

HA Energy

Generated by Doxygen 1.9.4

| | |
|---------------------------------------|----------|
| 1 Data Structure Index | 1 |
| 1.1 Data Structures | 1 |
| 2 File Index | 3 |
| 2.1 File List | 3 |
| 3 Data Structure Documentation | 5 |
| 3.1 energy_type Struct Reference | 5 |
| 3.1.1 Field Documentation | 6 |
| 3.1.1.1 ac_low_adj | 6 |
| 3.1.1.2 ac_mismatch | 6 |
| 3.1.1.3 ac_sw_on | 7 |
| 3.1.1.4 ac_sw_status | 7 |
| 3.1.1.5 client_ha | 7 |
| 3.1.1.6 client_p | 7 |
| 3.1.1.7 client_sd | 7 |
| 3.1.1.8 dc_mismatch | 7 |
| 3.1.1.9 dl_excess | 7 |
| 3.1.1.10 dl_excess_adj | 7 |
| 3.1.1.11 dumpload | 8 |
| 3.1.1.12 fm80 | 8 |
| 3.1.1.13 gti_delay | 8 |
| 3.1.1.14 gti_low_adj | 8 |
| 3.1.1.15 gti_sw_on | 8 |
| 3.1.1.16 gti_sw_status | 8 |
| 3.1.1.17 ha_lock | 8 |
| 3.1.1.18 homeassistant | 8 |
| 3.1.1.19 iammeter | 9 |
| 3.1.1.20 im_delay | 9 |
| 3.1.1.21 im_display | 9 |
| 3.1.1.22 im_vars | 9 |
| 3.1.1.23 link | 9 |
| 3.1.1.24 log_spam | 9 |
| 3.1.1.25 log_time_reset | 9 |
| 3.1.1.26 mode | 9 |
| 3.1.1.27 mode_mismatch | 10 |
| 3.1.1.28 mvar | 10 |
| 3.1.1.29 once_ac | 10 |
| 3.1.1.30 once_gti | 10 |
| 3.1.1.31 once_gti_zero | 10 |
| 3.1.1.32 print_vars | 10 |
| 3.1.1.33 rc | 10 |
| 3.1.1.34 sane | 10 |

| | |
|-----------------------------------|----|
| 3.1.1.35 solar_mode | 11 |
| 3.1.1.36 solar_shutdown | 11 |
| 3.1.1.37 speed_go | 11 |
| 3.1.1.38 startup | 11 |
| 3.1.1.39 ten_sec_clock | 11 |
| 3.2 ha_flag_type Struct Reference | 11 |
| 3.2.1 Field Documentation | 12 |
| 3.2.1.1 deliveredtoken | 12 |
| 3.2.1.2 energy_mode | 12 |
| 3.2.1.3 ha_id | 12 |
| 3.2.1.4 rec_ok | 12 |
| 3.2.1.5 receivedtoken | 12 |
| 3.2.1.6 runner | 12 |
| 3.2.1.7 var_update | 12 |
| 3.3 link_type Struct Reference | 13 |
| 3.3.1 Field Documentation | 13 |
| 3.3.1.1 iammeter_count | 13 |
| 3.3.1.2 iammeter_error | 13 |
| 3.3.1.3 mqtt_count | 13 |
| 3.3.1.4 mqtt_error | 13 |
| 3.3.1.5 shutdown | 13 |
| 3.4 local_type Struct Reference | 14 |
| 3.4.1 Field Documentation | 14 |
| 3.4.1.1 ac_weight | 14 |
| 3.4.1.2 bat_c_std_dev | 14 |
| 3.4.1.3 bat_current | 14 |
| 3.4.1.4 bat_voltage | 14 |
| 3.4.1.5 batc_std_dev | 14 |
| 3.4.1.6 coef | 15 |
| 3.4.1.7 gti_weight | 15 |
| 3.4.1.8 pv_voltage | 15 |
| 3.5 mode_type Struct Reference | 15 |
| 3.5.1 Field Documentation | 16 |
| 3.5.1.1 bat_crit | 16 |
| 3.5.1.2 con0 | 16 |
| 3.5.1.3 con1 | 16 |
| 3.5.1.4 con2 | 17 |
| 3.5.1.5 con3 | 17 |
| 3.5.1.6 con4 | 17 |
| 3.5.1.7 con5 | 17 |
| 3.5.1.8 con6 | 17 |
| 3.5.1.9 con7 | 17 |

| | |
|------------------------------------------------|-----------|
| 3.5.1.10 data_error | 17 |
| 3.5.1.11 dl_mqtt_max | 17 |
| 3.5.1.12 E | 18 |
| 3.5.1.13 error | 18 |
| 3.5.1.14 gti_dumpload | 18 |
| 3.5.1.15 in_pid_control | 18 |
| 3.5.1.16 mode | 18 |
| 3.5.1.17 mode_tmr | 18 |
| 3.5.1.18 no_float | 18 |
| 3.5.1.19 off_grid | 18 |
| 3.5.1.20 pid | 19 |
| 3.5.1.21 pv_bias | 19 |
| 3.5.1.22 R | 19 |
| 3.5.1.23 sequence | 19 |
| 3.5.1.24 target | 19 |
| 3.5.1.25 total_system | 19 |
| 3.6 SPid Struct Reference | 19 |
| 3.6.1 Field Documentation | 20 |
| 3.6.1.1 dGain | 20 |
| 3.6.1.2 dState | 20 |
| 3.6.1.3 iGain | 20 |
| 3.6.1.4 iMax | 20 |
| 3.6.1.5 iMin | 20 |
| 3.6.1.6 iState | 20 |
| 3.6.1.7 pGain | 20 |
| 4 File Documentation | 21 |
| 4.1 energy.c File Reference | 21 |
| 4.1.1 Macro Definition Documentation | 22 |
| 4.1.1.1 _DEFAULT_SOURCE | 22 |
| 4.1.2 Function Documentation | 22 |
| 4.1.2.1 connlost() | 22 |
| 4.1.2.2 ha_ac_off() | 22 |
| 4.1.2.3 ha_ac_on() | 23 |
| 4.1.2.4 ha_dc_off() | 23 |
| 4.1.2.5 ha_dc_on() | 23 |
| 4.1.2.6 log_time() | 23 |
| 4.1.2.7 log_timer() | 24 |
| 4.1.2.8 main() | 24 |
| 4.1.2.9 ramp_down_ac() | 30 |
| 4.1.2.10 ramp_down_gti() | 31 |
| 4.1.2.11 ramp_up_ac() | 31 |

