex);
1072
1073
1074     //
1075     // track the coordinate system moving flags
1076     //
1077     pthread_mutex_lock( &lspmac_moving_mutex);
1078     if( md2_status.moving_flags != lspmac_moving_flags) {
1079         lspmac_moving_flags = md2_status.moving_flags;
1080         pthread_cond_signal( &lspmac_moving_cond);
1081     }
1082     pthread_mutex_unlock( &lspmac_moving_mutex);
1083
1084
1085     pthread_mutex_lock( &ncurses_mutex);
1086
1087     for( i=0; i<lspmac_nmotors; i++) {
1088         lspmac_motors[i].read(&(lspmac_motors[i]));
1089     }
1090
1091     // acc11c_1
1092     // mask bit
1093     // 0x01 0    Air pressure OK
1094     // 0x02 1    Air bearing OK
1095     // 0x04 2    Cryo switch
1096     // 0x08 3
1097     // 0x10 4
1098     // 0x20 5
1099     // 0x40 6    Cryo is back
1100
1101     if( md2_status.acc11c_1 & 0x40)
1102         mvwprintw( term_status2, 3, 1, "%s", -8, "Cryo Out");
1103     else
1104         mvwprintw( term_status2, 3, 1, "%s", -8, "Cryo In ");
1105
1106     //
1107     // acc11c_2
1108     // mask bit
1109     // 0x01 0    Fluor Dector back
1110     // 0x02 1    Sample Detected
1111     // 0x04 2
1112     // 0x08 3
1113     // 0x10 4
1114     // 0x20 5    Etel Ready
1115     // 0x40 6    Etel On
1116     // 0x80 7    Etel Init OK
1117
1118     if( md2_status.acc11c_2 & 0x01)
1119         mvwprintw( term_status2, 3, 10, "%s", -8, "Fluor Out");
1120     else
1121         mvwprintw( term_status2, 3, 10, "%s", -8, "Fluor In");
1122
1123     if( md2_status.acc11c_5 & 0x08)
```

```

1124     mvwprintw( term_status2, 4, 1, "%s", -(LS_DISPLAY_WINDOW_WIDTH-2), "Dryer On
1125 ");
1126     else
1127     mvwprintw( term_status2, 4, 1, "%s", -(LS_DISPLAY_WINDOW_WIDTH-2), "Dryer Of
1128 f");
1129     if( md2_status.acc11c_2 & 0x02)
1130     mvwprintw( term_status2, 2, 1, "%s", -(LS_DISPLAY_WINDOW_WIDTH-2), "Cap Dect
1131 ected");
1132     else
1133     mvwprintw( term_status2, 2, 1, "%s", -(LS_DISPLAY_WINDOW_WIDTH-2), "Cap Not
1134 Dected");
1135     wnoutrefresh( term_status2);
1136     // acc11c_3
1137     // mask bit
1138     // 0x01 0 Minikappa OK
1139     // 0x02 1
1140     // 0x04 2
1141     // 0x08 3 Arm Parked
1142     // acc11c_5
1143     // mask bit
1144     // 0x01 0 Mag Off
1145     // 0x02 1 Condenser Out
1146     // 0x04 2 Cryo Back
1147     // 0x08 3 Dryer On
1148     // 0x10 4 FluoDet Out
1149     // 0x20 5
1150     // 0x40 6 1=SmartMag, 0=Permanent Mag
1151     //
1152     // acc11c_6
1153     // mask bit
1154     // 0x0080 7 Etel Enable
1155     // 0x0100 8 Fast Shutter Enable
1156     // 0x0200 9 Fast Shutter Manual Enable
1157     // 0x0400 10 Fast Shutter On
1158     //
1159     if( md2_status.acc11c_5 & 0x02)
1160     mvwprintw( term_status, 3, 1, "%s", -(LS_DISPLAY_WINDOW_WIDTH-2), "Backligh
1161 t Up");
1162     else
1163     mvwprintw( term_status, 3, 1, "%s", -(LS_DISPLAY_WINDOW_WIDTH-2), "Backligh
1164 t Down");
1165     mvwprintw( term_status, 4, 1, "Front: %d", LS_DISPLAY_WINDOW_WIDTH-2-8,
1166 md2_status.front_dac);
1167     mvwprintw( term_status, 5, 1, "Back: %d", LS_DISPLAY_WINDOW_WIDTH-2-7,
1168 md2_status.back_dac);
1169     mvwprintw( term_status, 6, 1, "Piezo: %d", LS_DISPLAY_WINDOW_WIDTH-2-8,
1170 md2_status.scint_piezo);
1171     wnoutrefresh( term_status);
1172     wnoutrefresh( term_input);
1173     doupdate();
1174     pthread_mutex_unlock( &ncurses_mutex);
1175 }

```

5.2.4.15 void lspmac_GetAllIVars ()

Request the values of all the I variables.

Definition at line 1204 of file lspmac.c.

```

1204         {
1205     static char *cmds = "I0..8191";
1206     lspmac_send_command( VR_DOWNLOAD, VR_PMAC_SENDLINE, 0, 0, strlen( cmds), cmds,
        lspmac_GetAllIVarsCB, 0);
1207 }
```

5.2.4.16 void lspmac_GetAllIVarsCB (pmac_cmd_queue_t * *cmd*, int *nreceived*, unsigned char * *buff*)

Receive the values of all the I variables Update our Postgresql database with the results.

Parameters:

- ← *cmd* The command that gave this response
- ← *nreceived* Number of bytes received
- ← *buff* The byte buffer

Definition at line 1187 of file lspmac.c.

```

1191         {
1192     static char qs[LS_PG_QUERY_STRING_LENGTH];
1193     char *sp;
1194     int i;
1195     for( i=0, sp=strtok(buff, "\r"); sp != NULL; sp=strtok( NULL, "\r"), i++) {
1196         snprintf( qs, sizeof( qs)-1, "SELECT pmac.md2_ivar_set( %d, '%s')", i, sp);
1197         qs[sizeof( qs)-1]=0;
1198         lspg_query_push( NULL, qs);
1199     }
1200 }
```

5.2.4.17 void lspmac_GetAllMVars ()

Request the values of all the M variables.

Definition at line 1229 of file lspmac.c.

```

1229         {
1230     static char *cmds = "M0..8191->";
1231     lspmac_send_command( VR_DOWNLOAD, VR_PMAC_SENDLINE, 0, 0, strlen( cmds), cmds,
        lspmac_GetAllMVarsCB, 0);
1232 }
```

5.2.4.18 void lspmac_GetAllMVarsCB (pmac_cmd_queue_t * *cmd*, int *nreceived*, unsigned char * *buff*)

Receive the values of all the M variables Update our database with the results.

Parameters:

- ← *cmd* The command that started this
- ← *nreceived* Number of bytes received

← *buff* Our byte buffer

Definition at line 1212 of file lspmac.c.

```

1216             {
1217     static char qs[LS_PG_QUERY_STRING_LENGTH];
1218     char *sp;
1219     int i;
1220     for( i=0, sp=strtok(buff, "\r"); sp != NULL; sp=strtok( NULL, "\r"), i++) {
1221         snprintf( qs, sizeof( qs)-1, "SELECT pmac.md2_mvar_set( %d, '%s'", i, sp);
1222         qs[sizeof( qs)-1]=0;
1223         lspg_query_push( NULL, qs);
1224     }
1225 }
```

5.2.4.19 void lspmac_Getmem ()

Request a block of double buffer memory.

Definition at line 902 of file lspmac.c.

```

902             {
903     int nbytes;
904     nbytes = (dbmemIn + 1400 > sizeof( dbmem)) ? sizeof( dbmem) - dbmemIn : 1400;
905     lspmac_SockGetmem( dbmemIn, nbytes);
906 }
```

5.2.4.20 void lspmac_GetmemReplyCB (pmac_cmd_queue_t *cmd, int nreceived, unsigned char *buff)

Service a reply to the getmem command. Not currently used.

< [in] Buffer of bytes received

Parameters:

← *cmd* Queue item this is a reply to
 ← *nreceived* Number of bytes received

Definition at line 834 of file lspmac.c.

```

837             {
839     memcpy( &(dbmem[ntohs(cmd->pcmd.wValue)]), buff, nreceived);
840
841     dbmemIn += nreceived;
842     if( dbmemIn >= sizeof( dbmem)) {
843         dbmemIn = 0;
844     }
845 }
```

5.2.4.21 void lspmac_GetShortReplyCB (pmac_cmd_queue_t *cmd, int nreceived, unsigned char *buff)

Receive a reply that does not require multiple buffers.

Parameters:

- ← **cmd** Queue item this is a reply to
- ← **nreceived** Number of bytes received
- ← **buff** The buffer of bytes

Definition at line 779 of file lspmac.c.

```

783                                     {
784
785     char *sp;           // pointer to the command this is a reply to
786
787     if( nreceived < 1400)
788         buff[nreceived]=0;
789
790     sp = (char *) (cmd->pcmd.bData);
791
792     if( *buff == 0) {
793         pthread_mutex_lock( &ncurses_mutex);
794         wprintw( term_output, "%s\n", sp);
795         pthread_mutex_unlock( &ncurses_mutex);
796     } else {
797         pthread_mutex_lock( &ncurses_mutex);
798         wprintw( term_output, "%s: ", sp);
799         pthread_mutex_unlock( &ncurses_mutex);
800         cleanstr( buff);
801     }
802     wnoutrefresh( term_output);
803     wnoutrefresh( term_input);
804     doupdate();
805
806     memset( cmd->pcmd.bData, 0, sizeof( cmd->pcmd.bData));
807 }
```

5.2.4.22 void lspmac_init (int ivarsflag, int mvarsflag)

Initialize this module.

Parameters:

- ← **ivarsflag** Set global flag to harvest i variables
- ← **mvarsflag** Set global flag to harvest m variables

Definition at line 1782 of file lspmac.c.

```

1785                                     {
1786     md2_status_t *p;
1787
1788     // Set our global harvest flags
1789     getivars = ivarsflag;
1790     getmvars = mvarsflag;
1791
1792     // All important status mutex
1793     pthread_mutex_init( &md2_status_mutex, NULL);
1794
1795     //
1796     // Initialize the motor objects
1797     //
1798
1799     p = &md2_status;
```

```

1800
1801  omega = lspmac_motor_init( &(lspmac_motors[ 0]), 1, 0, 0, &p->omega_act_pos,
    &p->omega_status_1, &p->omega_status_2, "Omega #1 &1 X", "omega",
    lspmac_moveabs_queue);
1802  alignx = lspmac_motor_init( &(lspmac_motors[ 1]), 2, 0, 1, &p->alignx_act_pos,
    &p->alignx_status_1, &p->alignx_status_2, "Align X #2 &3 X", "align.x",
    lspmac_moveabs_queue);
1803  aligny = lspmac_motor_init( &(lspmac_motors[ 2]), 3, 0, 2, &p->aligny_act_pos,
    &p->aligny_status_1, &p->aligny_status_2, "Align Y #3 &3 Y", "align.y",
    lspmac_moveabs_queue);
1804  alignz = lspmac_motor_init( &(lspmac_motors[ 3]), 4, 0, 3, &p->alignz_act_pos,
    &p->alignz_status_1, &p->alignz_status_2, "Align Z #4 &3 Z", "align.z",
    lspmac_moveabs_queue);
1805  anal = lspmac_motor_init( &(lspmac_motors[ 4]), 5, 0, 4, &p->
    analyzer_act_pos, &p->analyzer_status_1, &p->analyzer_status_2, "Anal #5",
    "lightPolar", lspmac_moveabs_queue);
1806  zoom = lspmac_motor_init( &(lspmac_motors[ 5]), 6, 1, 0, &p->zoom_act_pos,
    &p->zoom_status_1, &p->zoom_status_2, "Zoom #6 &4 Z", "zoom",
    lspmac_movezoom_queue);
1807  apery = lspmac_motor_init( &(lspmac_motors[ 6]), 7, 1, 1, &p->
    aperture_act_pos, &p->aperture_status_1, &p->aperture_status_2, "Aper Y #7 &5
    Y", "appy", lspmac_moveabs_queue);
1808  aperz = lspmac_motor_init( &(lspmac_motors[ 7]), 8, 1, 2, &p->
    aperturez_act_pos, &p->aperturez_status_1, &p->aperturez_status_2, "Aper Z #8 &5
    Z", "appz", lspmac_moveabs_queue);
1809  capy = lspmac_motor_init( &(lspmac_motors[ 8]), 9, 1, 3, &p->capy_act_pos,
    &p->capy_status_1, &p->capy_status_2, "Cap Y #9 &5 U", "capy",
    lspmac_moveabs_queue);
1810  capz = lspmac_motor_init( &(lspmac_motors[ 9]), 10, 1, 4, &p->capz_act_pos,
    &p->capz_status_1, &p->capz_status_2, "Cap Z #10 &5 V", "capz",
    lspmac_moveabs_queue);
1811  scinz = lspmac_motor_init( &(lspmac_motors[10]), 11, 2, 0, &p->scint_act_pos,
    &p->scint_status_1, &p->scint_status_2, "Scin Z #11 &5 W", "scint",
    lspmac_moveabs_queue);
1812  cenx = lspmac_motor_init( &(lspmac_motors[11]), 17, 2, 1, &p->
    centerx_act_pos, &p->centerx_status_1, &p->centerx_status_2, "Cen X #17 &2
    X", "centering.x", lspmac_moveabs_queue);
1813  ceny = lspmac_motor_init( &(lspmac_motors[12]), 18, 2, 2, &p->
    centery_act_pos, &p->centery_status_1, &p->centery_status_2, "Cen Y #18 &2
    Y", "centering.y", lspmac_moveabs_queue);
1814  kappa = lspmac_motor_init( &(lspmac_motors[13]), 19, 2, 3, &p->kappa_act_pos,
    &p->kappa_status_1, &p->kappa_status_2, "Kappa #19 &7 X", "kappa",
    lspmac_moveabs_queue);
1815  phi = lspmac_motor_init( &(lspmac_motors[14]), 20, 2, 4, &p->phi_act_pos,
    &p->phi_status_1, &p->phi_status_2, "Phi #20 &7 Y", "phi",
    lspmac_moveabs_queue);
1816
1817  fshut = lspmac_fshut_init( &(lspmac_motors[15]));
1818  flight = lspmac_dac_init( &(lspmac_motors[16]), &p->front_dac, 160.0, "M1200"
    , "frontLight.intensity");
1819  blight = lspmac_dac_init( &(lspmac_motors[17]), &p->back_dac, 160.0, "M1201"
    , "backLight.intensity");
1820  fscint = lspmac_dac_init( &(lspmac_motors[18]), &p->scint_piezo, 320.0, "M1203"
    , "scint.focus");
1821
1822  blight_ud = lspmac_bio_init( &(lspmac_motors[19]), "backLight", "M1101=%d", &(
    md2_status.acc11c_5), 0x02);
1823
1824
1825
1826
1827  //
1828  // Initialize several commands that get called, perhaps, alot
1829  //
1830  rr_cmd.RequestType = VR_UPLOAD;
1831  rr_cmd.Request = VR_PMAC_READREADY;
1832  rr_cmd.wValue = 0;

```

```

1833 rr_cmd.wIndex      = 0;
1834 rr_cmd.wLength     = htons(2);
1835 memset( rr_cmd.bData, 0, sizeof(rr_cmd.bData));
1836
1837 gb_cmd.RequestType = VR_UPLOAD;
1838 gb_cmd.Request     = VR_PMAC_GETBUFFER;
1839 gb_cmd.wValue      = 0;
1840 gb_cmd.wIndex      = 0;
1841 gb_cmd.wLength     = htons(1400);
1842 memset( gb_cmd.bData, 0, sizeof(gb_cmd.bData));
1843
1844 cr_cmd.RequestType = VR_UPLOAD;
1845 cr_cmd.Request     = VR_CTRL_RESPONSE;
1846 cr_cmd.wValue      = 0;
1847 cr_cmd.wIndex      = 0;
1848 cr_cmd.wLength     = htons(1400);
1849 memset( cr_cmd.bData, 0, sizeof(cr_cmd.bData));
1850
1851 //
1852 // Initialize some mutexs and conditions
1853 //
1854
1855 pthread_mutex_init( &pmac_queue_mutex, NULL);
1856 pthread_cond_init( &pmac_queue_cond, NULL);
1857
1858 lspmac_shutter_state = 0; // assume the shutter is
now closed: not a big deal if we are wrong
1859 pthread_mutex_init( &lspmac_shutter_mutex, NULL);
1860 pthread_cond_init( &lspmac_shutter_cond, NULL);
1861 pmacfd.fd = -1;
1862
1863 pthread_mutex_init( &lspmac_moving_mutex, NULL);
1864 pthread_cond_init( &lspmac_moving_cond, NULL);
1865
1866 }

```

5.2.4.23 double lspmac_lut (int *nlut*, double * *lut*, double *x*)

Look up table support for motor positions (think x=zoom, y=light intensity) use a lookup table to find the "counts" to move the motor to the requested position The look up table is a simple one dimensional array with the x values as even indicies and the y values as odd indicies. Returns: y value

Parameters:

- ← *nlut* number of entries in lookup table
- ← *lut* The lookup table: even indicies are the x values, odd are the y's
- ← *x* The x value we are looking up.

Definition at line 1401 of file lspmac.c.

```

1403                                     : even indicies
are the x values, odd are the y's */
1404 double x                          /**< [in] The x value we are looking up.
/
1405     ) {
1406     int i, foundone;
1407     double m;
1408     double y1, y2, x1, x2, y;
1409
1410
1411     foundone = 0;

```

```

1412 if( lut != NULL && nlut > 1) {
1413
1414     for( i=0; i < 2*nlut; i += 2) {
1415
1416         x1 = lut[i];
1417         y1 = lut[i+1];
1418         if( i < 2*nlut - 2) {
1419             x2 = lut[i+2];
1420             y2 = lut[i+3];
1421         }
1422
1423         //
1424         // First one too big? Use the y value of the first element
1425         //
1426         if( i == 0 && x1 > x) {
1427             y = y1;
1428             foundone = 1;
1429             break;
1430         }
1431
1432         //
1433         // Look for equality
1434         //
1435         if( x1 == x) {
1436             y = y1;
1437             foundone = 1;
1438             break;
1439         }
1440
1441         //
1442         // Maybe interpolate
1443         //
1444         if( (i < 2*nlut-2) && x < x2) {
1445             m = (y2 - y1) / (x2 - x1);
1446             y = m*(x - x1) + y1;
1447             foundone = 1;
1448             break;
1449         }
1450     }
1451     if( foundone == 0) {
1452         // must be bigger than the last entry
1453         //
1454         //
1455         y = lut[2*(nlut-1) + 1];
1456     }
1457     return y;
1458 }
1459 }

```

5.2.4.24 `lspmac_motor_t* lspmac_motor_init (lspmac_motor_t * d, int motor_number, int wy, int wx, int *posp, int *stat1p, int *stat2p, char *wtitle, char *name, void(*) (lspmac_motor_t *, double) moveAbs)`

Initialize a pmac stepper or servo motor.

Parameters:

- ↔ *d* An uninitialize motor object
- ← *motor_number* The PMAC motor number
- ← *wy* Curses status window row index
- ← *wx* Curses status window column index
- ← *posp* Pointer to position status

- ← *stat1p* Pointer to 1st status word
- ← *stat2p* Pointer to 2nd status word
- ← *wtile* Title for this motor (to display)
- ← *name* Name of this motor (to match database)
- ← *moveAbs* Method to use to move this motor

Definition at line 1661 of file lspmac.c.

```

1672                                     {
1673     lspmac_nmotors++;
1674
1675     pthread_mutex_init( &(d->mutex), NULL);
1676     pthread_cond_init( &(d->cond), NULL);
1677
1678     d->name = strdup(name);
1679     d->moveAbs = moveAbs;
1680     d->read = lspmac_pmacmotor_read;
1681     d->lut = NULL;
1682     d->nlut = 0;
1683     d->actual_pos_cnts_p = posp;
1684     d->status1 = stat1p;
1685     d->status2 = stat2p;
1686     d->motor_num = motor_number;
1687     d->dac_mvar = NULL;
1688     d->win = newwin( LS_DISPLAY_WINDOW_HEIGHT, LS_DISPLAY_WINDOW_WIDTH, wy*
LS_DISPLAY_WINDOW_HEIGHT, wx*LS_DISPLAY_WINDOW_WIDTH);
1689     box( d->win, 0, 0);
1690     mvwprintw( d->win, 1, 1, "%s", wtile);
1691     wnoutrefresh( d->win);
1692
1693     return d;
1694 }
```

5.2.4.25 void lspmac_moveabs_bio_queue (lspmac_motor_t * mp, double requested_position)

Move method for binary i/o motor objects.

Parameters:

- ← *mp* A binary i/o motor object
- ← *requested_position* a 1 or a 0 request to move

Definition at line 1562 of file lspmac.c.