| | |
|---------------------------------------|----|
| 4.1.2.12 ramp_up_gti() | 31 |
| 4.1.2.13 sanity_check() | 32 |
| 4.1.2.14 showIP() | 32 |
| 4.1.2.15 skeleton_daemon() | 33 |
| 4.1.2.16 solar_shutdown() | 34 |
| 4.1.2.17 sync_ha() | 34 |
| 4.1.2.18 timer_callback() | 35 |
| 4.1.3 Variable Documentation | 35 |
| 4.1.3.1 board_name | 35 |
| 4.1.3.2 driver_name | 35 |
| 4.1.3.3 E | 35 |
| 4.1.3.4 fout | 35 |
| 4.1.3.5 ha_flag_vars_ha | 36 |
| 4.1.3.6 ha_flag_vars_pc | 36 |
| 4.1.3.7 ha_flag_vars_sd | 36 |
| 4.1.3.8 ha_flag_vars_ss | 36 |
| 4.2 ha_energy/.dep.inc File Reference | 37 |
| 4.3 ha_energy/bsoc.c File Reference | 37 |
| 4.3.1 Function Documentation | 37 |
| 4.3.1.1 ac0_filter() | 38 |
| 4.3.1.2 ac1_filter() | 38 |
| 4.3.1.3 ac2_filter() | 38 |
| 4.3.1.4 ac_test() | 38 |
| 4.3.1.5 bat_current_stable() | 38 |
| 4.3.1.6 bsoc_ac() | 39 |
| 4.3.1.7 bsoc_data_collect() | 39 |
| 4.3.1.8 bsoc_gti() | 39 |
| 4.3.1.9 bsoc_init() | 40 |
| 4.3.1.10 bsoc_set_mode() | 40 |
| 4.3.1.11 bsoc_set_std_dev() | 41 |
| 4.3.1.12 calculateStandardDeviation() | 42 |
| 4.3.1.13 dc0_filter() | 42 |
| 4.3.1.14 dc1_filter() | 42 |
| 4.3.1.15 dc2_filter() | 43 |
| 4.3.1.16 drive0_filter() | 43 |
| 4.3.1.17 drive1_filter() | 43 |
| 4.3.1.18 error_filter() | 43 |
| 4.3.1.19 get_batc_dev() | 43 |
| 4.3.1.20 gti_test() | 44 |
| 4.3.2 Variable Documentation | 44 |
| 4.3.2.1 L | 44 |
| 4.3.2.2 mqtt_name | 44 |

| | |
|--------------------------------------------------------------------------|----|
| 4.4 ha_energy/bsoc.h File Reference | 44 |
| 4.4.1 Macro Definition Documentation | 46 |
| 4.4.1.1 BAT_C_DRAW | 46 |
| 4.4.1.2 COEF | 46 |
| 4.4.1.3 COEFF | 46 |
| 4.4.1.4 COEFN | 46 |
| 4.4.1.5 DEV_SIZE | 46 |
| 4.4.1.6 MAX_BATC_DEV | 47 |
| 4.4.1.7 MIN_BAT_KW | 47 |
| 4.4.1.8 MIN_BAT_VOLTS | 47 |
| 4.4.1.9 MIN_PV_VOLTS | 47 |
| 4.4.1.10 PBAL_OFFSET | 47 |
| 4.4.1.11 PV_FULL_PWR | 47 |
| 4.4.1.12 PV_MIN_PWR | 47 |
| 4.4.1.13 PV_V_FAKE | 47 |
| 4.4.1.14 PV_V_NOM | 48 |
| 4.4.2 Function Documentation | 48 |
| 4.4.2.1 ac0_filter() | 48 |
| 4.4.2.2 ac1_filter() | 48 |
| 4.4.2.3 ac2_filter() | 48 |
| 4.4.2.4 ac_test() | 48 |
| 4.4.2.5 bat_current_stable() | 49 |
| 4.4.2.6 bsoc_ac() | 49 |
| 4.4.2.7 bsoc_data_collect() | 49 |
| 4.4.2.8 bsoc_gti() | 50 |
| 4.4.2.9 bsoc_init() | 50 |
| 4.4.2.10 bsoc_set_mode() | 50 |
| 4.4.2.11 bsoc_set_std_dev() | 52 |
| 4.4.2.12 calculateStandardDeviation() | 52 |
| 4.4.2.13 dc0_filter() | 52 |
| 4.4.2.14 dc1_filter() | 52 |
| 4.4.2.15 dc2_filter() | 53 |
| 4.4.2.16 drive0_filter() | 53 |
| 4.4.2.17 drive1_filter() | 53 |
| 4.4.2.18 get_batc_dev() | 53 |
| 4.4.2.19 gti_test() | 53 |
| 4.5 bsoc.h | 54 |
| 4.6 ha_energy/build/Debug/GNU-Linux/_ext/5c0/energy.o.d File Reference | 55 |
| 4.7 ha_energy/build/Release/GNU-Linux/_ext/5c0/energy.o.d File Reference | 55 |
| 4.8 ha_energy/build/Debug/GNU-Linux/bsoc.o.d File Reference | 55 |
| 4.9 ha_energy/build/Release/GNU-Linux/bsoc.o.d File Reference | 55 |
| 4.10 ha_energy/build/Debug/GNU-Linux/http_vars.o.d File Reference | 55 |

| | |
|-------------------------------------------------------------------------------|----|
| 4.11 ha_energy/build/Release/GNU-Linux/http_vars.o.d File Reference | 55 |
| 4.12 ha_energy/build/Debug/GNU-Linux/mqtt_rec.o.d File Reference | 55 |
| 4.13 ha_energy/build/Release/GNU-Linux/mqtt_rec.o.d File Reference | 55 |
| 4.14 ha_energy/build/Debug/GNU-Linux/mqtt_vars.o.d File Reference | 55 |
| 4.15 ha_energy/build/Release/GNU-Linux/mqtt_vars.o.d File Reference | 55 |
| 4.16 ha_energy/build/Debug/GNU-Linux/pid.o.d File Reference | 55 |
| 4.17 ha_energy/build/Release/GNU-Linux/pid.o.d File Reference | 55 |
| 4.18 ha_energy/energy.h File Reference | 55 |
| 4.18.1 Macro Definition Documentation | 59 |
| 4.18.1.1 ADDRESS | 59 |
| 4.18.1.2 BAL_MAX_ENERGY_AC | 59 |
| 4.18.1.3 BAL_MAX_ENERGY_GTI | 59 |
| 4.18.1.4 BAL_MIN_ENERGY_AC | 60 |
| 4.18.1.5 BAL_MIN_ENERGY_GTI | 60 |
| 4.18.1.6 BAMPS_SANE | 60 |
| 4.18.1.7 BAT_CRITICAL | 60 |
| 4.18.1.8 BAT_M_KW | 60 |
| 4.18.1.9 BAT_SOC_HIGH | 60 |
| 4.18.1.10 BAT_SOC_LOW | 60 |
| 4.18.1.11 BAT_SOC_LOW_AC | 60 |
| 4.18.1.12 BAT_SOC_TOP | 61 |
| 4.18.1.13 CLIENTID1 | 61 |
| 4.18.1.14 CLIENTID2 | 61 |
| 4.18.1.15 CLIENTID3 | 61 |
| 4.18.1.16 CMD_SEC | 61 |
| 4.18.1.17 CRITIAL_SHUTDOWN_LOG | 61 |
| 4.18.1.18 DAQ_STR | 61 |
| 4.18.1.19 DAQ_STR_M | 61 |
| 4.18.1.20 DL_AC_DC_EFF | 62 |
| 4.18.1.21 GTI_DELAY | 62 |
| 4.18.1.22 IAM_DELAY | 62 |
| 4.18.1.23 IM_DELAY | 62 |
| 4.18.1.24 IM_DISPLAY | 62 |
| 4.18.1.25 L1_P | 62 |
| 4.18.1.26 L2_P | 62 |
| 4.18.1.27 L3_P | 62 |
| 4.18.1.28 LADDRESS | 63 |
| 4.18.1.29 LOG_TO_FILE | 63 |
| 4.18.1.30 LOG_TO_FILE_ALT | 63 |
| 4.18.1.31 LOG_VERSION | 63 |
| 4.18.1.32 LOW_LOG_SPAM | 63 |
| 4.18.1.33 MAX_ERROR | 63 |

| | |
|-----------------------------------------|----|
| 4.18.1.34 MAX_IM_VAR | 63 |
| 4.18.1.35 MAX_LOG_SPAM | 63 |
| 4.18.1.36 MIN_BAT_KW_AC_HI | 64 |
| 4.18.1.37 MIN_BAT_KW_AC_LO | 64 |
| 4.18.1.38 MIN_BAT_KW_BSOC_HI | 64 |
| 4.18.1.39 MIN_BAT_KW_BSOC_SLP | 64 |
| 4.18.1.40 MIN_BAT_KW_GTI_HI | 64 |
| 4.18.1.41 MIN_BAT_KW_GTI_LO | 64 |
| 4.18.1.42 MQTT_TIMEOUT | 64 |
| 4.18.1.43 MQTT_VERSION | 64 |
| 4.18.1.44 NO_CYLON | 65 |
| 4.18.1.45 NORM_MODE | 65 |
| 4.18.1.46 PAMPS_SANE | 65 |
| 4.18.1.47 PID_MODE | 65 |
| 4.18.1.48 PV_BIAS | 65 |
| 4.18.1.49 PV_BIAS_FLOAT | 65 |
| 4.18.1.50 PV_BIAS_LOW | 65 |
| 4.18.1.51 PV_BIAS_RATE | 65 |
| 4.18.1.52 PV_BIAS_SLEEP | 66 |
| 4.18.1.53 PV_BIAS_ZERO | 66 |
| 4.18.1.54 PV_DL_B_AH_LOW | 66 |
| 4.18.1.55 PV_DL_B_AH_MIN | 66 |
| 4.18.1.56 PV_DL_B_V_LOW | 66 |
| 4.18.1.57 PV_DL_BIAS_RATE | 66 |
| 4.18.1.58 PV_DL_EXCESS | 66 |
| 4.18.1.59 PV_DL_MPTT_EXCESS | 66 |
| 4.18.1.60 PV_DL_MPTT_IDLE | 67 |
| 4.18.1.61 PV_DL_MPTT_MAX | 67 |
| 4.18.1.62 PV_IGAIN | 67 |
| 4.18.1.63 PV_IMAX | 67 |
| 4.18.1.64 PV_PGAIN | 67 |
| 4.18.1.65 PVOLTS_SANE | 67 |
| 4.18.1.66 PWA_SANE | 67 |
| 4.18.1.67 PWA_SLEEP | 67 |
| 4.18.1.68 QOS | 68 |
| 4.18.1.69 RBUF_SIZ | 68 |
| 4.18.1.70 RESET_LOG_SPAM | 68 |
| 4.18.1.71 SBUF_SIZ | 68 |
| 4.18.1.72 SPACING_USEC | 68 |
| 4.18.1.73 SW_QOS | 68 |
| 4.18.1.74 SYSLOG_SIZ | 68 |
| 4.18.1.75 TIME_SYNC_SEC | 68 |

| | | |
|-----------|--------------------------------|----|
| 4.18.1.76 | TIMEOUT | 69 |
| 4.18.1.77 | TNAME | 69 |
| 4.18.1.78 | TOPIC_HA | 69 |
| 4.18.1.79 | TOPIC_P | 69 |
| 4.18.1.80 | TOPIC_PACA | 69 |
| 4.18.1.81 | TOPIC_PACC | 69 |
| 4.18.1.82 | TOPIC_PDCA | 69 |
| 4.18.1.83 | TOPIC_PDCC | 69 |
| 4.18.1.84 | TOPIC_PPID | 70 |
| 4.18.1.85 | TOPIC_SD | 70 |
| 4.18.1.86 | TOPIC_SHUTDOWN | 70 |
| 4.18.1.87 | TOPIC_SPAM | 70 |
| 4.18.1.88 | TOPIC_SS | 70 |
| 4.18.1.89 | UNIT_TEST | 70 |
| 4.18.1.90 | USEC_SEC | 70 |
| 4.18.2 | Enumeration Type Documentation | 70 |
| 4.18.2.1 | energy_state | 70 |
| 4.18.2.2 | iammeter_id | 71 |
| 4.18.2.3 | iammeter_phase | 71 |
| 4.18.2.4 | mqtt_vars | 72 |
| 4.18.2.5 | running_state | 73 |
| 4.18.2.6 | sane_vars | 73 |
| 4.18.3 | Function Documentation | 74 |
| 4.18.3.1 | connlost() | 75 |
| 4.18.3.2 | ha_ac_off() | 75 |
| 4.18.3.3 | ha_ac_on() | 75 |
| 4.18.3.4 | ha_dc_off() | 75 |
| 4.18.3.5 | ha_dc_on() | 76 |
| 4.18.3.6 | iammeter_read() | 76 |
| 4.18.3.7 | iammeter_write_callback() | 76 |
| 4.18.3.8 | log_time() | 77 |
| 4.18.3.9 | log_timer() | 77 |
| 4.18.3.10 | print_im_vars() | 78 |
| 4.18.3.11 | print_mvar_vars() | 78 |
| 4.18.3.12 | ramp_down_ac() | 78 |
| 4.18.3.13 | ramp_down_gti() | 79 |
| 4.18.3.14 | ramp_up_ac() | 79 |
| 4.18.3.15 | ramp_up_gti() | 79 |
| 4.18.3.16 | sanity_check() | 80 |
| 4.18.3.17 | sync_ha() | 80 |
| 4.18.3.18 | timer_callback() | 81 |
| 4.18.4 | Variable Documentation | 81 |