```

1565                                     {
1566     pthread_mutex_lock( &(mp->mutex));
1567     mp->requested_position = requested_position;
1568     mp->not_done = 1;
1569     mp->motion_seen = 0;
1570     mp->requested_pos_cnts = requested_position;
1571     mp->pq = lspmac_SockSendline_nr( mp->write_fmt, mp->requested_pos_cnts);
1572     pthread_mutex_unlock( &(mp->mutex));
1573 }
```

5.2.4.26 void lspmac_moveabs_fshut_queue (lspmac_motor_t * mp, double requested_position)

Move method for the fast shutter. Slightly more complicated than a binary io as some flags need to be set up.

Parameters:

- mp* The fast shutter motor instance
- requested_position* 1 (open) or 0 (close), really

Definition at line 1535 of file lspmac.c.

```

1538                                     {
1539     pthread_mutex_lock( &(mp->mutex));
1540
1541     mp->requested_position = requested_position;
1542     mp->not_done         = 1;
1543     mp->motion_seen      = 0;
1544     mp->requested_pos_cnts = requested_position;
1545     if( requested_position != 0) {
1546         //
1547         // ScanEnable=0, ManualEnable=1, ManualOn=1
1548         //
1549         mp->pq = lspmac_SockSendline_nr( "M1124=0 M1125=1 M1126=1");
1550     } else {
1551         //
1552         // ManualOn=0, ManualEnable=0, ScanEnable=1
1553         //
1554         mp->pq = lspmac_SockSendline_nr( "M1126=0 M1125=0 M1124=1");
1555     }
1556
1557     pthread_mutex_unlock( &(mp->mutex));
1558 }
```

5.2.4.27 void lspmac_moveabs_queue (lspmac_motor_t * mp, double requested_position)

Move method for normal stepper and servo motor objects.

Parameters:

- ← *mp* The motor to move
- ← *requested_position* Where to move it

Definition at line 1577 of file lspmac.c.

```

1580                                     {
1581     char s[512];
1582
1583     pthread_mutex_lock( &(mp->mutex));
1584     mp->requested_position = requested_position;
1585     if( mp->u2c != 0.0) {
1586         mp->not_done         = 1;
1587         mp->motion_seen      = 0;
1588         mp->requested_pos_cnts = mp->u2c * requested_position;
1589         snprintf( s, sizeof(s)-1, "%#d j=%d", mp->motor_num, mp->requested_pos_cnts);
1590
1591         mp->pq = lspmac_SockSendline_nr( s);
1592     }
1593     pthread_mutex_unlock( &(mp->mutex));
1594 }
```

5.2.4.28 void lspmac_moveabs_wait (lspmac_motor_t * mp)

Wait for motor to finish moving. Assume motion already queued, now just wait

Parameters:

← *mp* The motor object to wait for

Definition at line 1600 of file lspmac.c.

```

1602     {
1603     struct timespec wt;
1604     int return_code;
1605
1606     pthread_mutex_lock( &pmac_queue_mutex);
1607
1608     //
1609     // wait for the command to be sent
1610     //
1611     while( mp->pq->time_sent.tv_sec==0)
1612         pthread_cond_wait( &pmac_queue_cond, &pmac_queue_mutex);
1613
1614     //
1615     // set the timeout to be long enough after we sent the motion request to ensure
1616     // that
1617     // we will have read back the motor moving status but not so long that the time
1618     // out causes
1619     // problems;
1620     //
1621     wt.tv_sec = mp->pq->time_sent.tv_sec;
1622     wt.tv_nsec = mp->pq->time_sent.tv_nsec + 500000000;
1623
1624     pthread_mutex_unlock( &pmac_queue_mutex);
1625
1626     if( wt.tv_nsec >= 1000000000) {
1627         wt.tv_nsec -= 1000000000;
1628         wt.tv_sec += 1;
1629     }
1630
1631     //
1632     // wait for the motion to have started
1633     // This will time out if the motion ends before we can read the status back
1634     // hence the added complication of time stamp of the sent packet.
1635     //
1636     return_code=0;
1637
1638     pthread_mutex_lock( &(mp->mutex));
1639     while( mp->motion_seen == 0 && return_code == 0)
1640         return_code = pthread_cond_timedwait( &(mp->cond), &(mp->mutex), &wt);
1641
1642     if( return_code == 0) {
1643         //
1644         // wait for the motion that we know has started to finish
1645         //
1646         while( mp->not_done)
1647             pthread_cond_wait( &(mp->cond), &(mp->mutex));
1648     }
1649
1650     //
1651     // if return code was not 0 then we know we shouldn't wait for not_done flag.
1652     // In this case the motion ended before we read the status that should the moto
1653     // r moving.
1654     //
1655     pthread_mutex_unlock( &(mp->mutex));

```

```
1655
1656 }
```

5.2.4.29 void lspmac_movedac_queue (lspmac_motor_t * mp, double requested_position)

Move method for dac motor objects (ie, lights).

Parameters:

- ← *mp* Our motor
- ← *requested_position* Desired x postion (look up and send y position)

Definition at line 1464 of file lspmac.c.

```
1467                                     {
1468     char s[512];
1469     double y;
1470
1471     pthread_mutex_lock( &(mp->mutex));
1472
1473     mp->requested_position = requested_position;
1474
1475     if( mp->nlut > 0 && mp->lut != NULL) {
1476         y = lspmac_lut( mp->nlut, mp->lut, requested_position);
1477
1478         mp->requested_pos_cnts = (int)y * mp->u2c;
1479         mp->not_done = 1;
1480         mp->motion_seen = 0;
1481
1482
1483         //
1484         // By convension requested_pos_cnts scales from 0 to 100
1485         // for the lights u2c converts this to 0 to 16,000
1486         // for the scintillator focus this is 0 to 32,000
1487         //
1488         snprintf( s, sizeof(s)-1, "%s=%d", mp->dac_mvar, (int)mp->requested_pos_cnts)
1489     ;
1489     mp->pq = lspmac_SockSendline_nr( s);
1490
1491     }
1492
1493     pthread_mutex_unlock( &(mp->mutex));
1494 }
```

5.2.4.30 void lspmac_movezoom_queue (lspmac_motor_t * mp, double requested_position)

Move method for the zoom motor.

Parameters:

- ← *mp* the zoom motor
- ← *requested_position* our desired zoom

Definition at line 1499 of file lspmac.c.

```
1502                                     {
1503     char s[512];
```

```

1504 double y;
1505 pthread_mutex_lock( &(mp->mutex));
1506
1507 mp->requested_position = requested_position;
1508
1509 if( mp->nlut > 0 && mp->lut != NULL) {
1510     y = lspmac_lut( mp->nlut, mp->lut, requested_position);
1511
1512     mp->requested_pos_cnts = (int)y;
1513     mp->not_done = 1;
1514     mp->motion_seen = 0;
1515
1516
1517     snprintf( s, sizeof(s)-1, "%d j=%d", mp->motor_num, mp->requested_pos_cnts);
1518
1519     mp->pq = lspmac_SockSendline_nr( s);
1520 }
1521 pthread_mutex_unlock( &(mp->mutex));
1522
1523 //
1524 // the lights should "move" with the zoom motor
1525 //
1526 lspmac_movedac_queue( flight, requested_position);
1527 lspmac_movedac_queue( blight, requested_position);
1528 }

```

5.2.4.31 void lspmac_next_state ()

State machine logic. Given the current state, generate the next one

Definition at line 1258 of file lspmac.c.

```

1258                                     {
1259
1260     //
1261     // Connect to the pmac and perhaps initialize it.
1262     // OK, this is slightly more than just the state
1263     // machine logic...
1264     //
1265     if( ls_pmac_state == LS_PMAC_STATE_DETACHED) {
1266         //
1267         // TODO (eventually)
1268         // This ip address wont change in a single PMAC installation
1269         // We'll need to audit the code if we decide to implement
1270         // multiple PMACs so might as well wait til then.
1271         //
1272         lsConnect( "192.6.94.5");
1273
1274         //
1275         // If the connect was successful we can proceed with the initialization
1276         //
1277         if( ls_pmac_state != LS_PMAC_STATE_DETACHED) {
1278             lspmac_SockFlush();
1279
1280             //
1281             // Harvest the I and M variables in case we need them
1282             // one day.
1283             //
1284             if( getmvars) {
1285                 lspmac_GetAllMVars();
1286                 getmvars = 0;
1287             }
1288

```

```

1289         if( getivars) {
1290             lspmac_GetAllIVars();
1291             getivars = 0;
1292         }
1293     }
1294 }
1295
1296 //
1297 // Check the command queue and perhaps go to the "Send Command" state.
1298 //
1299 if( ls_pmac_state == LS_PMAC_STATE_IDLE && ethCmdOn != ethCmdOff)
1300     ls_pmac_state = LS_PMAC_STATE_SC;
1301
1302
1303 //
1304 // Set the events flag
1305 // to tell poll what we are waiting for.
1306 //
1307 switch( ls_pmac_state) {
1308 case LS_PMAC_STATE_DETACHED:
1309     //
1310     // there shouldn't be a valid fd, so ignore the events
1311     //
1312     pmacfd.events = 0;
1313     break;
1314
1315 case LS_PMAC_STATE_IDLE:
1316     if( ethCmdOn == ethCmdOff)
1317         //
1318         // Anytime we are idle we want to
1319         // get the status of the PMAC
1320         //
1321         lspmac_get_status();
1322
1323
1324
1325 //
1326 // These state require that we listen for packets
1327 //
1328 case LS_PMAC_STATE_WACK_NFR:
1329 case LS_PMAC_STATE_WACK:
1330 case LS_PMAC_STATE_WACK_CC:
1331 case LS_PMAC_STATE_WACK_RR:
1332 case LS_PMAC_STATE_WCR:
1333 case LS_PMAC_STATE_WGB:
1334 case LS_PMAC_STATE_GMR:
1335     pmacfd.events = POLLIN;
1336     break;
1337
1338 //
1339 // These state require that we send packets out.
1340 //
1341 case LS_PMAC_STATE_SC:
1342 case LS_PMAC_STATE_CR:
1343 case LS_PMAC_STATE_RR:
1344 case LS_PMAC_STATE_GB:
1345     //
1346     // Sad fact: PMAC will fail to process commands if we send them too quickly.
1347     // We deal with that by waiting a tad before we let poll tell us the PMAC soc
ket is ready to write.
1348     //
1349     gettimeofday( &now, NULL);
1350     if( ((now.tv_sec * 1000000. + now.tv_usec) - (pmac_time_sent.tv_sec * 100000
0. + pmac_time_sent.tv_usec)) < PMAC_MIN_CMD_TIME) {
1351         pmacfd.events = 0;
1352     } else {
1353         pmacfd.events = POLLOUT;

```

```

1354     }
1355     break;
1356 }
1357 }

```

5.2.4.32 void lspmac_pmacmotor_read (lspmac_motor_t * mp)

Read the position and status of a normal PMAC motor.

Parameters:

← *mp* Our motor

Definition at line 981 of file lspmac.c.

```

983     {
984     char s[512], *sp;
985
986     if( *mp->status2 & 0x000001) {
987         if( mp->not_done) {
988             pthread_mutex_lock( &(mp->mutex));
989             mp->not_done = 0;
990             pthread_cond_signal( &(mp->cond));
991             pthread_mutex_unlock( &(mp->mutex));
992         }
993     } else if( mp->not_done == 0) {
994         mp->not_done = 1;
995     }
996
997     if( (*mp->status1 & 0x020000) || (*mp->status1 & 0x000400)) {
998         if( mp->motion_seen == 0) {
999             pthread_mutex_lock( &(mp->mutex));
1000             mp->motion_seen = 1;
1001             pthread_cond_signal( &(mp->cond));
1002             pthread_mutex_unlock( &(mp->mutex));
1003         }
1004     }
1005
1006     mvwprintw( mp->win, 2, 1, "%s", LS_DISPLAY_WINDOW_WIDTH-2, " ");
1007     mvwprintw( mp->win, 2, 1, "%d cts", LS_DISPLAY_WINDOW_WIDTH-6, *mp->
actual_pos_cnts_p);
1008     mvwprintw( mp->win, 3, 1, "%s", LS_DISPLAY_WINDOW_WIDTH-2, " ");
1009
1010     if( mp->u2c != 0.0) {
1011         mp->position = *mp->actual_pos_cnts_p/mp->u2c;
1012         snprintf( s, sizeof(s)-1, mp->format, 8, *mp->actual_pos_cnts_p/mp->u2c);
1013     } else {
1014         mp->position = 1.0* (*mp->actual_pos_cnts_p);
1015         snprintf( s, sizeof(s)-1, mp->format, 8, 1.0* (*mp->actual_pos_cnts_p));
1016     }
1017     s[sizeof(s)-1] = 0;
1018     mvwprintw( mp->win, 3, 1, "%s", LS_DISPLAY_WINDOW_WIDTH-6, s);
1019
1020     mvwprintw( mp->win, 4, 1, "%u", LS_DISPLAY_WINDOW_WIDTH-2, *mp->status1);
1021     mvwprintw( mp->win, 5, 1, "%u", LS_DISPLAY_WINDOW_WIDTH-2, *mp->status2);
1022     sp = "";
1023     if( *mp->status2 & 0x000002)
1024         sp = "Following Warning";
1025     else if( *mp->status2 & 0x000004)
1026         sp = "Following Error";
1027     else if( *mp->status2 & 0x000020)
1028         sp = "I2T Amp Fault";
1029     else if( *mp->status2 & 0x000008)

```

```

1030     sp = "Amp. Fault";
1031     else if( *mp->status2 & 0x000800)
1032         sp = "Stopped on Limit";
1033     else if( *mp->status1 & 0x040000)
1034         sp = "Open Loop";
1035     else if( ~( *mp->status1) & 0x080000)
1036         sp = "Motor Disabled";
1037     else if( *mp->status1 & 0x000400)
1038         sp = "Homing";
1039     else if( ( *mp->status1 & 0x600000) == 0x600000)
1040         sp = "Both Limits Tripped";
1041     else if( *mp->status1 & 0x200000)
1042         sp = "Positive Limit";
1043     else if( *mp->status1 & 0x400000)
1044         sp = "Negative Limit";
1045     else if( ~( *mp->status2) & 0x000400)
1046         sp = "Not Homed";
1047     else if( *mp->status2 & 0x000001)
1048         sp = "In Position";
1049
1050     mvwprintw( mp->win, 6, 1, "%s", LS_DISPLAY_WINDOW_WIDTH-2, sp);
1051     wnoutrefresh( mp->win);
1052 }

```

5.2.4.33 pmac_cmd_queue_t* lspmac_pop_queue ()

Remove the oldest queue item. Used to send command to PMAC. Note that there is a separate reply index to ensure we've know to what command a reply is refering. Returns the item.

Definition at line 438 of file lspmac.c.

```

438                                     {
439     pmac_cmd_queue_t *rtn;
440
441     pthread_mutex_lock( &pmac_queue_mutex);
442
443     if( ethCmdOn == ethCmdOff)
444         rtn = NULL;
445     else {
446         rtn = &(ethCmdQueue[(ethCmdOff++) % PMAC_CMD_QUEUE_LENGTH]);
447         clock_gettime( CLOCK_REALTIME, &(rtn->time_sent));
448     }
449     pthread_mutex_unlock( &pmac_queue_mutex);
450     return rtn;
451 }

```

5.2.4.34 pmac_cmd_queue_t* lspmac_pop_reply ()

Remove the next command queue item that is waiting for a reply. We always need a reply to know we are done with a given command. Returns the item.

Definition at line 458 of file lspmac.c.

```

458                                     {
459     pmac_cmd_queue_t *rtn;
460
461     pthread_mutex_lock( &pmac_queue_mutex);
462
463     if( ethCmdOn == ethCmdReply)
464         rtn = NULL;

```



```

465     else
466         rtn = &(ethCmdQueue[(ethCmdReply++) % PMAC_CMD_QUEUE_LENGTH]);
467
468     pthread_mutex_unlock( &pmac_queue_mutex);
469     return rtn;
470 }

```

5.2.4.35 `pmac_cmd_queue_t* lspmac_push_queue (pmac_cmd_queue_t * cmd)`

Put a new command on the queue. Pointer is returned so caller can evaluate the time command was actually sent.

Parameters:

cmd Command to send to the PMAC

Definition at line 414 of file lspmac.c.

```

416                                     {
417     pmac_cmd_queue_t *rtn;
418
419     pthread_mutex_lock( &pmac_queue_mutex);
420     rtn = &(ethCmdQueue[(ethCmdOn++) % PMAC_CMD_QUEUE_LENGTH]);
421     memcpy( rtn, cmd, sizeof( pmac_cmd_queue_t));
422     rtn->time_sent.tv_sec = 0;
423     rtn->time_sent.tv_nsec = 0;
424     pthread_cond_signal( &pmac_queue_cond);
425     pthread_mutex_unlock( &pmac_queue_mutex);
426
427     return rtn;
428 }

```

5.2.4.36 `void lspmac_Reset ()`

Clear the queue and put the PMAC into a known state.

Definition at line 541 of file lspmac.c.

```

541                                     {
542     ls_pmac_state = LS_PMAC_STATE_IDLE;
543
544     // clear queue
545     ethCmdReply = ethCmdOn;
546     ethCmdOff   = ethCmdOn;
547
548     lspmac_SockFlush();
549 }

```

5.2.4.37 `void lspmac_run ()`

Start up the lspmac thread.

Definition at line 1870 of file lspmac.c.

```

1870                                     {
1871     pthread_create( &pmac_thread, NULL, lspmac_worker, NULL);
1872 }

```

5.2.4.38 `pmac_cmd_queue_t* lspmac_send_command (int rqType, int rq, int wValue, int wIndex, int wLength, unsigned char * data, void(*)(pmac_cmd_queue_t *, int, unsigned char *) responseCB, int no_reply)`

Compose a packet and send it to the PMAC. This is the meat of the PMAC communications routines. The queued command is returned.

Parameters:

- ← *rqType* VR_UPLOAD or VR_DOWNLOAD
- ← *rq* PMAC command (see PMAC User Manual)
- ← *wValue* Command argument 1
- ← *wIndex* Command argument 2
- ← *wLength* Length of data array
- ← *data* Data array (or NULL)
- ← *responseCB* Function to call when a response is read from the PMAC
- ← *no_reply* Flag, non-zero means no reply is expected

Definition at line 476 of file `lspmac.c`.

```

486                                     {
487     static pmac_cmd_queue_t cmd;
488
489     cmd.pcmd.RequestType = rqType;
490     cmd.pcmd.Request     = rq;
491     cmd.pcmd.wValue      = htons(wValue);
492     cmd.pcmd.wIndex      = htons(wIndex);
493     cmd.pcmd.wLength     = htons(wLength);
494     cmd.onResponse       = responseCB;
495     cmd.no_reply         = no_reply;
496
497     //
498     // Setting the message buff bData requires a bit more care to avoid over fillin
499     // or sending garbage in the unused bytes.
500     //
501
502     if( wLength > sizeof( cmd.pcmd.bData)) {
503         //
504         // Bad things happen if we do not catch this case.
505         //
506         pthread_mutex_lock( &ncurses_mutex);
507         wprintw( term_output, "Message Length %d longer than maximum of %ld, aborting
508         \n", wLength, sizeof( cmd.pcmd.bData));
509
510         wnoutrefresh( term_output);
511         wnoutrefresh( term_input);
512         doupdate();
513         pthread_mutex_unlock( &ncurses_mutex);
514         exit( -1);
515     }
516     if( data == NULL) {
517         memset( cmd.pcmd.bData, 0, sizeof( cmd.pcmd.bData));
518     } else {
519         //
520         // This could leave bData non-null terminated. I do not know if this is a pr
521         // oblem.
522         if( wLength > 0)
523             memcpy( cmd.pcmd.bData, data, wLength);

```

```

523     if( wLength < sizeof( cmd.pcmd.bData))
524         memset( cmd.pcmd.bData + wLength, 0, sizeof( cmd.pcmd.bData) - wLength);
525 }
526
527 return lspmac_push_queue( &cmd);
528 }

```

5.2.4.39 void lspmac_sendcmd_nocb (char **fmt*, ...)

Send a command that does not need to deal with the reply.

Parameters:

← *fmt* A printf style format string

Definition at line 1238 of file lspmac.c.