| | |
|-----------------------------------------------------|-----|
| 4.18.4.1 E | 81 |
| 4.18.4.2 fout | 81 |
| 4.18.4.3 ha_flag_vars_ss | 81 |
| 4.19 energy.h | 82 |
| 4.20 ha_energy/http_vars.c File Reference | 86 |
| 4.20.1 Function Documentation | 86 |
| 4.20.1.1 iammeter_get_data() | 87 |
| 4.20.1.2 iammeter_read() | 87 |
| 4.20.1.3 iammeter_write_callback() | 87 |
| 4.20.1.4 mqtt_gti_time() | 88 |
| 4.20.1.5 print_im_vars() | 89 |
| 4.20.2 Variable Documentation | 89 |
| 4.20.2.1 curl | 89 |
| 4.20.2.2 res | 89 |
| 4.21 ha_energy/http_vars.h File Reference | 89 |
| 4.21.1 Variable Documentation | 90 |
| 4.21.1.1 fout | 90 |
| 4.22 http_vars.h | 90 |
| 4.23 ha_energy/mqtt_rec.c File Reference | 91 |
| 4.23.1 Function Documentation | 91 |
| 4.23.1.1 delivered() | 91 |
| 4.23.1.2 fm80_float() | 91 |
| 4.23.1.3 fm80_sleep() | 92 |
| 4.23.1.4 json_get_data() | 92 |
| 4.23.1.5 msgarrvd() | 93 |
| 4.23.1.6 print_mvar_vars() | 94 |
| 4.24 ha_energy/mqtt_rec.h File Reference | 95 |
| 4.24.1 Macro Definition Documentation | 96 |
| 4.24.1.1 FLOAT_CODE | 96 |
| 4.24.1.2 MBMQTT | 96 |
| 4.24.1.3 RDEV_SIZE | 96 |
| 4.24.1.4 SLEEP_CODE | 96 |
| 4.24.2 Enumeration Type Documentation | 96 |
| 4.24.2.1 mqtt_id | 96 |
| 4.24.3 Function Documentation | 97 |
| 4.24.3.1 delivered() | 97 |
| 4.24.3.2 fm80_float() | 97 |
| 4.24.3.3 fm80_sleep() | 98 |
| 4.24.3.4 json_get_data() | 98 |
| 4.24.3.5 msgarrvd() | 99 |
| 4.24.4 Variable Documentation | 100 |
| 4.24.4.1 fout | 101 |

| | |
|--------------------------------------------------------------------------------------------|-----|
| 4.25 mqtt_rec.h | 101 |
| 4.26 ha_energy/mqtt_vars.c File Reference | 101 |
| 4.26.1 Macro Definition Documentation | 102 |
| 4.26.1.1 _DEFAULT_SOURCE | 102 |
| 4.26.2 Function Documentation | 102 |
| 4.26.2.1 mqtt_gti_power() | 103 |
| 4.26.2.2 mqtt_gti_time() | 103 |
| 4.26.2.3 mqtt_ha_pid() | 104 |
| 4.26.2.4 mqtt_ha_shutdown() | 105 |
| 4.26.2.5 mqtt_ha_switch() | 105 |
| 4.26.3 Variable Documentation | 106 |
| 4.26.3.1 FW_Date | 106 |
| 4.26.3.2 FW_Time | 106 |
| 4.27 ha_energy/mqtt_vars.h File Reference | 107 |
| 4.27.1 Macro Definition Documentation | 107 |
| 4.27.1.1 GTI_TOKEN_DELAY | 107 |
| 4.27.1.2 HA_SW_DELAY | 108 |
| 4.27.1.3 QOS | 108 |
| 4.27.1.4 TOKEN_DELAY | 108 |
| 4.27.2 Function Documentation | 108 |
| 4.27.2.1 mqtt_gti_power() | 108 |
| 4.27.2.2 mqtt_gti_time() | 109 |
| 4.27.2.3 mqtt_ha_pid() | 109 |
| 4.27.2.4 mqtt_ha_shutdown() | 110 |
| 4.27.2.5 mqtt_ha_switch() | 111 |
| 4.27.3 Variable Documentation | 112 |
| 4.27.3.1 mqtt_name | 112 |
| 4.28 mqtt_vars.h | 112 |
| 4.29 ha_energy/nbproject/private/c_standard_headers_indexer.c File Reference | 113 |
| 4.30 ha_energy/nbproject/private/cpp_standard_headers_indexer.cpp File Reference | 113 |
| 4.31 ha_energy/pid.c File Reference | 114 |
| 4.31.1 Function Documentation | 115 |
| 4.31.1.1 ResetPI() | 115 |
| 4.31.1.2 UpdatePI() | 115 |
| 4.32 ha_energy/pid.h File Reference | 116 |
| 4.32.1 Function Documentation | 116 |
| 4.32.1.1 ResetPI() | 116 |
| 4.32.1.2 UpdatePI() | 117 |
| 4.33 pid.h | 117 |

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

| | |
|------------------------------|----|
| energy_type | 5 |
| ha_flag_type | 11 |
| link_type | 13 |
| local_type | 14 |
| mode_type | 15 |
| SPid | 19 |

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

| | |
|------------------------------------------------------------------------------|-----|
| energy.c | 21 |
| ha_energy/.dep.inc | 37 |
| ha_energy/bsoc.c | 37 |
| ha_energy/bsoc.h | 44 |
| ha_energy/energy.h | 55 |
| ha_energy/http_vars.c | 86 |
| ha_energy/http_vars.h | 89 |
| ha_energy/mqtt_rec.c | 91 |
| ha_energy/mqtt_rec.h | 95 |
| ha_energy/mqtt_vars.c | 101 |
| ha_energy/mqtt_vars.h | 107 |
| ha_energy/pid.c | 114 |
| ha_energy/pid.h | 116 |
| ha_energy/build/Debug/GNU-Linux/bsoc.o.d | 55 |
| ha_energy/build/Debug/GNU-Linux/http_vars.o.d | 55 |
| ha_energy/build/Debug/GNU-Linux/mqtt_rec.o.d | 55 |
| ha_energy/build/Debug/GNU-Linux/mqtt_vars.o.d | 55 |
| ha_energy/build/Debug/GNU-Linux/pid.o.d | 55 |
| ha_energy/build/Debug/GNU-Linux/_ext/5c0/energy.o.d | 55 |
| ha_energy/build/Release/GNU-Linux/bsoc.o.d | 55 |
| ha_energy/build/Release/GNU-Linux/http_vars.o.d | 55 |
| ha_energy/build/Release/GNU-Linux/mqtt_rec.o.d | 55 |
| ha_energy/build/Release/GNU-Linux/mqtt_vars.o.d | 55 |
| ha_energy/build/Release/GNU-Linux/pid.o.d | 55 |
| ha_energy/build/Release/GNU-Linux/_ext/5c0/energy.o.d | 55 |
| ha_energy/nbproject/private/c_standard_headers_indexer.c | 113 |
| ha_energy/nbproject/private/cpp_standard_headers_indexer.cpp | 113 |

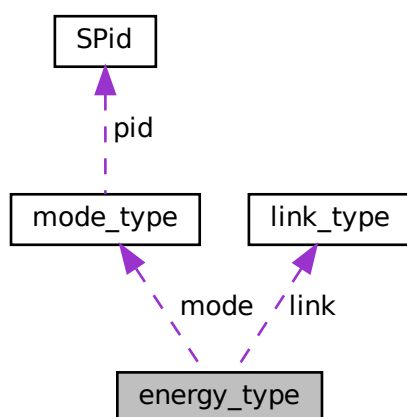
Chapter 3

Data Structure Documentation

3.1 energy_type Struct Reference

```
#include <energy.h>
```

Collaboration diagram for energy_type:



Data Fields

- volatile double `print_vars` [`MAX_IM_VAR`]
- volatile double `im_vars` [`IA_LAST`][`PHASE_LAST`]
- volatile double `mvar` [`V_DLAST+1`]
- volatile bool `once_gti`
- volatile bool `once_ac`
- volatile bool `iammeter`
- volatile bool `fm80`
- volatile bool `dumpload`

- volatile bool [homeassistant](#)
- volatile bool [once_gti_zero](#)
- volatile double [gti_low_adj](#)
- volatile double [ac_low_adj](#)
- volatile double [dl_excess_adj](#)
- volatile bool [ac_sw_on](#)
- volatile bool [gti_sw_on](#)
- volatile bool [ac_sw_status](#)
- volatile bool [gti_sw_status](#)
- volatile bool [solar_shutdown](#)
- volatile bool [solar_mode](#)
- volatile bool [startup](#)
- volatile bool [ac_mismatch](#)
- volatile bool [dc_mismatch](#)
- volatile bool [mode_mismatch](#)
- volatile bool [dl_excess](#)
- volatile uint32_t [speed_go](#)
- volatile uint32_t [im_delay](#)
- volatile uint32_t [im_display](#)
- volatile uint32_t [gti_delay](#)
- volatile int32_t [rc](#)
- volatile int32_t [sane](#)
- volatile uint32_t [ten_sec_clock](#)
- volatile uint32_t [log_spam](#)
- volatile uint32_t [log_time_reset](#)
- pthread_mutex_t [ha_lock](#)
- struct [mode_type](#) mode
- struct [link_type](#) link
- MQTTClient [client_p](#)
- MQTTClient [client_sd](#)
- MQTTClient [client_ha](#)

3.1.1 Field Documentation

3.1.1.1 `ac_low_adj`

```
volatile double energy_type::ac_low_adj
```

3.1.1.2 `ac_mismatch`

```
volatile bool energy_type::ac_mismatch
```

3.1.1.3 ac_sw_on

```
volatile bool energy_type::ac_sw_on
```

3.1.1.4 ac_sw_status

```
volatile bool energy_type::ac_sw_status
```

3.1.1.5 client_ha

```
MQTTClient energy_type::client_ha
```

3.1.1.6 client_p

```
MQTTClient energy_type::client_p
```

3.1.1.7 client_sd

```
MQTTClient energy_type::client_sd
```

3.1.1.8 dc_mismatch

```
volatile bool energy_type::dc_mismatch
```

3.1.1.9 dl_excess

```
volatile bool energy_type::dl_excess
```

3.1.1.10 dl_excess_adj

```
volatile double energy_type::dl_excess_adj
```

3.1.1.11 dumpload

```
volatile bool energy_type::dumpload
```

3.1.1.12 fm80

```
volatile bool energy_type::fm80
```

3.1.1.13 gti_delay

```
volatile uint32_t energy_type::gti_delay
```

3.1.1.14 gti_low_adj

```
volatile double energy_type::gti_low_adj
```

3.1.1.15 gti_sw_on

```
volatile bool energy_type::gti_sw_on
```

3.1.1.16 gti_sw_status

```
volatile bool energy_type::gti_sw_status
```

3.1.1.17 ha_lock

```
pthread_mutex_t energy_type::ha_lock
```

3.1.1.18 homeassistant

```
volatile bool energy_type::homeassistant
```

3.1.1.19 iammeter

```
volatile bool energy_type::iammeter
```

3.1.1.20 im_delay

```
volatile uint32_t energy_type::im_delay
```

3.1.1.21 im_display

```
volatile uint32_t energy_type::im_display
```

3.1.1.22 im_vars

```
volatile double energy_type::im_vars[IA_LAST][PHASE_LAST]
```

3.1.1.23 link

```
struct link_type energy_type::link
```

3.1.1.24 log_spam

```
volatile uint32_t energy_type::log_spam
```

3.1.1.25 log_time_reset

```
volatile uint32_t energy_type::log_time_reset
```

3.1.1.26 mode

```
struct mode_type energy_type::mode
```

3.1.1.27 mode_mismatch

```
volatile bool energy_type::mode_mismatch
```

3.1.1.28 mvar

```
volatile double energy_type::mvar[V_DLAST+1]
```

3.1.1.29 once_ac

```
volatile bool energy_type::once_ac
```

3.1.1.30 once_gti

```
volatile bool energy_type::once_gti
```

3.1.1.31 once_gti_zero

```
volatile bool energy_type::once_gti_zero
```

3.1.1.32 print_vars

```
volatile double energy_type::print_vars[MAX_IM_VAR]
```

3.1.1.33 rc

```
volatile int32_t energy_type::rc
```

3.1.1.34 sane

```
volatile int32_t energy_type::sane
```

3.1.1.35 solar_mode

```
volatile bool energy_type::solar_mode
```

3.1.1.36 solar_shutdown

```
volatile bool energy_type::solar_shutdown
```

3.1.1.37 speed_go

```
volatile uint32_t energy_type::speed_go
```

3.1.1.38 startup

```
volatile bool energy_type::startup
```

3.1.1.39 ten_sec_clock

```
volatile uint32_t energy_type::ten_sec_clock
```

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

- [ha_energy/energy.h](#)

3.2 ha_flag_type Struct Reference

```
#include <mqtt_rec.h>
```

Data Fields

- volatile MQTTClient_deliveryToken [deliveredtoken](#)
- volatile MQTTClient_deliveryToken [receivedtoken](#)
- volatile bool [runner](#)
- volatile bool [rec_ok](#)
- int32_t [ha_id](#)
- volatile int32_t [var_update](#)
- volatile int32_t [energy_mode](#)

3.2.1 Field Documentation

3.2.1.1 deliveredtoken

```
volatile MQTTClient_deliveryToken ha_flag_type::deliveredtoken
```

3.2.1.2 energy_mode

```
volatile int32_t ha_flag_type::energy_mode
```

3.2.1.3 ha_id

```
int32_t ha_flag_type::ha_id
```

3.2.1.4 rec_ok

```
volatile bool ha_flag_type::rec_ok
```

3.2.1.5 receivedtoken

```
volatile MQTTClient_deliveryToken ha_flag_type::receivedtoken
```

3.2.1.6 runner

```
volatile bool ha_flag_type::runner
```

3.2.1.7 var_update

```
volatile int32_t ha_flag_type::var_update
```

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

- [ha_energy/mqtt_rec.h](#)

3.3 link_type Struct Reference

```
#include <energy.h>
```

Data Fields

- volatile uint32_t [iammeter_error](#)
- volatile uint32_t [iammeter_count](#)
- volatile uint32_t [mqtt_error](#)
- volatile uint32_t [mqtt_count](#)
- volatile uint32_t [shutdown](#)