```

1241     {
1242     static char tmps[1024];
1243     va_list arg_ptr;
1244
1245     va_start( arg_ptr, fmt);
1246     vsnprintf( tmps, sizeof(tmps)-1, fmt, arg_ptr);
1247     tmps[sizeof(tmps)-1]=0;
1248     va_end( arg_ptr);
1249
1250     lspmac_send_command( VR_DOWNLOAD, VR_PMAC_SENDLINE, 0, 0, strlen(tmps), tmps, N
        ULL, 0);
1251 }

```

5.2.4.40 void lspmac_SendControlReplyPrintCB (pmac_cmd_queue_t **cmd*, int *nreceived*, unsigned char **buff*)

Receive a reply to a control character Print a "printable" version of the character to the terminal Followed by a hex dump of the response.

Parameters:

← *cmd* Queue item this is a reply to

← *nreceived* Number of bytes received

← *buff* Buffer of bytes received

Definition at line 813 of file lspmac.c.

```

817     {
818     pthread_mutex_lock( &ncurses_mutex);
819     wprintw( term_output, "control-%c: ", '@' + ntohs(cmd->pcmd.wValue));
820     pthread_mutex_unlock( &ncurses_mutex);
821     hex_dump( nreceived, buff);
822     pthread_mutex_lock( &ncurses_mutex);
823     wnoutrefresh( term_output);
824     wnoutrefresh( term_input);
825     doupdate();
826     pthread_mutex_unlock( &ncurses_mutex);
827 }

```

5.2.4.41 void lspmac_Service (struct pollfd * *evt*)

Service routine for packet coming from the PMAC. All communications is asynchronous so this is the only place incoming packets are handled

Parameters:

← *evt* pollfd object returned by poll

Definition at line 586 of file lspmac.c.

```

588     {
589     static unsigned char *receiveBuffer = NULL;    // the buffer inwhich to stick ou
r incoming characters
590     static int receiveBufferSize = 0;            // size of receiveBuffer
591     static int receiveBufferIn = 0;              // next location to write to in r
eceiveBuffer
592     pmac_cmd_queue_t *cmd;                      // maybe the command we are servi
cing
593     ssize_t nsent, nread;                       // nbytes dealt with
594     int i;                                       // loop counter
595     int foundEOCR;                             // end of command response flag
596
597     if( evt->revents & (POLLERR | POLLHUP | POLLNVAL)) {
598         if( evt->fd != -1) {
599             close( evt->fd);
600             evt->fd = -1;
601         }
602         ls_pmac_state = LS_PMAC_STATE_DETACHED;
603         return;
604     }
605
606
607     if( evt->revents & POLLOUT) {
608
609         switch( ls_pmac_state) {
610             case LS_PMAC_STATE_DETACHED:
611                 break;
612             case LS_PMAC_STATE_IDLE:
613                 break;
614
615             case LS_PMAC_STATE_SC:
616                 cmd = lspmac_pop_queue();
617                 if( cmd != NULL) {
618                     if( cmd->pcmd.Request == VR_PMAC_GETMEM) {
619                         nsent = send( evt->fd, cmd, pmac_cmd_size, 0);
620                         if( nsent != pmac_cmd_size) {
621                             pthread_mutex_lock( &ncurses_mutex);
622                             wprintw( term_output, "\nCould only send %d of %d bytes....Not good."
, (int)nsent, (int)(pmac_cmd_size));
623                             wnoutrefresh( term_output);
624                             wnoutrefresh( term_input);
625                             doupdate();
626                             pthread_mutex_unlock( &ncurses_mutex);
627                         }
628                     } else {
629                         nsent = send( evt->fd, cmd, pmac_cmd_size + ntohs(cmd->pcmd.wLength), 0
);
630                         gettimeofday( &pmac_time_sent, NULL);
631                         if( nsent != pmac_cmd_size + ntohs(cmd->pcmd.wLength)) {
632                             pthread_mutex_lock( &ncurses_mutex);
633                             wprintw( term_output, "\nCould only send %d of %d bytes....Not good."
, (int)nsent, (int)(pmac_cmd_size + ntohs(cmd->pcmd.wLength)));
634                             wnoutrefresh( term_output);
635                             wnoutrefresh( term_input);

```

```

636         doupdate();
637         pthread_mutex_unlock( &ncurses_mutex);
638     }
639 }
640 }
641 if( cmd->pcmd.Request == VR_PMAC_SENDCTRLCHAR)
642     ls_pmac_state = LS_PMAC_STATE_WACK_CC;
643 else if( cmd->pcmd.Request == VR_PMAC_GETMEM)
644     ls_pmac_state = LS_PMAC_STATE_GMR;
645 else if( cmd->no_reply == 0)
646     ls_pmac_state = LS_PMAC_STATE_WACK;
647 else
648     ls_pmac_state = LS_PMAC_STATE_WACK_NFR;
649 break;
650
651 case LS_PMAC_STATE_CR:
652     nsent = send( evt->fd, &cr_cmd, pmac_cmd_size, 0);
653     gettimeofday( &pmac_time_sent, NULL);
654     ls_pmac_state = LS_PMAC_STATE_WCR;
655     break;
656
657 case LS_PMAC_STATE_RR:
658     nsent = send( evt->fd, &rr_cmd, pmac_cmd_size, 0);
659     gettimeofday( &pmac_time_sent, NULL);
660     ls_pmac_state = LS_PMAC_STATE_WACK_RR;
661     break;
662
663 case LS_PMAC_STATE_GB:
664     nsent = send( evt->fd, &gb_cmd, pmac_cmd_size, 0);
665     gettimeofday( &pmac_time_sent, NULL);
666     ls_pmac_state = LS_PMAC_STATE_WGB;
667     break;
668 }
669 }
670
671 if( evt->revents & POLLIN) {
672
673     if( receiveBufferSize - receiveBufferIn < 1400) {
674         unsigned char *newbuff;
675
676         receiveBufferSize += 1400;
677         newbuff = calloc( receiveBufferSize, sizeof( unsigned char));
678         if( newbuff == NULL) {
679             pthread_mutex_lock( &ncurses_mutex);
680             wprintw( term_output, "\nOut of memory\n");
681             wnoutrefresh( term_output);
682             wnoutrefresh( term_input);
683             doupdate();
684             pthread_mutex_unlock( &ncurses_mutex);
685             exit( -1);
686         }
687         memcpy( newbuff, receiveBuffer, receiveBufferIn);
688         receiveBuffer = newbuff;
689     }
690
691     nread = read( evt->fd, receiveBuffer + receiveBufferIn, 1400);
692
693     foundEOCR = 0;
694     if( ls_pmac_state == LS_PMAC_STATE_GMR) {
695         //
696         // get memory returns binary stuff, don't try to parse it
697         //
698         receiveBufferIn += nread;
699     } else {
700         //
701         // other commands end in 6 if OK, 7 if not
702         //

```

```
703     for( i=receiveBufferIn; i<receiveBufferIn+nread; i++) {
704         if( receiveBuffer[i] == 7) {
705             //
706             // Error condition
707             //
708             lspmac_Error( &(amp;receiveBuffer[i]));
709             receiveBufferIn = 0;
710             return;
711         }
712         if( receiveBuffer[i] == 6) {
713             //
714             // End of command response
715             //
716             foundEOCR = 1;
717             receiveBuffer[i] = 0;
718             break;
719         }
720     }
721     receiveBufferIn = i;
722 }
723
724 cmd = NULL;
725
726 switch( ls_pmac_state) {
727 case LS_PMAC_STATE_WACK_NFR:
728     receiveBuffer[--receiveBufferIn] = 0;
729     cmd = lspmac_pop_reply();
730     ls_pmac_state = LS_PMAC_STATE_IDLE;
731     break;
732 case LS_PMAC_STATE_WACK:
733     receiveBuffer[--receiveBufferIn] = 0;
734     ls_pmac_state = LS_PMAC_STATE_RR;
735     break;
736 case LS_PMAC_STATE_WACK_CC:
737     receiveBuffer[--receiveBufferIn] = 0;
738     ls_pmac_state = LS_PMAC_STATE_CR;
739     break;
740 case LS_PMAC_STATE_WACK_RR:
741     receiveBufferIn -= 2;
742     if( receiveBuffer[receiveBufferIn])
743         ls_pmac_state = LS_PMAC_STATE_GB;
744     else
745         ls_pmac_state = LS_PMAC_STATE_RR;
746     receiveBuffer[receiveBufferIn] = 0;
747     break;
748 case LS_PMAC_STATE_GMR:
749     cmd = lspmac_pop_reply();
750     ls_pmac_state = LS_PMAC_STATE_IDLE;
751     break;
752
753 case LS_PMAC_STATE_WCR:
754     cmd = lspmac_pop_reply();
755     ls_pmac_state = LS_PMAC_STATE_IDLE;
756     break;
757 case LS_PMAC_STATE_WGB:
758     if( foundEOCR) {
759         cmd = lspmac_pop_reply();
760         ls_pmac_state = LS_PMAC_STATE_IDLE;
761     } else {
762         ls_pmac_state = LS_PMAC_STATE_RR;
763     }
764     break;
765 }
766
767
768 if( cmd != NULL && cmd->onResponse != NULL) {
769     cmd->onResponse( cmd, receiveBufferIn, receiveBuffer);
770 }
```

```

770     receiveBufferIn = 0;
771 }
772 }
773 }

```

5.2.4.42 void lspmac_shutter_read (lspmac_motor_t * mp)

Fast shutter read routine The shutter is mildly complicated in that we need to take into account the fact that the shutter can open and close again between status updates. This means that we need to rely on a PCL program running in the PMAC to monitor the shutter state and let us know that this has happened.

Parameters:

← *mp* The motor object associated with the fast shutter

5.2.4.43 void lspmac_SockFlush ()

Reset the PMAC socket from the PMAC side. Puts the PMAC into a known communications state

Definition at line 534 of file lspmac.c.

```

534     {
535     lspmac_send_command( VR_DOWNLOAD, VR_PMAC_FLUSH, 0, 0, 0, NULL, NULL, 1);
536 }

```

5.2.4.44 pmac_cmd_queue_t* lspmac_SockGetmem (int offset, int nbytes)

Request a chunk of memory to be returned. Not currently used

Parameters:

← *offset* Offset in PMAC Double Buffer

← *nbytes* Number of bytes to request

Definition at line 850 of file lspmac.c.

```

853     {
854     return lspmac_send_command( VR_UPLOAD, VR_PMAC_GETMEM, offset, 0, nbytes, NULL,
855                                lspmac_GetmemReplyCB, 0);
856 }

```

5.2.4.45 pmac_cmd_queue_t* lspmac_SockSendControlCharPrint (char c)

Send a control character.

Parameters:

c The control character to send

Definition at line 894 of file lspmac.c.

```

896     {
897     return lspmac_send_command( VR_DOWNLOAD, VR_PMAC_SENDCTRLCHAR, c, 0, 0, NULL,
898                                lspmac_SendControlReplyPrintCB, 0);
899 }

```

5.2.4.46 `pmac_cmd_queue_t* lspmac_SockSendline (char * fmt, ...)`

Send a one line command. Uses printf style arguments.

Parameters:

← *fmt* Printf style format string

Definition at line 860 of file `lspmac.c`.

```

863                                     {
864     va_list arg_ptr;
865     char payload[1400];
866
867     va_start( arg_ptr, fmt);
868     vsnprintf( payload, sizeof(payload)-1, fmt, arg_ptr);
869     payload[ sizeof(payload)-1] = 0;
870     va_end( arg_ptr);
871
872     return lspmac_send_command( VR_DOWNLOAD, VR_PMAC_SENDBLINE, 0, 0, strlen( payloa
873 d), payload, lspmac_GetShortReplyCB, 0);
874 }
```

5.2.4.47 `pmac_cmd_queue_t* lspmac_SockSendline_nr (char * fmt, ...)`

Send a command and ignore the response.

Parameters:

← *fmt* Printf style format string

Definition at line 877 of file `lspmac.c`.

```

880                                     {
881     va_list arg_ptr;
882     char s[512];
883
884     va_start( arg_ptr, fmt);
885     vsnprintf( s, sizeof(s)-1, fmt, arg_ptr);
886     s[sizeof(s)-1] = 0;
887     va_end( arg_ptr);
888
889     return lspmac_send_command( VR_DOWNLOAD, VR_PMAC_SENDBLINE, 0, 0, strlen( s), s,
890 NULL, 1);
891 }
```

5.2.4.48 `void* lspmac_worker (void * dummy)`

Our lspmac worker thread.

Parameters:

← *dummy* Unused but required by pthread library

Definition at line 1362 of file `lspmac.c`.


```
1364         {
1365
1366     while( 1) {
1367         int pollrtn;
1368
1369         lspmac_next_state();
1370
1371         if( pmacfd.fd == -1) {
1372             sleep( 10);          // The pmac is not connected.  Should we warn someone?
1373             //
1374             // This just puts us into a holding pattern until the pmac becomes connecte
1375         d again
1376         //
1377         // TODO:
1378         // Check PMAC initialization logic and our queues to ensure that it is sane
1379         to
1380         // re-initialize things.  Probably bad things will happen.
1381         //
1382         continue;
1383     }
1384     pollrtn = poll( &pmacfd, 1, 10);
1385     if( pollrtn) {
1386         lspmac_Service( &pmacfd);
1387     }
1388 }
```

5.2.5 Variable Documentation

5.2.5.1 lspmac_motor_t* alignx

Alignment stage X.

Definition at line 74 of file lspmac.c.

5.2.5.2 lspmac_motor_t* aligny

Alignment stage Y.

Definition at line 75 of file lspmac.c.

5.2.5.3 lspmac_motor_t* alignz

Alignment stage X.

Definition at line 76 of file lspmac.c.

5.2.5.4 lspmac_motor_t* anal

Polaroid analyzer motor.

Definition at line 77 of file lspmac.c.

5.2.5.5 lspmac_motor_t* apery

Aperture Y.

Definition at line 79 of file lspmac.c.

5.2.5.6 lspmac_motor_t* aperz

Aperture Z.

Definition at line 80 of file lspmac.c.

5.2.5.7 lspmac_motor_t* blight

Back Light DAC.

Definition at line 91 of file lspmac.c.

5.2.5.8 lspmac_motor_t* blight_ud

Back Light Up/Down actuator.

Definition at line 94 of file lspmac.c.

5.2.5.9 lspmac_motor_t* capy

Capillary Y.

Definition at line 81 of file lspmac.c.

5.2.5.10 lspmac_motor_t* capz

Capillary Z.

Definition at line 82 of file lspmac.c.

5.2.5.11 lspmac_motor_t* cenx

Centering Table X.

Definition at line 84 of file lspmac.c.

5.2.5.12 lspmac_motor_t* ceny

Centering Table Y.

Definition at line 85 of file lspmac.c.

5.2.5.13 pmac_cmd_t cr_cmd [static]

commands to send out "readready", "getbuffer", controlresponse (initialized in main)

Definition at line 138 of file lspmac.c.

5.2.5.14 unsigned char dbmem[64 * 1024] [static]

double buffered memory

Definition at line 128 of file lspmac.c.

5.2.5.15 int dbmemIn = 0 [static]

next location

Definition at line 129 of file lspmac.c.

5.2.5.16 unsigned int ethCmdOff = 0 [static]

points to current command (or none if == ethCmdOn)

Definition at line 141 of file lspmac.c.

5.2.5.17 unsigned int ethCmdOn = 0 [static]

points to next empty PMAC command queue position

Definition at line 140 of file lspmac.c.

5.2.5.18 pmac_cmd_queue_t ethCmdQueue[PMAC_CMD_QUEUE_LENGTH] [static]

PMAC command queue.

Definition at line 139 of file lspmac.c.

5.2.5.19 unsigned int ethCmdReply = 0 [static]

Used like ethCmdOff only to deal with the pmac reply to a command.

Definition at line 142 of file lspmac.c.

5.2.5.20 lspmac_motor_t* flight

Front Light DAC.

Definition at line 90 of file lspmac.c.

5.2.5.21 lspmac_motor_t* fscint

Scintillator Piezo DAC.

Definition at line 92 of file lspmac.c.

5.2.5.22 lspmac_motor_t* fshut

Fast shutter.

Definition at line 89 of file lspmac.c.

5.2.5.23 pmac_cmd_t gb_cmd [static]

Definition at line 138 of file lspmac.c.

5.2.5.24 int getivars = 0 [static]

flag set at initialization to send i vars to db

Definition at line 68 of file lspmac.c.

5.2.5.25 int getmvars = 0 [static]

flag set at initialization to send m vars to db

Definition at line 69 of file lspmac.c.

5.2.5.26 lspmac_motor_t* kappa

Kappa.

Definition at line 86 of file lspmac.c.

5.2.5.27 int linesReceived = 0 [static]

current number of lines received

Definition at line 127 of file lspmac.c.

5.2.5.28 int ls_pmac_state = LS_PMAC_STATE_DETACHED [static]

Current state of the PMAC communications state machine.

Definition at line 52 of file lspmac.c.

5.2.5.29 lspmac_motor_t lspmac_motors[32]

All our motors.

Definition at line 71 of file lspmac.c.

5.2.5.30 pthread_cond_t lspmac_moving_cond

Wait for motor(s) to finish moving condition.

Definition at line 59 of file lspmac.c.

5.2.5.31 int lspmac_moving_flags

Flag used to implement motor moving condition.

Definition at line 60 of file lspmac.c.

5.2.5.32 pthread_mutex_t lspmac_moving_mutex

Coordinate moving motors between threads.

Definition at line 58 of file lspmac.c.

5.2.5.33 int lspmac_nmotors = 0

The number of motors we manage.

Definition at line 72 of file lspmac.c.

5.2.5.34 pthread_cond_t lspmac_shutter_cond

Allows waiting for the shutter status to change.

Definition at line 57 of file lspmac.c.

5.2.5.35 lspmac_shutter_has_opened = md2_status.fs_has_opened

Indicates that the shutter had opened, perhaps briefly even if the state did not change.

Definition at line 55 of file lspmac.c.

5.2.5.36 pthread_mutex_t lspmac_shutter_mutex

Coordinates threads reading shutter status.

Definition at line 56 of file lspmac.c.

5.2.5.37 int lspmac_shutter_state

State of the shutter, used to detect changes.

Definition at line 54 of file lspmac.c.

5.2.5.38 md2_status_t md2_status [static]

Buffer for MD2 Status.

Definition at line 276 of file lspmac.c.

5.2.5.39 pthread_mutex_t md2_status_mutex

Synchronize reading/writing status buffer.

Definition at line 277 of file lspmac.c.

5.2.5.40 struct timeval pmac_time_sent now [static]

used to ensure we do not send commands to the pmac too often. Only needed for non-DB commands.

Definition at line 134 of file lspmac.c.

5.2.5.41 lspmac_motor_t* omega

MD2 omega axis (the air bearing).

Definition at line 73 of file lspmac.c.

5.2.5.42 lspmac_motor_t* phi

Phi (not data collection axis).

Definition at line 87 of file lspmac.c.

5.2.5.43 char* pmac_error_strs[] [static]

Initial value:

```
{
  "ERR000: Unknown error",
  "ERR001: Command not allowed during program execution",
  "ERR002: Password error",
  "ERR003: Data error or unrecognized command",
  "ERR004: Illegal character",
  "ERR005: Command not allowed unless buffer is open",
  "ERR006: No room in buffer for command",
  "ERR007: Buffer already in use",
  "ERR008: MACRO auxiliary communication error",
  "ERR009: Program structure error (e.g. ENDIF without IF)",
  "ERR010: Both overtravel limits set for a motor in the C.S.",
  "ERR011: Previous move not completed",
  "ERR012: A motor in the coordinate system is open-loop",
  "ERR013: A motor in the coordinate system is not activated",
  "ERR014: No motors in the coordinate system",
  "ERR015: Not pointer to valid program buffer",
  "ERR016: Running improperly structure program (e.g. missing ENDWHILE)",
  "ERR017: Trying to resume after H or Q with motors out of stopped position",
  "ERR018: Attempt to perform phase reference during move, move during phase reference, or enabling with phase clock error",
  "ERR019: Illegal position-change command while moves stored in CCBUFFER"
}
```

Decode the errors perhaps returned by the PMAC.

Definition at line 145 of file lspmac.c.

5.2.5.44 pthread_cond_t pmac_queue_cond [static]

wait for a command to be sent to PMAC before continuing

Definition at line 65 of file lspmac.c.

5.2.5.45 pthread_mutex_t pmac_queue_mutex [static]

manage access to the pmac command queue

Definition at line 64 of file lspmac.c.

5.2.5.46 pthread_t pmac_thread [static]

our thread to manage access and communication to the pmac

Definition at line 63 of file lspmac.c.

5.2.5.47 struct pollfd pmacfd [static]

our poll structure

Definition at line 66 of file lspmac.c.

5.2.5.48 pmac_cmd_t rr_cmd [static]

Definition at line 138 of file lspmac.c.

5.2.5.49 lspmac_motor_t* scinz

Scintillator Z.

Definition at line 83 of file lspmac.c.

5.2.5.50 lspmac_motor_t* zoom

Optical zoom.

Definition at line 78 of file lspmac.c.

5.3 lsupdate.c File Reference

Brings this MD2 code and the database kvs table into agreement. `#include "pgpmac.h"`

Functions

- void [lsupdate_updateit](#) (int first_time)
Query the motors and perhaps tell the DB about it.
- void * [lsupdate_worker](#) (void *dummy)
Our worker thread.
- void [lsupdate_init](#) ()
Initialize this module.
- void [lsupdate_run](#) ()
run the update routines

Variables

- static pthread_t [lsupdate_thread](#)
our worker thread

5.3.1 Detailed Description

Brings this MD2 code and the database kvs table into agreement.

Date:

2012

Author:

Keith Brister All Rights Reserved

Definition in file [lsupdate.c](#).

5.3.2 Function Documentation

5.3.2.1 void lsupdate_init ()

Initialize this module.

Definition at line 89 of file [lsupdate.c](#).