3.3.1 Field Documentation

3.3.1.1 iammeter_count

```
volatile uint32_t link_type::iammeter_count
```

3.3.1.2 iammeter_error

```
volatile uint32_t link_type::iammeter_error
```

3.3.1.3 mqtt_count

```
volatile uint32_t link_type::mqtt_count
```

3.3.1.4 mqtt_error

```
volatile uint32_t link_type::mqtt_error
```

3.3.1.5 shutdown

```
volatile uint32_t link_type::shutdown
```

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

- ha_energy/[energy.h](#)

3.4 local_type Struct Reference

Data Fields

- volatile double [ac_weight](#)
- volatile double [gti_weight](#)
- volatile double [pv_voltage](#)
- volatile double [bat_current](#)
- volatile double [batc_std_dev](#)
- volatile double [bat_voltage](#)
- double [bat_c_std_dev](#) [[DEV_SIZE](#)]
- double [coef](#)

3.4.1 Field Documentation

3.4.1.1 ac_weight

```
volatile double local_type::ac_weight
```

3.4.1.2 bat_c_std_dev

```
double local_type::bat_c_std_dev[DEV_SIZE]
```

3.4.1.3 bat_current

```
volatile double local_type::bat_current
```

3.4.1.4 bat_voltage

```
volatile double local_type::bat_voltage
```

3.4.1.5 batc_std_dev

```
volatile double local_type::batc_std_dev
```

3.4.1.6 coef

```
double local_type::coef
```

3.4.1.7 gti_weight

```
volatile double local_type::gti_weight
```

3.4.1.8 pv_voltage

```
volatile double local_type::pv_voltage
```

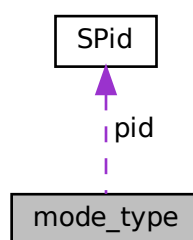
The documentation for this struct was generated from the following file:

- [ha_energy/bsoc.c](#)

3.5 mode_type Struct Reference

```
#include <energy.h>
```

Collaboration diagram for mode_type:



Data Fields

- volatile double [error](#)
- volatile double [target](#)
- volatile double [total_system](#)
- volatile double [gti_dumpload](#)
- volatile double [pv_bias](#)
- volatile double [dl_mqtt_max](#)
- volatile double [off_grid](#)
- volatile double [sequence](#)
- volatile bool [mode](#)
- volatile bool [in_pid_control](#)
- volatile bool [con0](#)
- volatile bool [con1](#)
- volatile bool [con2](#)
- volatile bool [con3](#)
- volatile bool [con4](#)
- volatile bool [con5](#)
- volatile bool [con6](#)
- volatile bool [con7](#)
- volatile bool [no_float](#)
- volatile bool [data_error](#)
- volatile bool [bat_crit](#)
- volatile uint32_t [mode_tmr](#)
- volatile struct [SPid](#) pid
- enum [energy_state](#) E
- enum [running_state](#) R

3.5.1 Field Documentation

3.5.1.1 bat_crit

```
volatile bool mode_type::bat_crit
```

3.5.1.2 con0

```
volatile bool mode_type::con0
```

3.5.1.3 con1

```
volatile bool mode_type::con1
```

3.5.1.4 con2

```
volatile bool mode_type::con2
```

3.5.1.5 con3

```
volatile bool mode_type::con3
```

3.5.1.6 con4

```
volatile bool mode_type::con4
```

3.5.1.7 con5

```
volatile bool mode_type::con5
```

3.5.1.8 con6

```
volatile bool mode_type::con6
```

3.5.1.9 con7

```
volatile bool mode_type::con7
```

3.5.1.10 data_error

```
volatile bool mode_type::data_error
```

3.5.1.11 dl_mqtt_max

```
volatile double mode_type::dl_mqtt_max
```

3.5.1.12 E

```
enum energy_state mode_type::E
```

3.5.1.13 error

```
volatile double mode_type::error
```

3.5.1.14 gti_dumpload

```
volatile double mode_type::gti_dumpload
```

3.5.1.15 in_pid_control

```
volatile bool mode_type::in_pid_control
```

3.5.1.16 mode

```
volatile bool mode_type::mode
```

3.5.1.17 mode_tmr

```
volatile uint32_t mode_type::mode_tmr
```

3.5.1.18 no_float

```
volatile bool mode_type::no_float
```

3.5.1.19 off_grid

```
volatile double mode_type::off_grid
```

3.5.1.20 pid

```
volatile struct SPid mode_type::pid
```

3.5.1.21 pv_bias

```
volatile double mode_type::pv_bias
```

3.5.1.22 R

```
enum running_state mode_type::R
```

3.5.1.23 sequence

```
volatile double mode_type::sequence
```

3.5.1.24 target

```
volatile double mode_type::target
```

3.5.1.25 total_system

```
volatile double mode_type::total_system
```

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

- [ha_energy/energy.h](#)

3.6 SPid Struct Reference

```
#include <pid.h>
```

Data Fields

- double [dState](#)
- double [iState](#)
- double [iMax](#)
- double [iMin](#)
- double [iGain](#)
- double [pGain](#)
- double [dGain](#)

3.6.1 Field Documentation

3.6.1.1 dGain

```
double SPid::dGain
```

3.6.1.2 dState

```
double SPid::dState
```

3.6.1.3 iGain

```
double SPid::iGain
```

3.6.1.4 iMax

```
double SPid::iMax
```

3.6.1.5 iMin

```
double SPid::iMin
```

3.6.1.6 iState

```
double SPid::iState
```

3.6.1.7 pGain

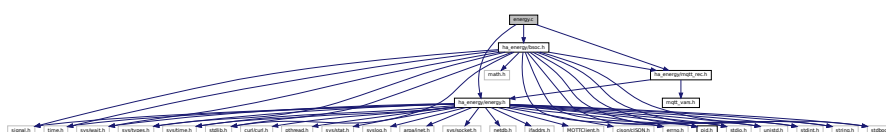
```
double SPid::pGain
```

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

- [ha_energy/pid.h](#)

File Documentation

```
#include "ha_energy/energy.h"
#include "ha_energy/mqtt_rec.h"
#include "ha_energy/bsoc.h"
Include dependency graph for energy.c:
```



- #define DEFAULT_SOURCE

- static bool `solar_shutdown` (void)
- void `showIP` (void)
- static void `skeleton_daemon` ()
- bool `sanity_check` (void)
- void `timer_callback` (int32_t signum)
- void `connlost` (void *context, char *cause)
- int `main` (int argc, char *argv[])
- void `ramp_up_gti` (MQTTClient client_p, bool start, bool excess)
- void `ramp_down_gti` (MQTTClient client_p, bool sw_off)
- void `ramp_up_ac` (MQTTClient client_p, bool start)
- void `ramp_down_ac` (MQTTClient client_p, bool sw_off)
- void `ha_ac_off` (void)
- void `ha_ac_on` (void)
- void `ha_dc_off` (void)
- void `ha_dc_on` (void)
- char * `log_time` (bool log)
- bool `sync_ha` (void)
- bool `log_timer` (void)