```
89         {  
90     }
```


5.3.2.2 void lsupdate_run ()

run the update routines

Definition at line 94 of file lsupdate.c.

```

94         {
95     // pthread_create( &lsupdate_thread, NULL, lsupdate_worker, NULL);
96 }

```

5.3.2.3 void lsupdate_updateit (int *first_time*)

Query the motors and perhaps tell the DB about it.

Parameters:

← *first_time* Flag: 1 means update everything, 0 means only send stuff that has changed

Definition at line 15 of file lsupdate.c.

```

16                                     : 1 means update ev
    everything, 0 means only send stuff that has changed */
17     ) {
18     static char s[4096];
19     static char s1[512];
20     lspmac_motor_t *mp;
21     int i;
22     int needComma;
23     int gotone;
24
25     needComma = 0;
26     gotone = 0;
27     s[0] = 0;
28     strcpy(s, "select px.kvupdate('{");
29
30     for( i=0; i<lspmac_nmotors; i++) {
31         mp = &(lspmac_motors[i]);
32
33         pthread_mutex_lock( &(mp->mutex));
34         if( fabs( mp->position - mp->reported_position) < mp->update_resolution && fi
rst_time == 0) {
35             pthread_mutex_unlock( &(mp->mutex));
36         } else {
37
38             gotone = 1;
39             s1[0]=0;
40
41             snprintf( s1, sizeof(s1)-1, mp->format, mp->position, sizeof( s1)-1);
42             s1[sizeof(s1)-1] = 0;
43
44             mp->reported_position = mp->position;
45             pthread_mutex_unlock( &(mp->mutex));
46
47             if( strlen(s1) + strlen(s) + 8 >= sizeof( s)-1) {
48                 // send off update now and reset s
49                 strcat( s, "}'");
50                 lspg_query_push( NULL, s);
51
52                 s[0] = 0;
53                 strcpy( s, "select px.kvupdate('{");
54                 needComma = 0;
55             }

```

```
56
57     if( needComma)
58         strcat( s, ",");
59     else
60         needComma=1;
61
62     strcat( s, "\"");
63     strcat( s, s1);
64     strcat( s, "\"");
65 }
66 }
67
68 if( gotone) {
69     strcat( s, "}'");
70     lspg_query_push( NULL, s);
71 }
72 }
```

5.3.2.4 void* lsupdate_worker (void * *dummy*)

Our worker thread.

Parameters:

← *dummy* Unused argument required by protocol

Definition at line 76 of file lsupdate.c.

```
78                                     {
79     sleep(10);
80     lsupdate_updateit( 1);
81     while( 1) {
82         usleep( 500000);
83         lsupdate_updateit( 0);
84     }
85 }
```

5.3.3 Variable Documentation

5.3.3.1 pthread_t lsupdate_thread [static]

our worker thread

Definition at line 10 of file lsupdate.c.

5.4 md2cmds.c File Reference

Implements commands to run the md2 diffractometer attached to a PMAC controlled by postgresql.
`#include "pgpmac.h"`

Functions

- void `md2cmds_transfer` ()
Transfer a sample TODO: Implement.
- char * `logtime` ()
Return a time string for loggin Time is from the first call to this funciton.
- void `md2cmds_moveAbs` (char *cmd)
Move a motor to the position requested.
- void `md2cmds_mvcenter_prep` ()
Sets up a centering table and alignment table move Ensures that when we issue the move command that we can detect that the move happened.
- void `md2cmds_mvcenter_move` (double cx, double cy, double ax, double ay, double az)
Move the centering and alignment tables.
- void `md2cmds_mvcenter_wait` ()
Wait for the centering and alignment tables to stop moving.
- void `md2cmds_collect` ()
Collect some data.
- void `md2cmds_rotate` ()
Spin 360 and make a video TODO: Implement.
- void `md2cmds_center` ()
Move centering and alignment tables as requested TODO: Implement.
- void * `md2cmds_worker` (void *dummy)
Our worker thread.
- void `md2cmds_init` ()
Initialize the md2cmds module.
- void `md2cmds_run` ()
Start up the thread.

Variables

- pthread_cond_t [md2cmds_cond](#)
condition to signal when it's time to run an md2 command
- pthread_mutex_t [md2cmds_mutex](#)
mutex for the condition
- pthread_cond_t [md2cmds_pg_cond](#)
coordinate call and response
- pthread_mutex_t [md2cmds_pg_mutex](#)
message passing between md2cmds and pg
- char [md2cmds_cmd](#) [MD2CMDS_CMD_LENGTH]
our command;
- static pthread_t [md2cmds_thread](#)

5.4.1 Detailed Description

Implements commands to run the md2 diffractometer attached to a PMAC controled by postgresql.

Date:

2012

Author:

Keith Brister All Rights Reserved

Definition in file [md2cmds.c](#).

5.4.2 Function Documentation

5.4.2.1 char* logtime ()

Return a time string for login Time is from the first call to this funciton.

Definition at line 30 of file md2cmds.c.

```

30      {
31  static char rtn[128];
32  static char tmp[64];
33  static int first_time = 1;
34  static struct timeval base;
35  struct timeval now;
36  struct tm nows;
37  double diffs;
38
39  if( first_time) {
40      first_time=0;
41      gettimeofday( &base, NULL);
42      strftime(tmp, sizeof(tmp)-1, "%Y-%m-%d %H:%M:%S", localtime( &(base.tv_sec)))
      ;

```

```

43     tmp[sizeof(tmp)-1]=0;
44     snprintf( rtn, sizeof(rtn)-1, "%s.%06d", tmp, base.tv_usec);
45     rtn[sizeof(rtn)-1]=0;
46 } else {
47     gettimeofday( &now, NULL);
48     diffs = (now.tv_sec - base.tv_sec);
49     diffs += (now.tv_usec - base.tv_usec)/1000000.;
50     snprintf( rtn, sizeof( rtn)-1, "%0.6f", diffs);
51     rtn[sizeof(rtn)-1]=0;
52 }
53
54 return rtn;
55 }

```

5.4.2.2 void md2cmds_center ()

Move centering and alignment tables as requested TODO: Implement.

Definition at line 422 of file md2cmds.c.

```

422     {
423 }

```

5.4.2.3 void md2cmds_collect ()

Collect some data.

Definition at line 214 of file md2cmds.c.

```

214     {
215     long long skey;
216     double p170; // start cnts
217     double p171; // end cnts
218     double p173; // omega velocity cnts/msec
219     double p175; // acceleration time (msec)
220     double p180; // exposure time (msec)
221     FILE *zzlog;
222     struct timeval tt_base, tt_now;
223     int center_request;
224
225     zzlog = fopen( "/tmp/collect_log.txt", "w");
226     fprintf( zzlog, "%s: Start md2cmds\n", logtime());
227     fflush( zzlog);
228
229     //
230     // reset shutter has opened flag
231     //
232     lspmac_SockSendline( "P3001=0 P3002=0");
233
234
235     while( 1) {
236         fprintf( zzlog, "%s: call lspg_nextshot_call\n", logtime());
237         fflush( zzlog);
238         lspg_nextshot_call();
239
240         //
241         // This is where we'd tell the md2 to move the organs into position
242         //
243
244         fprintf( zzlog, "%s: call lspg_nextshot_wait\n", logtime());
245         fflush( zzlog);

```

```

246     lspg_nextshot_wait();
247     fprintf( zzlog, "%s: returned from  lspg_nextshot_wait\n", logtime());
248     fflush( zzlog);
249
250
251     if( lspg_nextshot.no_rows_returned) {
252         lspg_nextshot_done();
253         break;
254     }
255
256     skey = lspg_nextshot.skey;
257     lspg_query_push( NULL, "SELECT px.shots_set_state(%lld, 'Preparing')", skey);
258
259
260     center_request = 0;
261     if( lspg_nextshot.active) {
262         if(
263             (fabs( lspg_nextshot.cx - cenx->position) > 0.1) ||
264             (fabs( lspg_nextshot.cy - ceny->position) > 0.1) ||
265             (fabs( lspg_nextshot.ax - alignx->position) > 0.1) ||
266             (fabs( lspg_nextshot.ay - aligny->position) > 0.1) ||
267             (fabs( lspg_nextshot.az - alignz->position) > 0.1)) {
268             center_request = 1;
269             md2cmds_mvcenter_prep();
270             md2cmds_mvcenter_move( lspg_nextshot.cx, lspg_nextshot.cy, lspg_nextshot.
ax, lspg_nextshot.ay, lspg_nextshot.az);
271         }
272     }
273
274     if( !lspg_nextshot.dsphi_isnull) {
275         lspmac_moveabs_queue( phi, lspg_nextshot.dsphi);
276     }
277
278     if( !lspg_nextshot.dskappa_isnull) {
279         lspmac_moveabs_queue( kappa, lspg_nextshot.dskappa);
280     }
281
282
283     //
284     // Wait for all those motors to stop
285     //
286     if( center_request) {
287         md2cmds_mvcenter_wait();
288     }
289
290     if( !lspg_nextshot.dsphi_isnull) {
291         lspmac_moveabs_wait( phi);
292     }
293
294     if( !lspg_nextshot.dskappa_isnull) {
295         lspmac_moveabs_wait( kappa);
296     }
297
298     //
299     // Calculate the parameters we'll need to run the scan
300     //
301     p180 = lspg_nextshot.dsexp * 1000.0;
302     p170 = omega->u2c * lspg_nextshot.sstart;
303     //    p171 = omega->u2c * ( lspg_nextshot.sstart + lspg_nextshot.dsowidth);
304     p171 = omega->u2c * lspg_nextshot.dsowidth;
305     p173 = fabs(p180) < 1.e-4 ? 0.0 : omega->u2c * lspg_nextshot.dsowidth / p180;
306
307     p175 = p173/omega->max_accel;
308
309     //

```

```
310     // free up access to nextshot
311     //
312     lspg_nextshot_done();
313
314     fprintf( zzlog, "%s: finished with lspg_nextshot_done, calling lspg_seq_run_p
rep_all\n", logtime());
315     fflush( zzlog);
316
317     //
318     // prepare the database and detector to expose
319     // On exit we own the diffractometer lock and
320     // have checked that all is OK with the detector
321     //
322     lspg_seq_run_prep_all( skey,
323                           kappa->position,
324                           phi->position,
325                           cenx->position,
326                           ceny->position,
327                           alignx->position,
328                           aligny->position,
329                           alignz->position
330                           );
331
332
333     fprintf( zzlog, "%s: finished with lspg_seq_run_prep_all\n", logtime());
334     fflush( zzlog);
335     //
336     // make sure our has opened flag is down
337     // wait for the p3001=0 command to be noticed
338     //
339     pthread_mutex_lock( &lspmac_shutter_mutex);
340     if( lspmac_shutter_has_opened == 1)
341         pthread_cond_wait( &lspmac_shutter_cond, &lspmac_shutter_mutex);
342     pthread_mutex_unlock( &lspmac_shutter_mutex);
343
344     //
345     // Start the exposure
346     //
347     lspmac_SockSendline( "P170=%.1f P171=%.1f P173=%.1f P174=0 P175=%.1f P176=0 P
177=1 P178=0 P180=%.1f M431=1 &lB131R",
348                         p170,      p171,      p173,      p175,
349                         p180);
350
351     fprintf( zzlog, "%s: sent command to pmac\n", logtime());
352     fflush( zzlog);
353
354     //
355     // wait for the shutter to open
356     //
357     pthread_mutex_lock( &lspmac_shutter_mutex);
358     if( lspmac_shutter_has_opened == 0)
359         pthread_cond_wait( &lspmac_shutter_cond, &lspmac_shutter_mutex);
360
361     fprintf( zzlog, "%s: shutter has opened\n", logtime());
362     fflush( zzlog);
363
364     //
365     // wait for the shutter to close
366     //
367     if( lspmac_shutter_state == 1)
368         pthread_cond_wait( &lspmac_shutter_cond, &lspmac_shutter_mutex);
369     pthread_mutex_unlock( &lspmac_shutter_mutex);
370
371     fprintf( zzlog, "%s: shutter now closed, unlocking diffractometer\n",
logtime());
372     fflush( zzlog);
```

```

373
374
375     lspg_query_push( NULL, "SELECT px.unlock_diffractionmeter()");
376
377     fprintf( zzlog, "%s: unlocked diffractionmeter\n", logtime());
378     fflush( zzlog);
379
380     lspg_query_push( NULL, "SELECT px.shots_set_state(%lld, 'Writing')", skey);
381
382     //
383     // reset shutter has opened flag
384     //
385     lspmac_SockSendline( "P3001=0");
386     //
387     // TODO:
388     // wait for omega to stop moving then position it for the next frame
389     //
390
391
392     if( !lspg_nextshot.active2_isnull && lspg_nextshot.active2) {
393         if(
394             (fabs( lspg_nextshot.cx2 - cenx->position) > 0.1) ||
395             (fabs( lspg_nextshot.cy2 - ceny->position) > 0.1) ||
396             (fabs( lspg_nextshot.ax2 - alignx->position) > 0.1) ||
397             (fabs( lspg_nextshot.ay2 - aligny->position) > 0.1) ||
398             (fabs( lspg_nextshot.az2 - alignz->position) > 0.1)) {
399
400             center_request = 1;
401             md2cmds_mvcenter_prep();
402             md2cmds_mvcenter_move( lspg_nextshot.cx, lspg_nextshot.cy, lspg_nextshot.
ax, lspg_nextshot.ay, lspg_nextshot.az);
403             md2cmds_mvcenter_wait();
404         }
405     }
406
407 }
408 fprintf( zzlog, "%s: done\n", logtime());
409 fflush( zzlog);
410 fclose( zzlog);
411 }

```

5.4.2.4 void md2cmds_init ()

Initialize the md2cmds module.

Definition at line 461 of file md2cmds.c.

```

461     {
462     memset( md2cmds_cmd, 0, sizeof( md2cmds_cmd));
463
464     pthread_mutex_init( &md2cmds_mutex, NULL);
465     pthread_cond_init( &md2cmds_cond, NULL);
466
467     pthread_mutex_init( &md2cmds_pg_mutex, NULL);
468     pthread_cond_init( &md2cmds_pg_cond, NULL);
469
470 }

```

5.4.2.5 void md2cmds_moveAbs (char * cmd)

Move a motor to the position requested.

Parameters:

← *cmd* The full command string to parse, ie, "moveAbs omega 180"

Definition at line 59 of file md2cmds.c.

```

61         {
62     char *ignore;
63     char *ptr;
64     char *mtr;
65     char *pos;
66     double fpos;
67     char *endptr;
68     lspmac_motor_t *mp;
69     int i;
70
71     // Parse the command string
72     //
73     ignore = strtok_r( cmd, " ", &ptr);
74     if( ignore == NULL) {
75         //
76         // Should generate error message
77         // about blank command
78         //
79         return;
80     }
81
82     // The first string should be "moveAbs" cause that's how we got here.
83     // Toss it.
84
85     mtr = strtok_r( NULL, " ", &ptr);
86     if( mtr == NULL) {
87         //
88         // Should generate error message
89         // about missing motor name
90         //
91         return;
92     }
93
94     pos = strtok_r( NULL, " ", &ptr);
95     if( pos == NULL) {
96         //
97         // Should generate error message
98         // about missing position
99         //
100        return;
101    }
102
103    fpos = strtod( pos, &endptr);
104    if( pos == endptr) {
105        //
106        // Should generate error message
107        // about bad double conversion
108        //
109        return;
110    }
111
112    mp = NULL;
113    for( i=0; i<lspmac_nmotors; i++) {
114        if( strcmp( lspmac_motors[i].name, mtr) == 0) {
115            mp = &(lspmac_motors[i]);
116            break;
117        }
118    }
119
120
121    if( mp != NULL && mp->moveAbs != NULL) {

```

```

122     wprintw( term_output, "Moving %s to %f\n", mtr, fpos);
123     wnoutrefresh( term_output);
124     mp->moveAbs( mp, fpos);
125 }
126
127 }

```

5.4.2.6 void md2cmds_mvcenter_move (double cx, double cy, double ax, double ay, double az)

Move the centering and alignment tables.

Parameters:

- ← **cx** Requested Centering Table X
- ← **cy** Requested Centering Table Y
- ← **ax** Requested Alignment Table X
- ← **ay** Requested Alignment Table Y
- ← **az** Requested Alignment Table Z

Definition at line 173 of file md2cmds.c.

```

179                                     {
180     //
181     // centering stage is coordinate system 2
182     // alignment stage is coordinate system 3
183     //
184
185     double cx_cts, cy_cts, ax_cts, ay_cts, az_cts;
186
187     cx_cts = cenx->u2c * cx;
188     cy_cts = ceny->u2c * cy;
189     ax_cts = alignx->u2c * ax;
190     ay_cts = aligny->u2c * ay;
191     az_cts = alignz->u2c * az;
192
193     lspmac_SockSendline( "M7075=(M7075 | 2) &2 Q100=2 Q20=%.1f Q21=%.1f B150R", cx_
        cts, cy_cts);
194     lspmac_SockSendline( "M7075=(M7075 | 4) &3 Q100=4 Q30=%.1f Q31=%.1f Q32=%.1f B1
        60R", ax_cts, ay_cts, az_cts);
195
196 }

```

5.4.2.7 void md2cmds_mvcenter_prep ()

Sets up a centering table and alignment table move Ensures that when we issue the move command that we can detect that the move happened.

Definition at line 134 of file md2cmds.c.