Variables

- struct [ha_flag_type](#) [ha_flag_vars_pc](#)
- struct [ha_flag_type](#) [ha_flag_vars_ss](#)
- struct [ha_flag_type](#) [ha_flag_vars_sd](#)
- struct [ha_flag_type](#) [ha_flag_vars_ha](#)
- const char * [board_name](#) = "NO_BOARD"
- const char * [driver_name](#) = "NO_DRIVER"
- FILE * [fout](#)
- struct [energy_type](#) [E](#)

4.1.1 Macro Definition Documentation

4.1.1.1 _DEFAULT_SOURCE

```
#define _DEFAULT_SOURCE
```

4.1.2 Function Documentation

4.1.2.1 connlost()

```
void connlost (
    void * context,
    char * cause )
298 {
299     struct ha_flag_type *ha_flag = context;
300     int32_t id_num;
301
302     // bug-out if no context variables passed to callback
303     if (context == NULL) {
304         id_num = -1;
305     } else {
306         id_num = ha_flag->ha_id;
307     }
308     fprintf(fout, "\n%s Connection lost, exit ha_energy program\n", log_time(false));
309     fprintf(fout, "%s cause: %s, %d\n", log_time(false), cause, id_num);
310     fprintf(fout, "%sDAEMON failure LOG Version %s : MQTT Version %s\n", log_time(false), LOG_VERSION,
MQTT_VERSION);
311     fflush(fout);
312     exit(EXIT_FAILURE);
313 }
```

4.1.2.2 ha_ac_off()

```
void ha_ac_off (
    void )
947 {
948     mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
949     E.ac_sw_status = false;
950 }
```

4.1.2.3 ha_ac_on()

```
void ha_ac_on (
    void )
953 {
954     mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
955     E.ac_sw_status = true;
956 }
```

4.1.2.4 ha_dc_off()

```
void ha_dc_off (
    void )
962 {
963     mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
964     E.gti_sw_status = false;
965 }
```

4.1.2.5 ha_dc_on()

```
void ha_dc_on (
    void )
968 {
969     mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
970     E.gti_sw_status = true;
971 }
```

4.1.2.6 log_time()

```
char * log_time (
    bool log )
1032 {
1033     static char time_log[RBUF_SIZ] = {0};
1034     static uint32_t len = 0, sync_time = TIME_SYNC_SEC - 1;
1035     time_t rawtime_log;
1036
1037     tzset();
1038     timezone = 0;
1039     daylight = 0;
1040     time(&rawtime_log);
1041     if (sync_time++ > TIME_SYNC_SEC) {
1042         sync_time = 0;
1043         snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
1044         time commands
1045             mqtt_gti_time(E.client_p, TOPIC_P, time_log);
1046     }
1047     sprintf(time_log, "%s", ctime(&rawtime_log));
1048     len = strlen(time_log);
1049     time_log[len - 1] = 0; // munge out the return character
1050     if (log) {
1051         fprintf(fout, "%s ", time_log);
1052         fflush(fout);
1053     }
1054
1055     return time_log;
1056 }
```

4.1.2.7 log_timer()

```

bool log_timer (
    void )
1091 {
1092     bool itstime = false;
1093
1094     if (E.log_spam < LOW_LOG_SPAM) {
1095         E.log_time_reset = 0;
1096         itstime = true;
1097     }
1098     if (E.log_time_reset > RESET_LOG_SPAM) {
1099         E.log_spam = 0;
1100         itstime = true;
1101     }
1102     return itstime;
1103 }

```

4.1.2.8 main()

```

int main (
    int argc,
    char * argv[] )
322 {
323     struct itimerval new_timer = {
324         .it_value.tv_sec = CMD_SEC,
325         .it_value.tv_usec = 0,
326         .it_interval.tv_sec = CMD_SEC,
327         .it_interval.tv_usec = 0,
328     };
329     struct itimerval old_timer;
330     time_t rawtime;
331     MQTTClient_connectOptions conn_opts_p = MQTTClient_connectOptions_initializer,
332     conn_opts_sd = MQTTClient_connectOptions_initializer,
333     conn_opts_ha = MQTTClient_connectOptions_initializer;
334     MQTTClient_message pubmsg = MQTTClient_message_initializer;
335     MQTTClient_deliveryToken token;
336     char hname[256], *hname_ptr = hname;
337     size_t hname_len = 12;
338
339     gethostname(hname, hname_len);
340     hname[12] = 0;
341     printf("\r\n LOG Version %s : MQTT Version %s : Host Name %s\r\n", LOG_VERSION, MQTT_VERSION,
hname);
342     showIP();
343     skeleton_daemon();
344
345     while (true) {
346         switch (E.mode.E) {
347             case E_INIT:
348
349 #ifdef LOG_TO_FILE
350                 fout = fopen(LOG_TO_FILE, "a");
351                 if (fout == NULL) {
352                     fout = fopen(LOG_TO_FILE_ALT, "a");
353                     if (fout == NULL) {
354                         fout = stdout;
355                         printf("\r\n%s Unable to open LOG file %s \r\n", log_time(false), LOG_TO_FILE_ALT);
356                     }
357                 }
358 #else
359                 fout = stdout;
360 #endif
361                 fprintf(fout, "\r\n%s LOG Version %s : MQTT Version %s\r\n", log_time(false), LOG_VERSION,
MQTT_VERSION);
362                 fflush(fout);
363
364                 if (!bsoc_init()) {
365                     fprintf(fout, "\r\n%s bsoc_init failure \r\n", log_time(false));
366                     fflush(fout);
367                     exit(EXIT_FAILURE);
368                 }
369                 /*
370 * set the timer for MQTT publishing sample speed
371 * CMD_SEC          10
372 */