```

134                                     {
135     //
136     // Clears the motion flags for coordinate systems 2 and 3
137     // Then sets them.
138     // Each time we wait until we've read back
139     // the changed values
140     //
141     // This guarantees that when we are waiting for motion to stop that it did, in

```

```

        fact, start
142 //
143
144 //
145 // Clear the centering and alignment stage flags
146 //
147 lspmac_SockSendline( "M7075=(M7075 | 6) ^ 6");
148
149 //
150 // Make sure it propagates
151 //
152 pthread_mutex_lock( &lspmac_moving_mutex);
153 while( lspmac_moving_flags & 6)
154     pthread_cond_wait( &lspmac_moving_cond, &lspmac_moving_mutex);
155 pthread_mutex_unlock( &lspmac_moving_mutex);
156
157 //
158 // Set the centering and alignment stage flags
159 //
160 lspmac_SockSendline( "M7075=(M7075 | 6)");
161
162 //
163 // Make sure it propagates
164 //
165 pthread_mutex_lock( &lspmac_moving_mutex);
166 while( (lspmac_moving_flags & 6) == 0)
167     pthread_cond_wait( &lspmac_moving_cond, &lspmac_moving_mutex);
168 pthread_mutex_unlock( &lspmac_moving_mutex);
169 }

```

5.4.2.8 void md2cmds_mvcenter_wait ()

Wait for the centering and alignment tables to stop moving.

Definition at line 200 of file md2cmds.c.

```

200                                     {
201 //
202 // Just wait until the motion flags are lowered
203 //
204
205 pthread_mutex_lock( &lspmac_moving_mutex);
206 while( lspmac_moving_flags & 6)
207     pthread_cond_wait( &lspmac_moving_cond, &lspmac_moving_mutex);
208 pthread_mutex_unlock( &lspmac_moving_mutex);
209 }

```

5.4.2.9 void md2cmds_rotate ()

Spin 360 and make a video TODO: Implement.

Definition at line 416 of file md2cmds.c.

```

416                                     {
417 }

```

5.4.2.10 void md2cmds_run ()

Start up the thread.

Definition at line 474 of file md2cmds.c.

```

474         {
475     pthread_create( &md2cmds_thread, NULL, md2cmds_worker, NULL);
476 }

```

5.4.2.11 void md2cmds_transfer ()

Transfer a sample TODO: Implement.

Definition at line 24 of file md2cmds.c.

```

24         {
25 }

```

5.4.2.12 void* md2cmds_worker (void * *dummy*)

Our worker thread.

Parameters:

dummy > [in] Unused but required by protocol

Definition at line 429 of file md2cmds.c.

```

431         {
432
433     pthread_mutex_lock( &md2cmds_mutex);
434
435     while( 1) {
436         //
437         // wait for someone to give us a command (and tell us they did so)
438         //
439         while( md2cmds_cmd[0] == 0)
440             pthread_cond_wait( &md2cmds_cond, &md2cmds_mutex);
441
442         if( strcmp( md2cmds_cmd, "transfer") == 0) {
443             md2cmds_transfer();
444         } else if( strcmp( md2cmds_cmd, "collect") == 0) {
445             md2cmds_collect();
446         } else if( strcmp( md2cmds_cmd, "rotate") == 0) {
447             md2cmds_rotate();
448         } else if( strcmp( md2cmds_cmd, "center") == 0) {
449             md2cmds_center();
450         } else if( strncmp( md2cmds_cmd, "moveAbs", 7) == 0) {
451             md2cmds_moveAbs( md2cmds_cmd);
452         }
453
454         md2cmds_cmd[0] = 0;
455     }
456 }

```

5.4.3 Variable Documentation

5.4.3.1 char md2cmds_cmd[MD2CMDS_CMD_LENGTH]

our command;

Definition at line 16 of file md2cmds.c.

5.4.3.2 pthread_cond_t md2cmds_cond

condition to signal when it's time to run an md2 command

Definition at line 10 of file md2cmds.c.

5.4.3.3 pthread_mutex_t md2cmds_mutex

mutex for the condition

Definition at line 11 of file md2cmds.c.

5.4.3.4 pthread_cond_t md2cmds_pg_cond

coordinate call and response

Definition at line 13 of file md2cmds.c.

5.4.3.5 pthread_mutex_t md2cmds_pg_mutex

message passing between md2cmds and pg

Definition at line 14 of file md2cmds.c.

5.4.3.6 pthread_t md2cmds_thread [static]

Definition at line 18 of file md2cmds.c.

5.5 pgpmac.c File Reference

Main for the pgpmac project. `#include "pgpmac.h"`

Functions

- void [stdinService](#) (struct pollfd *evt)
Handle keyboard input.
- void [pgpmac_printf](#) (char *fmt,...)
Terminal output routine ala printf.
- int [main](#) (int argc, char **argv)
Our main routine.

Variables

- WINDOW * [term_output](#)
place to print stuff out
- WINDOW * [term_input](#)
place to put the cursor
- WINDOW * [term_status](#)
shutter, lamp, air, etc status
- WINDOW * [term_status2](#)
shutter, lamp, air, etc status
- pthread_mutex_t [ncurses_mutex](#)
allow more than one thread access to the screen
- static struct pollfd [stdinfd](#)
Handle input from the keyboard.

5.5.1 Detailed Description

Main for the pgpmac project.

Date:

2012

Author:

Keith Brister All Rights Reserved

Definition in file [pgpmac.c](#).

5.5.2 Function Documentation

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

Our main routine.

Parameters:

- ← *argc* Number of arguments
- ← *argv* Vector of argument strings

Definition at line 340 of file pgpmac.c.

```

343     {
344     static nfds_t nfds;
345
346     static struct pollfd fda[3], *fdp;    // input for poll: room for postgres, pma
347     c, and stdin
348     static int nfd = 0;                  // number of items in fda
349     static int pollrtn = 0;
350     static struct option long_options[] = {
351         { "i-vars", 0, NULL, 'i'},
352         { "m-vars", 0, NULL, 'm'},
353         { NULL, 0, NULL, 0}
354     };
355     int c;
356     int ivars, mvars;
357     mvars=0;
358     ivars=0;
359     int i;                               // standard loop counter
360
361     while( 1) {
362         c=getopt_long( argc, argv, "im", long_options, NULL);
363         if( c == -1)
364             break;
365
366         switch( c) {
367             case 'i':
368                 ivars=1;
369                 break;
370
371             case 'm':
372                 mvars=1;
373                 break;
374
375         }
376     }
377
378     stdinfda.fd = 0;
379     stdinfda.events = POLLIN;
380
381     initscr();                            // Start ncurses
382     raw();                                // Line buffering disabled, control chars
383     trapped
384     keypad( stdscr, TRUE);                // Why is F1 nifty?
385     refresh();
386
387     pthread_mutex_init( &ncurses_mutex, NULL);    // don't lock this mutex yet beca
388     use we are not multi-threaded until the "_run" functions
389
390     //
391     // Since the modules reference objects in other modules it is important
392     // that everyone is initialized before anyone runs
393     //

```

```
392  lspmac_init( ivars, mvars);
393  lspg_init();
394  lsupdate_init();
395  md2cmds_init();
396
397  term_status = newwin( LS_DISPLAY_WINDOW_HEIGHT, LS_DISPLAY_WINDOW_WIDTH, 3*
LS_DISPLAY_WINDOW_HEIGHT, 0*LS_DISPLAY_WINDOW_WIDTH);
398  box( term_status, 0, 0);
399  wnoutrefresh( term_status);
400
401  term_status2 = newwin( LS_DISPLAY_WINDOW_HEIGHT, LS_DISPLAY_WINDOW_WIDTH, 3*
LS_DISPLAY_WINDOW_HEIGHT, 1*LS_DISPLAY_WINDOW_WIDTH);
402  box( term_status2, 0, 0);
403  wnoutrefresh( term_status2);
404
405  term_output = newwin( 10, 5*LS_DISPLAY_WINDOW_WIDTH, 4*
LS_DISPLAY_WINDOW_HEIGHT, 0);
406  scrollok( term_output, 1);
407  wnoutrefresh( term_output);
408
409  term_input  = newwin( 3, 5*LS_DISPLAY_WINDOW_WIDTH, 10+4*
LS_DISPLAY_WINDOW_HEIGHT, 0);
410  box( term_input, 0, 0);
411  mvwprintw( term_input, 1, 1, "PMAC> ");
412  nodelay( term_input, TRUE);
413  keypad( term_input, TRUE);
414  wnoutrefresh( term_input);
415
416  douupdate();
417
418  lspmac_run();
419  lspg_run();
420  lsupdate_run();
421  md2cmds_run();
422
423  while( 1) {
424      //
425      // Big loop
426      //
427
428      nfd = 0;
429
430      //
431      // keyboard
432      //
433      memcpy( &(fda[nfd++]), &stdinfda, sizeof( struct pollfd));
434
435
436      if( nfd == 0) {
437          //
438          // No connectons yet.  Wait a bit and try again.
439          //
440          sleep( 10);
441          //
442          // go try to connect again
443          //
444          continue;
445      }
446
447
448      pollrtn = poll( fda, nfd, 10);
449
450      for( i=0; pollrtn>0 && i<nfd; i++) {
451          if( fda[i].revents) {
452              pollrtn--;
453              if( fda[i].fd == 0) {
454                  stdinService( &fda[i]);
```



```

455     }
456     }
457 }
458 }
459 }

```

5.5.2.2 void pgpmac_printf (char **fmt*, ...)

Terminal output routine ala printf.

Parameters:

← *fmt* Printf style formatting string

Definition at line 317 of file pgpmac.c.

```

320     {
321     va_list arg_ptr;
322
323     pthread_mutex_lock( &ncurses_mutex);
324
325     va_start( arg_ptr, fmt);
326     vwprintw( term_output, fmt, arg_ptr);
327     va_end( arg_ptr);
328
329     wnoutrefresh( term_output);
330     wnoutrefresh( term_input);
331     doupdate();
332
333     pthread_mutex_unlock( &ncurses_mutex);
334
335 }

```

5.5.2.3 void stdinService (struct pollfd **evt*)

Handle keyboard input.

Parameters:

← *evt* The pollfd object that caused this call

Definition at line 245 of file pgpmac.c.

```

247     {
248     static char cmds[1024];
249     static char cntrlcmd[2];
250     static char cmds_on = 0;
251     int ch;
252
253
254     for( ch=wgetch(term_input); ch != ERR; ch=wgetch(term_input)) {
255         // wprintw( term_output, "%04x\n", ch);
256         // wnoutrefresh( term_output);
257
258         switch( ch) {
259             case KEY_F(1):
260                 endwin();
261                 exit(0);
262                 break;

```

```

263
264     case 0x0001:           // Control-A
265     case 0x0002:           // Control-B
266     case 0x0003:           // Control-C
267     case 0x0004:           // Control-D
268     case 0x0005:           // Control-E
269     case 0x0006:           // Control-F
270     case 0x0007:           // Control-G
271     case 0x000b:           // Control-K
272     case 0x000f:           // Control-O
273     case 0x0010:           // Control-P
274     case 0x0011:           // Control-Q
275     case 0x0012:           // Control-R
276     case 0x0013:           // Control-Q
277     case 0x0016:           // Control-V
278         cntrlcmd[0] = ch;
279         cntrlcmd[1] = 0;
280         lspmac_SockSendline( cntrlcmd);
281         //      PmacSockSendControlCharPrint( ch);
282         break;
283
284     case KEY_BACKSPACE:
285         cmds[cmds_on] = 0;
286         cmds_on == 0 ? 0 : cmds_on--;
287         break;
288
289     case KEY_ENTER:
290     case 0x000a:
291         if( cmds_on > 0 && strlen( cmds) > 0) {
292             lspmac_SockSendline( cmds);
293         }
294         memset( cmds, 0, sizeof(cmds));
295         cmds_on = 0;
296         break;
297
298     default:
299         if( cmds_on < sizeof( cmds)-1) {
300             cmds[cmds_on++] = ch;
301             cmds[cmds_on] = 0;
302         }
303         break;
304     }
305
306     mvwprintw( term_input, 1, 1, "PMAC> %s", cmds);
307     wclrtoeol( term_input);
308     box( term_input, 0, 0);
309     wnoutrefresh( term_input);
310     doupdate();
311
312 }
313 }

```

5.5.3 Variable Documentation

5.5.3.1 pthread_mutex_t ncurses_mutex

allow more than one thread access to the screen

Definition at line 233 of file pgpmac.c.

5.5.3.2 struct pollfd stdinfd [static]

Handle input from the keyboard.

Definition at line 239 of file pgpmac.c.

5.5.3.3 WINDOW* term_input

place to put the cursor

Definition at line 229 of file pgpmac.c.

5.5.3.4 WINDOW* term_output

place to print stuff out

Definition at line 228 of file pgpmac.c.

5.5.3.5 WINDOW* term_status

shutter, lamp, air, etc status

Definition at line 230 of file pgpmac.c.

5.5.3.6 WINDOW* term_status2

shutter, lamp, air, etc status

Definition at line 231 of file pgpmac.c.

5.6 pgpmac.h File Reference

Headers for the entire pgpmac project. `#include <stdio.h>`

`#include <stdlib.h>`

`#include <unistd.h>`

`#include <sys/types.h>`

`#include <sys/socket.h>`

`#include <netdb.h>`

`#include <string.h>`

`#include <netinet/in.h>`

`#include <errno.h>`

`#include <poll.h>`

`#include <libpq-fe.h>`

`#include <ncurses.h>`

`#include <math.h>`

`#include <pthread.h>`

`#include <signal.h>`

`#include <sys/signalfd.h>`

`#include <sys/time.h>`

`#include <time.h>`

`#include <getopt.h>`

Data Structures

- struct [tagEthernetCmd](#)
PMAC ethernet packet definition.
- struct [lspmac_cmd_queue_struct](#)
PMAC command queue item.
- struct [lspmac_motor_struct](#)
Motor information.
- struct [lspg_nextshot_struct](#)
Storage definition for nextshot query.

Defines

- `#define` [LS_DISPLAY_WINDOW_HEIGHT](#) 8
Number of status box rows.

- #define [LS_DISPLAY_WINDOW_WIDTH](#) 24
Number of status box columns.
- #define [LS_PG_QUERY_STRING_LENGTH](#) 1024
Fixed length postgresql query strings. Queries should all be function calls so this is not as weird as one might think.
- #define [MD2CMDS_CMD_LENGTH](#) 32

Typedefs

- typedef struct [tagEthernetCmd](#) [pmac_cmd_t](#)
PMAC ethernet packet definition.
- typedef struct [lspmac_cmd_queue_struct](#) [pmac_cmd_queue_t](#)
PMAC command queue item.
- typedef struct [lspmac_motor_struct](#) [lspmac_motor_t](#)
Motor information.
- typedef struct [lspg_nextshot_struct](#) [lspg_nextshot_t](#)
Storage definition for nextshot query.

Functions

- void [PmacSockSendline](#) (char *s)
- void [lspg_seq_run_prep_all](#) (long long skey, double [kappa](#), double [phi](#), double cx, double cy, double ax, double ay, double az)
Convinence function to call seq run prep.
- void [lspg_zoom_lut_call](#) ()
- void [pgpmac_printf](#) (char *fmt,...)
Terminal output routine ala printf.
- void [lspmac_init](#) (int, int)
Initialize this module.
- void [lspg_init](#) ()
Initallize the lspg module.
- void [lsupdate_init](#) ()
Initialize this module.
- void [md2cmds_init](#) ()
Initialize the md2cmds module.
- void [lspmac_run](#) ()
Start up the lspmac thread.

- void `lspg_run ()`
Start 'er runnin'.
- void `lsupdate_run ()`
run the update routines
- void `md2cmds_run ()`
Start up the thread.

Variables

- `lspg_nextshot_t lspg_nextshot`
the nextshot object
- `lspmac_motor_t lspmac_motors []`
All our motors.
- `lspmac_motor_t * omega`
MD2 omega axis (the air bearing).
- `lspmac_motor_t * alignx`
Alignment stage X.
- `lspmac_motor_t * aligny`
Alignment stage Y.
- `lspmac_motor_t * alignz`
Alignment stage X.
- `lspmac_motor_t * anal`
Polaroid analyzer motor.
- `lspmac_motor_t * zoom`
Optical zoom.
- `lspmac_motor_t * apery`
Aperture Y.
- `lspmac_motor_t * aperz`
Aperture Z.
- `lspmac_motor_t * capy`
Capillary Y.
- `lspmac_motor_t * capz`
Capillary Z.

- [lspmac_motor_t * scinz](#)
Scintillator Z.
- [lspmac_motor_t * cenx](#)
Centering Table X.
- [lspmac_motor_t * ceny](#)
Centering Table Y.
- [lspmac_motor_t * kappa](#)
Kappa.
- [lspmac_motor_t * phi](#)
Phi (not data collection axis).
- [lspmac_motor_t * fshut](#)
Fast shutter.
- [lspmac_motor_t * flight](#)
Front Light DAC.
- [lspmac_motor_t * blight](#)
Back Light DAC.
- [lspmac_motor_t * fscint](#)
Scintillator Piezo DAC.
- [lspmac_motor_t * blight_up](#)
- [int lspmac_nmotors](#)
The number of motors we manage.
- WINDOW * [term_output](#)
place to print stuff out
- WINDOW * [term_input](#)
place to put the cursor
- WINDOW * [term_status](#)
shutter, lamp, air, etc status
- WINDOW * [term_status2](#)
shutter, lamp, air, etc status
- [pthread_mutex_t ncurses_mutex](#)
allow more than one thread access to the screen
- [pthread_cond_t md2cmds_cond](#)
condition to signal when it's time to run an md2 command
- [pthread_mutex_t md2cmds_mutex](#)

mutex for the condition

- pthread_cond_t [md2cmds_pg_cond](#)
coordinate call and response
- pthread_mutex_t [md2cmds_pg_mutex](#)
message passing between md2cmds and pg
- pthread_mutex_t [lspmac_shutter_mutex](#)
Coordinates threads reading shutter status.
- pthread_cond_t [lspmac_shutter_cond](#)
Allows waiting for the shutter status to change.
- int [lspmac_shutter_state](#)
State of the shutter; used to detect changes.
- int [lspmac_shutter_has_opened](#)
Indicates that the shutter had opened, perhaps briefly even if the state did not change.
- pthread_mutex_t [lspmac_moving_mutex](#)
Coordinate moving motors between threads.
- pthread_cond_t [lspmac_moving_cond](#)
Wait for motor(s) to finish moving condition.
- int [lspmac_moving_flags](#)
Flag used to implement motor moving condition.
- pthread_mutex_t [md2_status_mutex](#)
Synchronize reading/writing status buffer.
- char [md2cmds_cmd](#) []
our command;

5.6.1 Detailed Description

Headers for the entire pgpmac project.

Date:

2012

Author:

Keith Brister All Rights Reserved

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

5.6.2 Define Documentation

5.6.2.1 `#define LS_DISPLAY_WINDOW_HEIGHT 8`

Number of status box rows.

Definition at line 29 of file pgpmac.h.

5.6.2.2 `#define LS_DISPLAY_WINDOW_WIDTH 24`

Number of status box columns.

Definition at line 33 of file pgpmac.h.

5.6.2.3 `#define LS_PG_QUERY_STRING_LENGTH 1024`

Fixed length postgresql query strings. Queries should all be function calls so this is not as weird as one might think.

Definition at line 36 of file pgpmac.h.

5.6.2.4 `#define MD2CMDS_CMD_LENGTH 32`

Definition at line 287 of file pgpmac.h.

5.6.3 Typedef Documentation

5.6.3.1 `typedef struct lspg_nextshot_struct lspg_nextshot_t`

Storage definition for nextshot query. The next shot query returns all the information needed to collect the next data frame. Since SQL allows for null fields independently from blank strings a separate integer is used as a flag for this case. This adds to the program complexity but allows for some important cases. Suck it up. definition of the next image to be taken (and the one after that, too!)

5.6.3.2 `typedef struct lspmac_motor_struct lspmac_motor_t`

Motor information. A catchall for motors and motor like objects. Not all members are used by all objects.

5.6.3.3 `typedef struct lspmac_cmd_queue_struct pmac_cmd_queue_t`

PMAC command queue item. Command queue items are fixed length to simplify memory management.

5.6.3.4 `typedef struct tagEthernetCmd pmac_cmd_t`

PMAC ethernet packet definition. Taken directly from the Delta Tau documentation.

5.6.4 Function Documentation

5.6.4.1 void lspg_init ()

Initiallize the lspg module.

Definition at line 1451 of file lspg.c.

```

1451     {
1452     pthread_mutex_init( &pg_queue_mutex, NULL);
1453     lspg_nextshot_init();
1454     lspg_wait_for_detector_init();
1455     lspg_lock_diffractionmeter_init();
1456     lspg_lock_detector_init();
1457 }
```

5.6.4.2 void lspg_run ()

Start 'er runnin'.

Definition at line 1461 of file lspg.c.

```

1461     {
1462     pthread_create( &lspg_thread, NULL, lspg_worker, NULL);
1463 }
```

5.6.4.3 void lspg_seq_run_prep_all (long long *skey*, double *kappa*, double *phi*, double *cx*, double *cy*, double *ax*, double *ay*, double *az*)

Convinence function to call seq run prep.

Parameters:

- ← *skey* px.shots key for this image
- ← *kappa* current kappa postion
- ← *phi* current phi postition
- ← *cx* current center table x
- ← *cy* current center table y
- ← *ax* current alignment table x
- ← *ay* current alignment table y
- ← *az* current alignment table z

Definition at line 839 of file lspg.c.

```

848     {
849     lspg_seq_run_prep_call( skey, kappa, phi, cx, cy, ax, ay, az);
850     lspg_seq_run_prep_wait();
851     lspg_seq_run_prep_done();
852 }
```

5.6.4.4 void lspg_zoom_lut_call ()**5.6.4.5 void lspmac_init (int, int)**

Initialize this module.

Definition at line 1782 of file lspmac.c.

```

1785         {
1786     md2_status_t *p;
1787
1788     // Set our global harvest flags
1789     getivars = ivarsflag;
1790     getmvars = mvarsflag;
1791
1792     // All important status mutex
1793     pthread_mutex_init( &md2_status_mutex, NULL);
1794
1795     //
1796     // Initialize the motor objects
1797     //
1798
1799     p = &md2_status;
1800
1801     omega = lspmac_motor_init( (&lspmac_motors[ 0]), 1, 0, 0, &p->omega_act_pos,
1802                                &p->omega_status_1, &p->omega_status_2, "Omega #1 &1 X", "omega",
1803                                lspmac_moveabs_queue);
1804     alignx = lspmac_motor_init( (&lspmac_motors[ 1]), 2, 0, 1, &p->alignx_act_pos,
1805                                &p->alignx_status_1, &p->alignx_status_2, "Align X #2 &3 X", "align.x",
1806                                lspmac_moveabs_queue);
1807     aligny = lspmac_motor_init( (&lspmac_motors[ 2]), 3, 0, 2, &p->aligny_act_pos,
1808                                &p->aligny_status_1, &p->aligny_status_2, "Align Y #3 &3 Y", "align.y",
1809                                lspmac_moveabs_queue);
1810     alignz = lspmac_motor_init( (&lspmac_motors[ 3]), 4, 0, 3, &p->alignz_act_pos,
1811                                &p->alignz_status_1, &p->alignz_status_2, "Align Z #4 &3 Z", "align.z",
1812                                lspmac_moveabs_queue);
1813     anal = lspmac_motor_init( (&lspmac_motors[ 4]), 5, 0, 4, &p->
1814                                analyzer_act_pos, &p->analyzer_status_1, &p->analyzer_status_2, "Anal #5",
1815                                "lightPolar", lspmac_moveabs_queue);
1816     zoom = lspmac_motor_init( (&lspmac_motors[ 5]), 6, 1, 0, &p->zoom_act_pos,
1817                                &p->zoom_status_1, &p->zoom_status_2, "Zoom #6 &4 Z", "zoom",
1818                                lspmac_movezoom_queue);
1819     apery = lspmac_motor_init( (&lspmac_motors[ 6]), 7, 1, 1, &p->
1820                                aperturey_act_pos, &p->aperturey_status_1, &p->aperturey_status_2, "Aper Y #7 &5
1821                                Y", "appy", lspmac_moveabs_queue);
1822     aperz = lspmac_motor_init( (&lspmac_motors[ 7]), 8, 1, 2, &p->
1823                                aperturez_act_pos, &p->aperturez_status_1, &p->aperturez_status_2, "Aper Z #8 &5
1824                                Z", "appz", lspmac_moveabs_queue);
1825     capy = lspmac_motor_init( (&lspmac_motors[ 8]), 9, 1, 3, &p->capy_act_pos,
1826                                &p->capy_status_1, &p->capy_status_2, "Cap Y #9 &5 U", "capy",
1827                                lspmac_moveabs_queue);
1828     capz = lspmac_motor_init( (&lspmac_motors[ 9]), 10, 1, 4, &p->capz_act_pos,
1829                                &p->capz_status_1, &p->capz_status_2, "Cap Z #10 &5 V", "capz",
1830                                lspmac_moveabs_queue);
1831     scinz = lspmac_motor_init( (&lspmac_motors[10]), 11, 2, 0, &p->scint_act_pos,
1832                                &p->scint_status_1, &p->scint_status_2, "Scin Z #11 &5 W", "scint",
1833                                lspmac_moveabs_queue);
1834     cenx = lspmac_motor_init( (&lspmac_motors[11]), 17, 2, 1, &p->
1835                                centerx_act_pos, &p->centerx_status_1, &p->centerx_status_2, "Cen X #17 &2
1836                                X", "centering.x", lspmac_moveabs_queue);
1837     ceny = lspmac_motor_init( (&lspmac_motors[12]), 18, 2, 2, &p->
1838                                centery_act_pos, &p->centery_status_1, &p->centery_status_2, "Cen Y #18 &2
1839                                Y", "centering.y", lspmac_moveabs_queue);
1840     kappa = lspmac_motor_init( (&lspmac_motors[13]), 19, 2, 3, &p->kappa_act_pos,
1841                                &p->kappa_status_1, &p->kappa_status_2, "Kappa #19 &7 X", "kappa",
1842                                lspmac_moveabs_queue);

```

```

1815  phi    = lspmac_motor_init( &(lspmac_motors[14]), 20, 2, 4, &p->phi_act_pos,
        &p->phi_status_1,      &p->phi_status_2,      "Phi    #20 &7 Y", "phi",
        lspmac_moveabs_queue);
1816
1817  fshut   = lspmac_fshut_init( &(lspmac_motors[15]));
1818  flight  = lspmac_dac_init( &(lspmac_motors[16]), &p->front_dac, 160.0, "M1200"
        , "frontLight.intensity");
1819  blight  = lspmac_dac_init( &(lspmac_motors[17]), &p->back_dac, 160.0, "M1201"
        , "backLight.intensity");
1820  fscint  = lspmac_dac_init( &(lspmac_motors[18]), &p->scint_piezo, 320.0, "M1203"
        , "scint.focus");
1821
1822  blight_ud = lspmac_bio_init( &(lspmac_motors[19]), "backLight", "M1101=%d", &(
        md2_status.acc11c_5), 0x02);
1823
1824
1825
1826
1827  //
1828  // Initialize several commands that get called, perhaps, alot
1829  //
1830  rr_cmd.RequestType = VR_UPLOAD;
1831  rr_cmd.Request     = VR_PMAC_READREADY;
1832  rr_cmd.wValue      = 0;
1833  rr_cmd.wIndex      = 0;
1834  rr_cmd.wLength     = htons(2);
1835  memset( rr_cmd.bData, 0, sizeof(rr_cmd.bData));
1836
1837  gb_cmd.RequestType = VR_UPLOAD;
1838  gb_cmd.Request     = VR_PMAC_GETBUFFER;
1839  gb_cmd.wValue      = 0;
1840  gb_cmd.wIndex      = 0;
1841  gb_cmd.wLength     = htons(1400);
1842  memset( gb_cmd.bData, 0, sizeof(gb_cmd.bData));
1843
1844  cr_cmd.RequestType = VR_UPLOAD;
1845  cr_cmd.Request     = VR_CTRL_RESPONSE;
1846  cr_cmd.wValue      = 0;
1847  cr_cmd.wIndex      = 0;
1848  cr_cmd.wLength     = htons(1400);
1849  memset( cr_cmd.bData, 0, sizeof(cr_cmd.bData));
1850
1851  //
1852  // Initialize some mutexs and conditions
1853  //
1854
1855  pthread_mutex_init( &pmac_queue_mutex, NULL);
1856  pthread_cond_init( &pmac_queue_cond, NULL);
1857
1858  lspmac_shutter_state = 0; // assume the shutter is
        now closed: not a big deal if we are wrong
1859  pthread_mutex_init( &lspmac_shutter_mutex, NULL);
1860  pthread_cond_init( &lspmac_shutter_cond, NULL);
1861  pmacfd.fd = -1;
1862
1863  pthread_mutex_init( &lspmac_moving_mutex, NULL);
1864  pthread_cond_init( &lspmac_moving_cond, NULL);
1865
1866 }

```

5.6.4.6 void lspmac_run ()

Start up the lspmac thread.

Definition at line 1870 of file lspmac.c.

```
1870     {
1871     pthread_create( &pmac_thread, NULL, lspmac_worker, NULL);
1872 }
```

5.6.4.7 void lsupdate_init ()

Initialize this module.

Definition at line 89 of file lsupdate.c.

```
89     {
90 }
```

5.6.4.8 void lsupdate_run ()

run the update routines

Definition at line 94 of file lsupdate.c.

```
94     {
95     // pthread_create( &lsupdate_thread, NULL, lsupdate_worker, NULL);
96 }
```

5.6.4.9 void md2cmds_init ()

Initialize the md2cmds module.

Definition at line 461 of file md2cmds.c.

```
461     {
462     memset( md2cmds_cmd, 0, sizeof( md2cmds_cmd));
463
464     pthread_mutex_init( &md2cmds_mutex, NULL);
465     pthread_cond_init( &md2cmds_cond, NULL);
466
467     pthread_mutex_init( &md2cmds_pg_mutex, NULL);
468     pthread_cond_init( &md2cmds_pg_cond, NULL);
469
470 }
```

5.6.4.10 void md2cmds_run ()

Start up the thread.

Definition at line 474 of file md2cmds.c.

```
474     {
475     pthread_create( &md2cmds_thread, NULL, md2cmds_worker, NULL);
476 }
```

5.6.4.11 void pgpmac_printf(char *fmt, ...)

Terminal output routine ala printf.

Parameters:

← *fmt* Printf style formatting string

Definition at line 317 of file pgpmac.c.

```
320     {
321     va_list arg_ptr;
322
323     pthread_mutex_lock( &ncurses_mutex);
324
325     va_start( arg_ptr, fmt);
326     vwprintw( term_output, fmt, arg_ptr);
327     va_end( arg_ptr);
328
329     wnoutrefresh( term_output);
330     wnoutrefresh( term_input);
331     doupdate();
332
333     pthread_mutex_unlock( &ncurses_mutex);
334
335 }
```

5.6.4.12 void PmacSockSendline(char *s)

5.6.5 Variable Documentation

5.6.5.1 lspmac_motor_t* alignx

Alignment stage X.

Definition at line 74 of file lspmac.c.

5.6.5.2 lspmac_motor_t* aligny

Alignment stage Y.

Definition at line 75 of file lspmac.c.

5.6.5.3 lspmac_motor_t* alignz

Alignment stage X.

Definition at line 76 of file lspmac.c.

5.6.5.4 lspmac_motor_t* anal

Polaroid analyzer motor.

Definition at line 77 of file lspmac.c.

5.6.5.5 lspmac_motor_t* apery

Aperture Y.

Definition at line 79 of file lspmac.c.

5.6.5.6 lspmac_motor_t* aperz

Aperture Z.

Definition at line 80 of file lspmac.c.

5.6.5.7 lspmac_motor_t* blight

Back Light DAC.

Definition at line 91 of file lspmac.c.

5.6.5.8 lspmac_motor_t* blight_up**5.6.5.9 lspmac_motor_t* capy**

Capillary Y.

Definition at line 81 of file lspmac.c.

5.6.5.10 lspmac_motor_t* capz

Capillary Z.

Definition at line 82 of file lspmac.c.

5.6.5.11 lspmac_motor_t* cenx

Centering Table X.

Definition at line 84 of file lspmac.c.

5.6.5.12 lspmac_motor_t* ceny

Centering Table Y.

Definition at line 85 of file lspmac.c.

5.6.5.13 lspmac_motor_t* flight

Front Light DAC.

Definition at line 90 of file lspmac.c.

5.6.5.14 `lspmac_motor_t* fscint`

Scintillator Piezo DAC.

Definition at line 92 of file `lspmac.c`.

5.6.5.15 `lspmac_motor_t* fshut`

Fast shutter.

Definition at line 89 of file `lspmac.c`.

5.6.5.16 `lspmac_motor_t* kappa`

Kappa.

Definition at line 86 of file `lspmac.c`.

5.6.5.17 `lspg_nextshot_t lspg_nextshot`

the nextshot object

Definition at line 73 of file `lspg.c`.

5.6.5.18 `lspmac_motor_t lspmac_motors[]`

All our motors.

Definition at line 71 of file `lspmac.c`.

5.6.5.19 `pthread_cond_t lspmac_moving_cond`

Wait for motor(s) to finish moving condition.

Definition at line 59 of file `lspmac.c`.

5.6.5.20 `int lspmac_moving_flags`

Flag used to implement motor moving condition.

Definition at line 60 of file `lspmac.c`.

5.6.5.21 `pthread_mutex_t lspmac_moving_mutex`

Coordinate moving motors between threads.

Definition at line 58 of file `lspmac.c`.

5.6.5.22 `int lspmac_nmotors`

The number of motors we manage.

Definition at line 72 of file `lspmac.c`.

5.6.5.23 pthread_cond_t lspmac_shutter_cond

Allows waiting for the shutter status to change.

Definition at line 57 of file lspmac.c.

5.6.5.24 int lspmac_shutter_has_opened

Indicates that the shutter had opened, perhaps briefly even if the state did not change.

Definition at line 55 of file lspmac.c.

5.6.5.25 pthread_mutex_t lspmac_shutter_mutex

Coordinates threads reading shutter status.

Definition at line 56 of file lspmac.c.

5.6.5.26 int lspmac_shutter_state

State of the shutter, used to detect changes.

Definition at line 54 of file lspmac.c.

5.6.5.27 pthread_mutex_t md2_status_mutex

Synchronize reading/writing status buffer.

Definition at line 277 of file lspmac.c.

5.6.5.28 char md2cmds_cmd[]

our command;

Definition at line 16 of file md2cmds.c.

5.6.5.29 pthread_cond_t md2cmds_cond

condition to signal when it's time to run an md2 command

Definition at line 10 of file md2cmds.c.

5.6.5.30 pthread_mutex_t md2cmds_mutex

mutex for the condition

Definition at line 11 of file md2cmds.c.

5.6.5.31 pthread_cond_t md2cmds_pg_cond

coordinate call and response

Definition at line 13 of file md2cmds.c.

5.6.5.32 pthread_mutex_t md2cmds_pg_mutex

message passing between md2cmds and pg

Definition at line 14 of file md2cmds.c.

5.6.5.33 pthread_mutex_t ncurses_mutex

allow more than one thread access to the screen

Definition at line 233 of file pgpmac.c.

5.6.5.34 lspmac_motor_t* omega

MD2 omega axis (the air bearing).

Definition at line 73 of file lspmac.c.

5.6.5.35 lspmac_motor_t* phi

Phi (not data collection axis).

Definition at line 87 of file lspmac.c.

5.6.5.36 lspmac_motor_t* scinz

Scintillator Z.

Definition at line 83 of file lspmac.c.

5.6.5.37 WINDOW* term_input

place to put the cursor

Definition at line 229 of file pgpmac.c.

5.6.5.38 WINDOW* term_output

place to print stuff out

Definition at line 228 of file pgpmac.c.

5.6.5.39 WINDOW* term_status

shutter, lamp, air, etc status

Definition at line 230 of file pgpmac.c.

5.6.5.40 WINDOW* term_status2

shutter, lamp, air, etc status

Definition at line 231 of file pgpmac.c.

5.6.5.41 lspmac_motor_t* zoom

Optical zoom.

Definition at line 78 of file lspmac.c.

Index

acc11c_1
 md2StatusStruct, 36

acc11c_2
 md2StatusStruct, 36

acc11c_3
 md2StatusStruct, 36

acc11c_5
 md2StatusStruct, 36

acc11c_6
 md2StatusStruct, 37

active
 lspg_nextshot_struct, 14

active2
 lspg_nextshot_struct, 14

active2_isnull
 lspg_nextshot_struct, 14

active_isnull
 lspg_nextshot_struct, 15

actual_pos_cnts_p
 lspmac_motor_struct, 31

alignx
 lspmac.c, 125
 pgpmac.h, 162

alignx_act_pos
 md2StatusStruct, 37

alignx_status_1
 md2StatusStruct, 37

alignx_status_2
 md2StatusStruct, 37

aligny
 lspmac.c, 125
 pgpmac.h, 162

aligny_act_pos
 md2StatusStruct, 37

aligny_status_1
 md2StatusStruct, 37

aligny_status_2
 md2StatusStruct, 37

alignz
 lspmac.c, 125
 pgpmac.h, 162

alignz_act_pos
 md2StatusStruct, 37

alignz_status_1
 md2StatusStruct, 37

alignz_status_2
 md2StatusStruct, 37

anal
 lspmac.c, 125
 pgpmac.h, 162

analyzer_act_pos
 md2StatusStruct, 37

analyzer_status_1
 md2StatusStruct, 38

analyzer_status_2
 md2StatusStruct, 38

aperturey_act_pos
 md2StatusStruct, 38

aperturey_status_1
 md2StatusStruct, 38

aperturey_status_2
 md2StatusStruct, 38

aperturez_act_pos
 md2StatusStruct, 38

aperturez_status_1
 md2StatusStruct, 38

aperturez_status_2
 md2StatusStruct, 38

apery
 lspmac.c, 125
 pgpmac.h, 162

aperz
 lspmac.c, 126
 pgpmac.h, 163

ax
 lspg_nextshot_struct, 15

ax2
 lspg_nextshot_struct, 15

ax2_isnull
 lspg_nextshot_struct, 15

ax_isnull
 lspg_nextshot_struct, 15

ay
 lspg_nextshot_struct, 15

ay2
 lspg_nextshot_struct, 15

ay2_isnull
 lspg_nextshot_struct, 15

ay_isnull
 lspg_nextshot_struct, 15

- az
 - lspg_nextshot_struct, 15
- az2
 - lspg_nextshot_struct, 16
- az2_isnull
 - lspg_nextshot_struct, 16
- az_isnull
 - lspg_nextshot_struct, 16
- back_dac
 - md2StatusStruct, 38
- bData
 - tagEthernetCmd, 44
- blight
 - lspmac.c, 126
 - pgpmac.h, 163
- blight_ud
 - lspmac.c, 126
- blight_up
 - pgpmac.h, 163
- capy
 - lspmac.c, 126
 - pgpmac.h, 163
- capy_act_pos
 - md2StatusStruct, 38
- capy_status_1
 - md2StatusStruct, 38
- capy_status_2
 - md2StatusStruct, 39
- capz
 - lspmac.c, 126
 - pgpmac.h, 163
- capz_act_pos
 - md2StatusStruct, 39
- capz_status_1
 - md2StatusStruct, 39
- capz_status_2
 - md2StatusStruct, 39
- centerx_act_pos
 - md2StatusStruct, 39
- centerx_status_1
 - md2StatusStruct, 39
- centerx_status_2
 - md2StatusStruct, 39
- centery_act_pos
 - md2StatusStruct, 39
- centery_status_1
 - md2StatusStruct, 39
- centery_status_2
 - md2StatusStruct, 39
- cenx
 - lspmac.c, 126
 - pgpmac.h, 163
- ceny
 - lspmac.c, 126
 - pgpmac.h, 163
- cleanstr
 - lspmac.c, 94
- cond
 - lspg_lock_detector_struct, 9
 - lspg_lock_diffractionmeter_struct, 10
 - lspg_nextshot_struct, 16
 - lspg_seq_run_prep_struct, 24
 - lspg_wait_for_detector_struct, 25
 - lspmac_motor_struct, 31
- cr_cmd
 - lspmac.c, 126
- cx
 - lspg_nextshot_struct, 16
- cx2
 - lspg_nextshot_struct, 16
- cx2_isnull
 - lspg_nextshot_struct, 16
- cx_isnull
 - lspg_nextshot_struct, 16
- cy
 - lspg_nextshot_struct, 16
- cy2
 - lspg_nextshot_struct, 16
- cy2_isnull
 - lspg_nextshot_struct, 17
- cy_isnull
 - lspg_nextshot_struct, 17
- dac_mvar
 - lspmac_motor_struct, 31
- dbmem
 - lspmac.c, 126
- dbmemIn
 - lspmac.c, 127
- dsdir
 - lspg_nextshot_struct, 17
- dsdir_isnull
 - lspg_nextshot_struct, 17
- dsdist
 - lspg_nextshot_struct, 17
- dsdist2
 - lspg_nextshot_struct, 17
- dsdist2_isnull
 - lspg_nextshot_struct, 17
- dsdist_isnull
 - lspg_nextshot_struct, 17
- dsexp
 - lspg_nextshot_struct, 17
- dsexp2
 - lspg_nextshot_struct, 17
- dsexp2_isnull

- lspg_nextshot_struct, 18
- dsexp_isnull
 - lspg_nextshot_struct, 18
- dshpid
 - lspg_nextshot_struct, 18
- dshpid_isnull
 - lspg_nextshot_struct, 18
- dskappa
 - lspg_nextshot_struct, 18
- dskappa2
 - lspg_nextshot_struct, 18
- dskappa2_isnull
 - lspg_nextshot_struct, 18
- dskappa_isnull
 - lspg_nextshot_struct, 18
- dsnrg
 - lspg_nextshot_struct, 18
- dsnrg2
 - lspg_nextshot_struct, 18
- dsnrg2_isnull
 - lspg_nextshot_struct, 19
- dsnrg_isnull
 - lspg_nextshot_struct, 19
- dsomega
 - lspg_nextshot_struct, 19
- dsomega2
 - lspg_nextshot_struct, 19
- dsomega2_isnull
 - lspg_nextshot_struct, 19
- dsomega_isnull
 - lspg_nextshot_struct, 19
- dsoscaxis
 - lspg_nextshot_struct, 19
- dsoscaxis2
 - lspg_nextshot_struct, 19
- dsoscaxis2_isnull
 - lspg_nextshot_struct, 19
- dsoscaxis_isnull
 - lspg_nextshot_struct, 19
- dsowidth
 - lspg_nextshot_struct, 20
- dsowidth2
 - lspg_nextshot_struct, 20
- dsowidth2_isnull
 - lspg_nextshot_struct, 20
- dsowidth_isnull
 - lspg_nextshot_struct, 20
- dsphi
 - lspg_nextshot_struct, 20
- dsphi2
 - lspg_nextshot_struct, 20
- dsphi2_isnull
 - lspg_nextshot_struct, 20
- dsphi_isnull
 - lspg_nextshot_struct, 20
- dspid
 - lspg_nextshot_struct, 20
- dspid_isnull
 - lspg_nextshot_struct, 20
- dummy1
 - md2StatusStruct, 39
- dummy2
 - md2StatusStruct, 40
- dummy3
 - md2StatusStruct, 40
- dummy4
 - md2StatusStruct, 40
- dummy5
 - md2StatusStruct, 40
- dummy6
 - md2StatusStruct, 40
- dummy7
 - md2StatusStruct, 40
- dummy8
 - md2StatusStruct, 40
- dummy9
 - md2StatusStruct, 40
- dummyA
 - md2StatusStruct, 40
- dummyB
 - md2StatusStruct, 40
- ethCmdOff
 - lspmac.c, 127
- ethCmdOn
 - lspmac.c, 127
- ethCmdQueue
 - lspmac.c, 127
- ethCmdReply
 - lspmac.c, 127
- flight
 - lspmac.c, 127
 - pgpmac.h, 163
- format
 - lspmac_motor_struct, 31
- front_dac
 - md2StatusStruct, 40
- fs_has_opened
 - md2StatusStruct, 41
- fs_has_opened_globally
 - md2StatusStruct, 41
- fs_is_open
 - md2StatusStruct, 41
- fscint
 - lspmac.c, 127
 - pgpmac.h, 163
- fshut

- lspmac.c, [127](#)
- pgpmac.h, [164](#)
- gb_cmd
 - lspmac.c, [127](#)
- getivars
 - lspmac.c, [128](#)
- getmvars
 - lspmac.c, [128](#)
- hex_dump
 - lspmac.c, [95](#)
- if
 - lspmac.c, [95](#), [96](#)
- kappa
 - lspmac.c, [128](#)
 - pgpmac.h, [164](#)
- kappa_act_pos
 - md2StatusStruct, [41](#)
- kappa_status_1
 - md2StatusStruct, [41](#)
- kappa_status_2
 - md2StatusStruct, [41](#)
- linesReceived
 - lspmac.c, [128](#)
- logtime
 - md2cmds.c, [136](#)
- LS_DISPLAY_WINDOW_HEIGHT
 - pgpmac.h, [157](#)
- LS_DISPLAY_WINDOW_WIDTH
 - pgpmac.h, [157](#)
- LS_PG_QUERY_QUEUE_LENGTH
 - lspg.c, [53](#)
- LS_PG_QUERY_STRING_LENGTH
 - pgpmac.h, [157](#)
- ls_pg_state
 - lspg.c, [80](#)
- LS_PG_STATE_IDLE
 - lspg.c, [53](#)
- LS_PG_STATE_INIT
 - lspg.c, [53](#)
- LS_PG_STATE_INIT_POLL
 - lspg.c, [53](#)
- LS_PG_STATE_RECV
 - lspg.c, [53](#)
- LS_PG_STATE_RESET
 - lspg.c, [53](#)
- LS_PG_STATE_RESET_POLL
 - lspg.c, [53](#)
- LS_PG_STATE_SEND
 - lspg.c, [53](#)
- LS_PG_STATE_SEND_FLUSH
 - lspg.c, [53](#)
- lspg.c, [53](#)
- ls_pmac_state
 - lspmac.c, [128](#)
- LS_PMAC_STATE_CR
 - lspmac.c, [91](#)
- LS_PMAC_STATE_DETACHED
 - lspmac.c, [91](#)
- LS_PMAC_STATE_GB
 - lspmac.c, [91](#)
- LS_PMAC_STATE_GMR
 - lspmac.c, [91](#)
- LS_PMAC_STATE_IDLE
 - lspmac.c, [91](#)
- LS_PMAC_STATE_RESET
 - lspmac.c, [91](#)
- LS_PMAC_STATE_RR
 - lspmac.c, [91](#)
- LS_PMAC_STATE_SC
 - lspmac.c, [91](#)
- LS_PMAC_STATE_WACK
 - lspmac.c, [91](#)
- LS_PMAC_STATE_WACK_CC
 - lspmac.c, [91](#)
- LS_PMAC_STATE_WACK_NFR
 - lspmac.c, [91](#)
- LS_PMAC_STATE_WACK_RR
 - lspmac.c, [92](#)
- LS_PMAC_STATE_WCR
 - lspmac.c, [92](#)
- LS_PMAC_STATE_WGB
 - lspmac.c, [92](#)
- lsConnect
 - lspmac.c, [96](#)
- lspg.c, [47](#)
 - LS_PG_QUERY_QUEUE_LENGTH, [53](#)
 - ls_pg_state, [80](#)
 - LS_PG_STATE_IDLE, [53](#)
 - LS_PG_STATE_INIT, [53](#)
 - LS_PG_STATE_INIT_POLL, [53](#)
 - LS_PG_STATE_RECV, [53](#)
 - LS_PG_STATE_RESET, [53](#)
 - LS_PG_STATE_RESET_POLL, [53](#)
 - LS_PG_STATE_SEND, [53](#)
 - LS_PG_STATE_SEND_FLUSH, [53](#)
 - lspg_blight_lut_cb, [54](#)
 - lspg_cmd_cb, [54](#)
 - lspg_connectPoll_response, [80](#)
 - lspg_flight_lut_cb, [55](#)
 - lspg_flush, [56](#)
 - lspg_getcenter_cb, [56](#)
 - lspg_init, [56](#)
 - lspg_init_motors_cb, [57](#)
 - lspg_lock_detector, [80](#)
 - lspg_lock_detector_all, [58](#)

- lspg_lock_detector_call, 58
- lspg_lock_detector_cb, 58
- lspg_lock_detector_done, 58
- lspg_lock_detector_init, 59
- lspg_lock_detector_t, 53
- lspg_lock_detector_wait, 59
- lspg_lock_diffractionmeter, 80
- lspg_lock_diffractionmeter_all, 59
- lspg_lock_diffractionmeter_call, 59
- lspg_lock_diffractionmeter_cb, 59
- lspg_lock_diffractionmeter_done, 60
- lspg_lock_diffractionmeter_init, 60
- lspg_lock_diffractionmeter_t, 53
- lspg_lock_diffractionmeter_wait, 60
- lspg_next_state, 60
- lspg_nextaction_cb, 61
- lspg_nextshot, 80
- lspg_nextshot_call, 62
- lspg_nextshot_cb, 62
- lspg_nextshot_done, 66
- lspg_nextshot_init, 66
- lspg_nextshot_wait, 66
- lspg_pg_connect, 67
- lspg_pg_service, 68
- lspg_query_next, 70
- lspg_query_push, 70
- lspg_query_queue, 80
- lspg_query_queue_off, 81
- lspg_query_queue_on, 81
- lspg_query_queue_reply, 81
- lspg_query_queue_t, 54
- lspg_query_reply_next, 71
- lspg_query_reply_peek, 71
- lspg_receive, 72
- lspg_resetPoll_response, 81
- lspg_run, 73
- lspg_send_next_query, 73
- lspg_seq_run_prep, 81
- lspg_seq_run_prep_all, 74
- lspg_seq_run_prep_call, 75
- lspg_seq_run_prep_cb, 75
- lspg_seq_run_prep_done, 75
- lspg_seq_run_prep_init, 76
- lspg_seq_run_prep_t, 54
- lspg_seq_run_prep_wait, 76
- lspg_sig_service, 76
- lspg_thread, 81
- lspg_wait_for_detector, 81
- lspg_wait_for_detector_all, 77
- lspg_wait_for_detector_call, 77
- lspg_wait_for_detector_cb, 77
- lspg_wait_for_detector_done, 77
- lspg_wait_for_detector_init, 77
- lspg_wait_for_detector_t, 54
- lspg_wait_for_detector_wait, 78
- lspg_worker, 78
- lspg_zoom_lut_cb, 79
- lspgfd, 81
- pg_queue_mutex, 82
- q, 82
- lspg_blight_lut_cb
 - lspg.c, 54
- lspg_cmd_cb
 - lspg.c, 54
- lspg_connectPoll_response
 - lspg.c, 80
- lspg_flight_lut_cb
 - lspg.c, 55
- lspg_flush
 - lspg.c, 56
- lspg_getcenter_cb
 - lspg.c, 56
- lspg_init
 - lspg.c, 56
 - pgpmac.h, 158
- lspg_init_motors_cb
 - lspg.c, 57
- lspg_lock_detector
 - lspg.c, 80
- lspg_lock_detector_all
 - lspg.c, 58
- lspg_lock_detector_call
 - lspg.c, 58
- lspg_lock_detector_cb
 - lspg.c, 58
- lspg_lock_detector_done
 - lspg.c, 58
- lspg_lock_detector_init
 - lspg.c, 59
- lspg_lock_detector_struct, 9
 - cond, 9
 - mutex, 9
 - new_value_ready, 9
- lspg_lock_detector_t
 - lspg.c, 53
- lspg_lock_detector_wait
 - lspg.c, 59
- lspg_lock_diffractionmeter
 - lspg.c, 80
- lspg_lock_diffractionmeter_all
 - lspg.c, 59
- lspg_lock_diffractionmeter_call
 - lspg.c, 59
- lspg_lock_diffractionmeter_cb
 - lspg.c, 59
- lspg_lock_diffractionmeter_done
 - lspg.c, 60
- lspg_lock_diffractionmeter_init

lspg.c, 60
lspg_lock_diffractionmeter_struct, 10
cond, 10
mutex, 10
new_value_ready, 10
lspg_lock_diffractionmeter_t
lspg.c, 53
lspg_lock_diffractionmeter_wait
lspg.c, 60
lspg_next_state
lspg.c, 60
lspg_nextaction_cb
lspg.c, 61
lspg_nextshot
lspg.c, 80
pgpmac.h, 164
lspg_nextshot_call
lspg.c, 62
lspg_nextshot_cb
lspg.c, 62
lspg_nextshot_done
lspg.c, 66
lspg_nextshot_init
lspg.c, 66
lspg_nextshot_struct, 11
active, 14
active2, 14
active2_isnull, 14
active_isnull, 15
ax, 15
ax2, 15
ax2_isnull, 15
ax_isnull, 15
ay, 15
ay2, 15
ay2_isnull, 15
ay_isnull, 15
az, 15
az2, 16
az2_isnull, 16
az_isnull, 16
cond, 16
cx, 16
cx2, 16
cx2_isnull, 16
cx_isnull, 16
cy, 16
cy2, 16
cy2_isnull, 17
cy_isnull, 17
dsdir, 17
dsdir_isnull, 17
dsdist, 17
dsdist2, 17
dsdist2_isnull, 17
dsdist_isnull, 17
dsexp, 17
dsexp2, 17
dsexp2_isnull, 18
dsexp_isnull, 18
dshpid, 18
dshpid_isnull, 18
dskappa, 18
dskappa2, 18
dskappa2_isnull, 18
dskappa_isnull, 18
dsnrg, 18
dsnrg2, 18
dsnrg2_isnull, 19
dsnrg_isnull, 19
dsomega, 19
dsomega2, 19
dsomega2_isnull, 19
dsomega_isnull, 19
dsoscaxis, 19
dsoscaxis2, 19
dsoscaxis2_isnull, 19
dsoscaxis_isnull, 19
dsowidth, 20
dsowidth2, 20
dsowidth2_isnull, 20
dsowidth_isnull, 20
dsphi, 20
dsphi2, 20
dsphi2_isnull, 20
dsphi_isnull, 20
dspid, 20
dspid_isnull, 20
mutex, 21
new_value_ready, 21
no_rows_returned, 21
sfn, 21
sfn_isnull, 21
sindex, 21
sindex2, 21
sindex2_isnull, 21
sindex_isnull, 21
skey, 21
skey_isnull, 22
sstart, 22
sstart2, 22
sstart2_isnull, 22
sstart_isnull, 22
stype, 22
stype2, 22
stype2_isnull, 22
stype_isnull, 22
lspg_nextshot_t

- pgpmac.h, 157
- lspg_nextshot_wait
 - lspg.c, 66
- lspg_pg_connect
 - lspg.c, 67
- lspg_pg_service
 - lspg.c, 68
- lspg_query_next
 - lspg.c, 70
- lspg_query_push
 - lspg.c, 70
- lspg_query_queue
 - lspg.c, 80
- lspg_query_queue_off
 - lspg.c, 81
- lspg_query_queue_on
 - lspg.c, 81
- lspg_query_queue_reply
 - lspg.c, 81
- lspg_query_queue_t
 - lspg.c, 54
- lspg_query_reply_next
 - lspg.c, 71
- lspg_query_reply_peek
 - lspg.c, 71
- lspg_receive
 - lspg.c, 72
- lspg_resetPoll_response
 - lspg.c, 81
- lspg_run
 - lspg.c, 73
 - pgpmac.h, 158
- lspg_send_next_query
 - lspg.c, 73
- lspg_seq_run_prep
 - lspg.c, 81
- lspg_seq_run_prep_all
 - lspg.c, 74
 - pgpmac.h, 158
- lspg_seq_run_prep_call
 - lspg.c, 75
- lspg_seq_run_prep_cb
 - lspg.c, 75
- lspg_seq_run_prep_done
 - lspg.c, 75
- lspg_seq_run_prep_init
 - lspg.c, 76
- lspg_seq_run_prep_struct, 24
 - cond, 24
 - mutex, 24
 - new_value_ready, 24
- lspg_seq_run_prep_t
 - lspg.c, 54
- lspg_seq_run_prep_wait
 - lspg.c, 76
- lspg_sig_service
 - lspg.c, 76
- lspg_thread
 - lspg.c, 81
- lspg_wait_for_detector
 - lspg.c, 81
- lspg_wait_for_detector_all
 - lspg.c, 77
- lspg_wait_for_detector_call
 - lspg.c, 77
- lspg_wait_for_detector_cb
 - lspg.c, 77
- lspg_wait_for_detector_done
 - lspg.c, 77
- lspg_wait_for_detector_init
 - lspg.c, 77
- lspg_wait_for_detector_struct, 25
 - cond, 25
 - mutex, 25
 - new_value_ready, 25
- lspg_wait_for_detector_t
 - lspg.c, 54
- lspg_wait_for_detector_wait
 - lspg.c, 78
- lspg_worker
 - lspg.c, 78
- lspg_zoom_lut_call
 - pgpmac.h, 158
- lspg_zoom_lut_cb
 - lspg.c, 79
- lspgfd
 - lspg.c, 81
- lspgQueryQueueStruct, 26
 - onResponse, 26
 - qs, 26
- lspmac.c, 83
 - alignx, 125
 - aligny, 125
 - alignz, 125
 - anal, 125
 - apery, 125
 - aperz, 126
 - blight, 126
 - blight_ud, 126
 - capy, 126
 - capz, 126
 - cenx, 126
 - ceny, 126
 - cleanstr, 94
 - cr_cmd, 126
 - dbmem, 126
 - dbmemIn, 127
 - ethCmdOff, 127

ethCmdOn, 127
ethCmdQueue, 127
ethCmdReply, 127
flight, 127
fscint, 127
fshut, 127
gb_cmd, 127
getivars, 128
getmvars, 128
hex_dump, 95
if, 95, 96
kappa, 128
linesReceived, 128
ls_pmac_state, 128
LS_PMAC_STATE_CR, 91
LS_PMAC_STATE_DETACHED, 91
LS_PMAC_STATE_GB, 91
LS_PMAC_STATE_GMR, 91
LS_PMAC_STATE_IDLE, 91
LS_PMAC_STATE_RESET, 91
LS_PMAC_STATE_RR, 91
LS_PMAC_STATE_SC, 91
LS_PMAC_STATE_WACK, 91
LS_PMAC_STATE_WACK_CC, 91
LS_PMAC_STATE_WACK_NFR, 91
LS_PMAC_STATE_WACK_RR, 92
LS_PMAC_STATE_WCR, 92
LS_PMAC_STATE_WGB, 92
lsConnect, 96
lspmac_bio_init, 97
lspmac_bio_read, 98
lspmac_dac_init, 98
lspmac_dac_read, 99
lspmac_Error, 99
lspmac_fshut_init, 100
lspmac_get_status, 100
lspmac_get_status_cb, 100
lspmac_GetAllIVars, 102
lspmac_GetAllIVarsCB, 103
lspmac_GetAllMVars, 103
lspmac_GetAllMVarsCB, 103
lspmac_Getmem, 104
lspmac_GetmemReplyCB, 104
lspmac_GetShortReplyCB, 104
lspmac_init, 105
lspmac_lut, 107
lspmac_motor_init, 108
lspmac_motors, 128
lspmac_moveabs_bio_queue, 109
lspmac_moveabs_fshut_queue, 109
lspmac_moveabs_queue, 110
lspmac_moveabs_wait, 110
lspmac_movedac_queue, 112
lspmac_movezoom_queue, 112
lspmac_moving_cond, 128
lspmac_moving_flags, 128
lspmac_moving_mutex, 128
lspmac_next_state, 113
lspmac_nmotors, 129
lspmac_pmacmotor_read, 115
lspmac_pop_queue, 116
lspmac_pop_reply, 116
lspmac_push_queue, 117
lspmac_Reset, 117
lspmac_run, 117
lspmac_send_command, 117
lspmac_sendcmd_nocb, 119
lspmac_SendControlReplyPrintCB, 119
lspmac_Service, 119
lspmac_shutter_cond, 129
lspmac_shutter_has_opened, 129
lspmac_shutter_mutex, 129
lspmac_shutter_read, 123
lspmac_shutter_state, 129
lspmac_SockFlush, 123
lspmac_SockGetmem, 123
lspmac_SockSendControlCharPrint, 123
lspmac_SockSendline, 123
lspmac_SockSendline_nr, 124
lspmac_worker, 124
md2_status, 129
md2_status_mutex, 129
md2_status_t, 94
now, 129
omega, 129
phi, 130
PMAC_CMD_QUEUE_LENGTH, 92
pmac_cmd_size, 92
pmac_error_strs, 130
PMAC_MIN_CMD_TIME, 92
pmac_queue_cond, 130
pmac_queue_mutex, 130
pmac_thread, 130
pmacfd, 131
PMACPORT, 92
rr_cmd, 131
scinz, 131
VR_CTRL_RESPONSE, 92
VR_DOWNLOAD, 92
VR_FWDOWNLOAD, 92
VR_IPADDRESS, 93
VR_PMAC_FLUSH, 93
VR_PMAC_GETBUFFER, 93
VR_PMAC_GETLINE, 93
VR_PMAC_GETMEM, 93
VR_PMAC_GETRESPONSE, 93
VR_PMAC_PORT, 93
VR_PMAC_READREADY, 93

- VR_PMAC_SENDCTRLCHAR, 93
- VR_PMAC_SENDLINE, 93
- VR_PMAC_SETBIT, 93
- VR_PMAC_SETBITS, 94
- VR_PMAC_SETMEM, 94
- VR_PMAC_WRITEBUFFER, 94
- VR_PMAC_WRITEERROR, 94
- VR_UPLOAD, 94
- zoom, 131
- lspmac_bio_init
 - lspmac.c, 97
- lspmac_bio_read
 - lspmac.c, 98
- lspmac_cmd_queue_struct, 27
 - no_reply, 27
 - onResponse, 27
 - pcmd, 27
 - rbuff, 27
 - time_sent, 28
- lspmac_dac_init
 - lspmac.c, 98
- lspmac_dac_read
 - lspmac.c, 99
- lspmac_Error
 - lspmac.c, 99
- lspmac_fshut_init
 - lspmac.c, 100
- lspmac_get_status
 - lspmac.c, 100
- lspmac_get_status_cb
 - lspmac.c, 100
- lspmac_GetAllIVars
 - lspmac.c, 102
- lspmac_GetAllIVarsCB
 - lspmac.c, 103
- lspmac_GetAllMVars
 - lspmac.c, 103
- lspmac_GetAllMVarsCB
 - lspmac.c, 103
- lspmac_Getmem
 - lspmac.c, 104
- lspmac_GetmemReplyCB
 - lspmac.c, 104
- lspmac_GetShortReplyCB
 - lspmac.c, 104
- lspmac_init
 - lspmac.c, 105
 - pgpmac.h, 159
- lspmac_lut
 - lspmac.c, 107
- lspmac_motor_init
 - lspmac.c, 108
- lspmac_motor_struct, 29
 - actual_pos_cnts_p, 31
 - cond, 31
 - dac_mvar, 31
 - format, 31
 - lut, 31
 - max_accel, 31
 - max_speed, 31
 - motion_seen, 31
 - motor_num, 31
 - moveAbs, 32
 - mutex, 32
 - name, 32
 - nlut, 32
 - not_done, 32
 - position, 32
 - pq, 32
 - read, 32
 - read_mask, 32
 - read_ptr, 32
 - reported_position, 33
 - requested_pos_cnts, 33
 - requested_position, 33
 - status1, 33
 - status2, 33
 - u2c, 33
 - units, 33
 - update_resolution, 33
 - win, 33
 - write_fmt, 34
- lspmac_motor_t
 - pgpmac.h, 157
- lspmac_motors
 - lspmac.c, 128
 - pgpmac.h, 164
- lspmac_moveabs_bio_queue
 - lspmac.c, 109
- lspmac_moveabs_fshut_queue
 - lspmac.c, 109
- lspmac_moveabs_queue
 - lspmac.c, 110
- lspmac_moveabs_wait
 - lspmac.c, 110
- lspmac_movedac_queue
 - lspmac.c, 112
- lspmac_movezoom_queue
 - lspmac.c, 112
- lspmac_moving_cond
 - lspmac.c, 128
 - pgpmac.h, 164
- lspmac_moving_flags
 - lspmac.c, 128
 - pgpmac.h, 164
- lspmac_moving_mutex
 - lspmac.c, 128
 - pgpmac.h, 164

- lspmac_next_state
 - lspmac.c, 113
- lspmac_nmotors
 - lspmac.c, 129
 - pgpmac.h, 164
- lspmac_pmacmotor_read
 - lspmac.c, 115
- lspmac_pop_queue
 - lspmac.c, 116
- lspmac_pop_reply
 - lspmac.c, 116
- lspmac_push_queue
 - lspmac.c, 117
- lspmac_Reset
 - lspmac.c, 117
- lspmac_run
 - lspmac.c, 117
 - pgpmac.h, 160
- lspmac_send_command
 - lspmac.c, 117
- lspmac_sendcmd_nocb
 - lspmac.c, 119
- lspmac_SendControlReplyPrintCB
 - lspmac.c, 119
- lspmac_Service
 - lspmac.c, 119
- lspmac_shutter_cond
 - lspmac.c, 129
 - pgpmac.h, 164
- lspmac_shutter_has_opened
 - lspmac.c, 129
 - pgpmac.h, 165
- lspmac_shutter_mutex
 - lspmac.c, 129
 - pgpmac.h, 165
- lspmac_shutter_read
 - lspmac.c, 123
- lspmac_shutter_state
 - lspmac.c, 129
 - pgpmac.h, 165
- lspmac_SockFlush
 - lspmac.c, 123
- lspmac_SockGetmem
 - lspmac.c, 123
- lspmac_SockSendControlCharPrint
 - lspmac.c, 123
- lspmac_SockSendline
 - lspmac.c, 123
- lspmac_SockSendline_nr
 - lspmac.c, 124
- lspmac_worker
 - lspmac.c, 124
- lsupdate.c, 132
 - lsupdate_init, 132
 - lsupdate_run, 132
 - lsupdate_thread, 134
 - lsupdate_updateit, 133
 - lsupdate_worker, 134
- lsupdate_init
 - lsupdate.c, 132
 - pgpmac.h, 161
- lsupdate_run
 - lsupdate.c, 132
 - pgpmac.h, 161
- lsupdate_thread
 - lsupdate.c, 134
- lsupdate_updateit
 - lsupdate.c, 133
- lsupdate_worker
 - lsupdate.c, 134
- lut
 - lspmac_motor_struct, 31
- main
 - pgpmac.c, 147
- max_accel
 - lspmac_motor_struct, 31
- max_speed
 - lspmac_motor_struct, 31
- md2_status
 - lspmac.c, 129
- md2_status_mutex
 - lspmac.c, 129
 - pgpmac.h, 165
- md2_status_t
 - lspmac.c, 94
- md2cmds.c, 135
 - logtime, 136
 - md2cmds_center, 137
 - md2cmds_cmd, 144
 - md2cmds_collect, 137
 - md2cmds_cond, 144
 - md2cmds_init, 140
 - md2cmds_moveAbs, 140
 - md2cmds_mutex, 145
 - md2cmds_mvcenter_move, 142
 - md2cmds_mvcenter_prep, 142
 - md2cmds_mvcenter_wait, 143
 - md2cmds_pg_cond, 145
 - md2cmds_pg_mutex, 145
 - md2cmds_rotate, 143
 - md2cmds_run, 143
 - md2cmds_thread, 145
 - md2cmds_transfer, 144
 - md2cmds_worker, 144
- md2cmds_center
 - md2cmds.c, 137
- md2cmds_cmd

- md2cmds.c, 144
- pgpmac.h, 165
- MD2CMDS_CMD_LENGTH
 - pgpmac.h, 157
- md2cmds_collect
 - md2cmds.c, 137
- md2cmds_cond
 - md2cmds.c, 144
 - pgpmac.h, 165
- md2cmds_init
 - md2cmds.c, 140
 - pgpmac.h, 161
- md2cmds_moveAbs
 - md2cmds.c, 140
- md2cmds_mutex
 - md2cmds.c, 145
 - pgpmac.h, 165
- md2cmds_mvcenter_move
 - md2cmds.c, 142
- md2cmds_mvcenter_prep
 - md2cmds.c, 142
- md2cmds_mvcenter_wait
 - md2cmds.c, 143
- md2cmds_pg_cond
 - md2cmds.c, 145
 - pgpmac.h, 165
- md2cmds_pg_mutex
 - md2cmds.c, 145
 - pgpmac.h, 165
- md2cmds_rotate
 - md2cmds.c, 143
- md2cmds_run
 - md2cmds.c, 143
 - pgpmac.h, 161
- md2cmds_thread
 - md2cmds.c, 145
- md2cmds_transfer
 - md2cmds.c, 144
- md2cmds_worker
 - md2cmds.c, 144
- md2StatusStruct, 35
 - acc1lc_1, 36
 - acc1lc_2, 36
 - acc1lc_3, 36
 - acc1lc_5, 36
 - acc1lc_6, 37
 - alignx_act_pos, 37
 - alignx_status_1, 37
 - alignx_status_2, 37
 - aligny_act_pos, 37
 - aligny_status_1, 37
 - aligny_status_2, 37
 - alignz_act_pos, 37
 - alignz_status_1, 37
 - alignz_status_2, 37
 - analyzer_act_pos, 37
 - analyzer_status_1, 38
 - analyzer_status_2, 38
 - aperturey_act_pos, 38
 - aperturey_status_1, 38
 - aperturey_status_2, 38
 - aperturez_act_pos, 38
 - aperturez_status_1, 38
 - aperturez_status_2, 38
 - back_dac, 38
 - capy_act_pos, 38
 - capy_status_1, 38
 - capy_status_2, 39
 - capz_act_pos, 39
 - capz_status_1, 39
 - capz_status_2, 39
 - centerx_act_pos, 39
 - centerx_status_1, 39
 - centerx_status_2, 39
 - centery_act_pos, 39
 - centery_status_1, 39
 - centery_status_2, 39
 - dummy1, 39
 - dummy2, 40
 - dummy3, 40
 - dummy4, 40
 - dummy5, 40
 - dummy6, 40
 - dummy7, 40
 - dummy8, 40
 - dummy9, 40
 - dummyA, 40
 - dummyB, 40
 - front_dac, 40
 - fs_has_opened, 41
 - fs_has_opened_globally, 41
 - fs_is_open, 41
 - kappa_act_pos, 41
 - kappa_status_1, 41
 - kappa_status_2, 41
 - moving_flags, 41
 - number_passes, 41
 - omega_act_pos, 41
 - omega_status_1, 41
 - omega_status_2, 41
 - phi_act_pos, 42
 - phi_status_1, 42
 - phi_status_2, 42
 - phiscan, 42
 - scint_act_pos, 42
 - scint_piezo, 42
 - scint_status_1, 42
 - scint_status_2, 42

- zoom_act_pos, 42
- zoom_status_1, 42
- zoom_status_2, 42
- motion_seen
 - lspmac_motor_struct, 31
- motor_num
 - lspmac_motor_struct, 31
- moveAbs
 - lspmac_motor_struct, 32
- moving_flags
 - md2StatusStruct, 41
- mutex
 - lspg_lock_detector_struct, 9
 - lspg_lock_diffractionmeter_struct, 10
 - lspg_nextshot_struct, 21
 - lspg_seq_run_prep_struct, 24
 - lspg_wait_for_detector_struct, 25
 - lspmac_motor_struct, 32
- name
 - lspmac_motor_struct, 32
- ncurses_mutex
 - pgpmac.c, 150
 - pgpmac.h, 166
- new_value_ready
 - lspg_lock_detector_struct, 9
 - lspg_lock_diffractionmeter_struct, 10
 - lspg_nextshot_struct, 21
 - lspg_seq_run_prep_struct, 24
 - lspg_wait_for_detector_struct, 25
- nlut
 - lspmac_motor_struct, 32
- no_reply
 - lspmac_cmd_queue_struct, 27
- no_rows_returned
 - lspg_nextshot_struct, 21
- not_done
 - lspmac_motor_struct, 32
- now
 - lspmac.c, 129
- number_passes
 - md2StatusStruct, 41
- omega
 - lspmac.c, 129
 - pgpmac.h, 166
- omega_act_pos
 - md2StatusStruct, 41
- omega_status_1
 - md2StatusStruct, 41
- omega_status_2
 - md2StatusStruct, 41
- onResponse
 - lspgQueryQueueStruct, 26
 - lspmac_cmd_queue_struct, 27
- pcmd
 - lspmac_cmd_queue_struct, 27
- pg_queue_mutex
 - lspg.c, 82
- pgpmac.c, 146
 - main, 147
 - ncurses_mutex, 150
 - pgpmac_printf, 149
 - stdinfd, 150
 - stdinService, 149
 - term_input, 151
 - term_output, 151
 - term_status, 151
 - term_status2, 151
- pgpmac.h, 152
 - alignx, 162
 - aligny, 162
 - alignz, 162
 - anal, 162
 - apery, 162
 - aperz, 163
 - blight, 163
 - blight_up, 163
 - capy, 163
 - capz, 163
 - cenx, 163
 - ceny, 163
 - flight, 163
 - fscint, 163
 - fshut, 164
 - kappa, 164
 - LS_DISPLAY_WINDOW_HEIGHT, 157
 - LS_DISPLAY_WINDOW_WIDTH, 157
 - LS_PG_QUERY_STRING_LENGTH, 157
 - lspg_init, 158
 - lspg_nextshot, 164
 - lspg_nextshot_t, 157
 - lspg_run, 158
 - lspg_seq_run_prep_all, 158
 - lspg_zoom_lut_call, 158
 - lspmac_init, 159
 - lspmac_motor_t, 157
 - lspmac_motors, 164
 - lspmac_moving_cond, 164
 - lspmac_moving_flags, 164
 - lspmac_moving_mutex, 164
 - lspmac_nmotors, 164
 - lspmac_run, 160
 - lspmac_shutter_cond, 164
 - lspmac_shutter_has_opened, 165
 - lspmac_shutter_mutex, 165
 - lspmac_shutter_state, 165

- lsupdate_init, 161
- lsupdate_run, 161
- md2_status_mutex, 165
- md2cmds_cmd, 165
- MD2CMDS_CMD_LENGTH, 157
- md2cmds_cond, 165
- md2cmds_init, 161
- md2cmds_mutex, 165
- md2cmds_pg_cond, 165
- md2cmds_pg_mutex, 165
- md2cmds_run, 161
- ncurses_mutex, 166
- omega, 166
- pgpmac_printf, 161
- phi, 166
- pmac_cmd_queue_t, 157
- pmac_cmd_t, 157
- PmacSockSendline, 162
- scinz, 166
- term_input, 166
- term_output, 166
- term_status, 166
- term_status2, 166
- zoom, 166
- pgpmac_printf
 - pgpmac.c, 149
 - pgpmac.h, 161
- phi
 - lspmac.c, 130
 - pgpmac.h, 166
- phi_act_pos
 - md2StatusStruct, 42
- phi_status_1
 - md2StatusStruct, 42
- phi_status_2
 - md2StatusStruct, 42
- phiscan
 - md2StatusStruct, 42
- PMAC_CMD_QUEUE_LENGTH
 - lspmac.c, 92
- pmac_cmd_queue_t
 - pgpmac.h, 157
- pmac_cmd_size
 - lspmac.c, 92
- pmac_cmd_t
 - pgpmac.h, 157
- pmac_error_strs
 - lspmac.c, 130
- PMAC_MIN_CMD_TIME
 - lspmac.c, 92
- pmac_queue_cond
 - lspmac.c, 130
- pmac_queue_mutex
 - lspmac.c, 130
- pmac_thread
 - lspmac.c, 130
- pmacfd
 - lspmac.c, 131
- PMACPORT
 - lspmac.c, 92
- PmacSockSendline
 - pgpmac.h, 162
- position
 - lspmac_motor_struct, 32
- pq
 - lspmac_motor_struct, 32
- q
 - lspg.c, 82
- qs
 - lspgQueryQueueStruct, 26
- rbuff
 - lspmac_cmd_queue_struct, 27
- read
 - lspmac_motor_struct, 32
- read_mask
 - lspmac_motor_struct, 32
- read_ptr
 - lspmac_motor_struct, 32
- reported_position
 - lspmac_motor_struct, 33
- Request
 - tagEthernetCmd, 44
- requested_pos_cnts
 - lspmac_motor_struct, 33
- requested_position
 - lspmac_motor_struct, 33
- RequestType
 - tagEthernetCmd, 44
- rr_cmd
 - lspmac.c, 131
- scint_act_pos
 - md2StatusStruct, 42
- scint_piezo
 - md2StatusStruct, 42
- scint_status_1
 - md2StatusStruct, 42
- scint_status_2
 - md2StatusStruct, 42
- scinz
 - lspmac.c, 131
 - pgpmac.h, 166
- sfn
 - lspg_nextshot_struct, 21
- sfn_isnull
 - lspg_nextshot_struct, 21

- index
 - lspg_nextshot_struct, [21](#)
- index2
 - lspg_nextshot_struct, [21](#)
- index2_isnull
 - lspg_nextshot_struct, [21](#)
- index_isnull
 - lspg_nextshot_struct, [21](#)
- skey
 - lspg_nextshot_struct, [21](#)
- skey_isnull
 - lspg_nextshot_struct, [22](#)
- sstart
 - lspg_nextshot_struct, [22](#)
- sstart2
 - lspg_nextshot_struct, [22](#)
- sstart2_isnull
 - lspg_nextshot_struct, [22](#)
- sstart_isnull
 - lspg_nextshot_struct, [22](#)
- status1
 - lspmac_motor_struct, [33](#)
- status2
 - lspmac_motor_struct, [33](#)
- stdinfda
 - pgpmac.c, [150](#)
- stdinService
 - pgpmac.c, [149](#)
- stype
 - lspg_nextshot_struct, [22](#)
- stype2
 - lspg_nextshot_struct, [22](#)
- stype2_isnull
 - lspg_nextshot_struct, [22](#)
- stype_isnull
 - lspg_nextshot_struct, [22](#)
- tagEthernetCmd, [44](#)
 - bData, [44](#)
 - Request, [44](#)
 - RequestType, [44](#)
 - wIndex, [44](#)
 - wLength, [45](#)
 - wValue, [45](#)
- term_input
 - pgpmac.c, [151](#)
 - pgpmac.h, [166](#)
- term_output
 - pgpmac.c, [151](#)
 - pgpmac.h, [166](#)
- term_status
 - pgpmac.c, [151](#)
 - pgpmac.h, [166](#)
- term_status2
 - pgpmac.c, [151](#)
 - pgpmac.h, [166](#)
- time_sent
 - lspmac_cmd_queue_struct, [28](#)
- u2c
 - lspmac_motor_struct, [33](#)
- units
 - lspmac_motor_struct, [33](#)
- update_resolution
 - lspmac_motor_struct, [33](#)
- VR_CTRL_RESPONSE
 - lspmac.c, [92](#)
- VR_DOWNLOAD
 - lspmac.c, [92](#)
- VR_FWDOWNLOAD
 - lspmac.c, [92](#)
- VR_IPADDRESS
 - lspmac.c, [93](#)
- VR_PMAC_FLUSH
 - lspmac.c, [93](#)
- VR_PMAC_GETBUFFER
 - lspmac.c, [93](#)
- VR_PMAC_GETLINE
 - lspmac.c, [93](#)
- VR_PMAC_GETMEM
 - lspmac.c, [93](#)
- VR_PMAC_GETRESPONSE
 - lspmac.c, [93](#)
- VR_PMAC_PORT
 - lspmac.c, [93](#)
- VR_PMAC_READREADY
 - lspmac.c, [93](#)
- VR_PMAC_SENDCTRLCHAR
 - lspmac.c, [93](#)
- VR_PMAC_SENDLINE
 - lspmac.c, [93](#)
- VR_PMAC_SETBIT
 - lspmac.c, [93](#)
- VR_PMAC_SETBITS
 - lspmac.c, [94](#)
- VR_PMAC_SETMEM
 - lspmac.c, [94](#)
- VR_PMAC_WRITEBUFFER
 - lspmac.c, [94](#)
- VR_PMAC_WRITEERROR
 - lspmac.c, [94](#)
- VR_UPLOAD
 - lspmac.c, [94](#)
- win
 - lspmac_motor_struct, [33](#)
- wIndex

- tagEthernetCmd, [44](#)
- wLength
 - tagEthernetCmd, [45](#)
- write_fmt
 - lspmac_motor_struct, [34](#)
- wValue
 - tagEthernetCmd, [45](#)
- zoom
 - lspmac.c, [131](#)
 - pgpmac.h, [166](#)
- zoom_act_pos
 - md2StatusStruct, [42](#)
- zoom_status_1
 - md2StatusStruct, [42](#)
- zoom_status_2
 - md2StatusStruct, [42](#)