```

```

373         settimer(ITIMER_REAL, &new_timer, &old_timer);
374         signal(SIGALRM, timer_callback);
375
376         if (strncmp(hname, TNAME, 6) == 0) {
377             MQTTClient_create(&E.client_p, LADDRESS, CLIENTID1,
378                 MQTTCLIENT_PERSISTENCE_NONE, NULL);
379             conn_opts_p.keepAliveInterval = 20;
380             conn_opts_p.cleansession = 1;
381             hname_ptr = LADDRESS;
382         } else {
383             MQTTClient_create(&E.client_p, ADDRESS, CLIENTID1,
384                 MQTTCLIENT_PERSISTENCE_NONE, NULL);
385             conn_opts_p.keepAliveInterval = 20;
386             conn_opts_p.cleansession = 1;
387             hname_ptr = ADDRESS;
388         }
389
390         fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID1);
391         fflush(fout);
392         MQTTClient_setCallbacks(E.client_p, &ha_flag_vars_ss, connlost, msgarrvd, delivered);
393         if ((E.rc = MQTTClient_connect(E.client_p, &conn_opts_p)) != MQTTCLIENT_SUCCESS) {
394             fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
395                 log_time(false), E.rc, hname_ptr, CLIENTID1);
396             fflush(fout);
397             pthread_mutex_destroy(&E.ha_lock);
398             exit(EXIT_FAILURE);
399         }
400
401         if (strncmp(hname, TNAME, 6) == 0) {
402             MQTTClient_create(&E.client_sd, LADDRESS, CLIENTID2,
403                 MQTTCLIENT_PERSISTENCE_NONE, NULL);
404             conn_opts_sd.keepAliveInterval = 20;
405             conn_opts_sd.cleansession = 1;
406             hname_ptr = LADDRESS;
407         } else {
408             MQTTClient_create(&E.client_sd, ADDRESS, CLIENTID2,
409                 MQTTCLIENT_PERSISTENCE_NONE, NULL);
410             conn_opts_sd.keepAliveInterval = 20;
411             conn_opts_sd.cleansession = 1;
412             hname_ptr = ADDRESS;
413         }
414
415         fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID2);
416         fflush(fout);
417         MQTTClient_setCallbacks(E.client_sd, &ha_flag_vars_sd, connlost, msgarrvd, delivered);
418         if ((E.rc = MQTTClient_connect(E.client_sd, &conn_opts_sd)) != MQTTCLIENT_SUCCESS) {
419             fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
420                 log_time(false), E.rc, hname_ptr, CLIENTID2);
421             fflush(fout);
422             pthread_mutex_destroy(&E.ha_lock);
423             exit(EXIT_FAILURE);
424         }
425
426         /*
427         * Home Assistant MQTT receive messages
428         */
429         if (strncmp(hname, TNAME, 6) == 0) {
430             MQTTClient_create(&E.client_ha, LADDRESS, CLIENTID3,
431                 MQTTCLIENT_PERSISTENCE_NONE, NULL);
432             conn_opts_ha.keepAliveInterval = 20;
433             conn_opts_ha.cleansession = 1;
434             hname_ptr = LADDRESS;
435         } else {
436             MQTTClient_create(&E.client_ha, ADDRESS, CLIENTID3,
437                 MQTTCLIENT_PERSISTENCE_NONE, NULL);
438             conn_opts_ha.keepAliveInterval = 20;
439             conn_opts_ha.cleansession = 1;
440             hname_ptr = ADDRESS;
441         }
442
443         fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID3);
444         fflush(fout);
445         MQTTClient_setCallbacks(E.client_ha, &ha_flag_vars_ha, connlost, msgarrvd, delivered);
446         if ((E.rc = MQTTClient_connect(E.client_ha, &conn_opts_ha)) != MQTTCLIENT_SUCCESS) {
447             fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
448                 log_time(false), E.rc, hname_ptr, CLIENTID3);
449             fflush(fout);
450             pthread_mutex_destroy(&E.ha_lock);
451             exit(EXIT_FAILURE);
452         }
453
454         /*
455         * on topic received data will trigger the msgarrvd function
456         */
457         MQTTClient_subscribe(E.client_p, TOPIC_SS, QOS); // FM80 Q84
458         MQTTClient_subscribe(E.client_sd, TOPIC_SD, QOS); // DUMPLoad K42
459         MQTTClient_subscribe(E.client_ha, TOPIC_HA, QOS); // Home Assistant Linux AMD64 and ARM64

```

```

457     pubmsg.payload = "online";
458     pubmsg.payloadlen = strlen("online");
459     pubmsg.qos = QOS;
460     pubmsg.retained = 0;
461     ha_flag_vars_ss.deliveredtoken = 0;
462     // notify HA we are running and controlling AC power plugs
463     MQTTClient_publishMessage(E.client_p, TOPIC_PACA, &pubmsg, &token);
464     MQTTClient_publishMessage(E.client_p, TOPIC_PDCA, &pubmsg, &token);
465
466     // sync HA power switches
467     mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
468     mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
469     mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
470     mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
471     mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
472     mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
473
474     E.ac_sw_on = true; // can be switched on once
475     E.gti_sw_on = true; // can be switched on once
476
477     /*
478     * use libcurl to read AC power meter HTTP data
479     * iammeter connected for split single phase monitoring and one leg GTI power exporting
480     */
481     iammeter_read();
482
483     /*
484     * start the main energy monitoring loop
485     */
486     fprintf(fout, "\r\n%s Solar Energy AC power controller\r\n", log_time(false));
487
488     #ifdef FAKE_VPV
489     fprintf(fout, "\r\n Faking dumpload PV voltage\r\n");
490     #endif
491     ha_flag_vars_ss.energy_mode = NORM_MODE;
492     E.mode.E = E_WAIT;
493     break;
494 case E_WAIT:
495     if (ha_flag_vars_ss.runner || E.speed_go++ > 1500000) {
496         E.speed_go = 0;
497         ha_flag_vars_ss.runner = false;
498         E.mode.E = E_RUN;
499     }
500
501     usleep(100);
502     /*
503     * main state-machine update sequence
504     */
505     bsoc_data_collect();
506     if (!sanity_check()) {
507         fprintf(fout, "\r\n%s Sanity Check error %d %s \r\n", log_time(false), E.sane,
508             mqtt_name[E.sane]);
509         fflush(fout);
510     }
511
512     /*
513     * stop and restart the energy control processing
514     * from inside the program or from a remote Home Assistant command
515     */
516     if (solar_shutdown()) {
517         if (!E.startup) {
518             fprintf(fout, "%s SHUTDOWN Solar Energy Control ---> \r\n", log_time(false));
519         }
520         fflush(fout);
521         ramp_down_gti(E.client_p, true);
522         usleep(100000); // wait
523         ramp_down_ac(E.client_p, true);
524         usleep(100000); // wait
525         ramp_down_gti(E.client_p, true);
526         usleep(100000); // wait
527         ramp_down_ac(E.client_p, true);
528         usleep(100000); // wait
529         if (!E.startup) {
530             fprintf(fout, "%s Completed SHUTDOWN, Press again to RESTART.\r\n",
531                 log_time(false));
532             fflush(fout);
533             fflush(fout);
534
535             uint8_t iam_delay = 0;
536             while (solar_shutdown()) {
537                 mqtt_ha_shutdown(E.client_p, TOPIC_SHUTDOWN);
538                 usleep(USEC_SEC); // wait
539                 if ((int32_t) E.mvar[V_HACSW]) {
540                     ha_ac_off();
541                 }

```

```

542         if ((int32_t) E.mvar[V_HDCSW]) {
543             ha_dc_off();
544         }
545         if ((iam_delay++ > IAM_DELAY) && E.link.shutdown) {
546             E.fm80 = true;
547             E.dumpload = true;
548             E.iammeter = true;
549             E.homeassistant = true;
550         }
551     }
552     E.link.shutdown = 0;
553     fprintf(fout, "%s RESTART Solar Energy Control\r\n", log_time(false));
554     fflush(fout);
555     bsoc_set_mode(E.mode.pv_bias, true, true);
556     E.dl_excess = true;
557     mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 1); // zero power at startup
558     E.dl_excess = false;
559 #ifdef AUTO_CHARGE
560     mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
561 #endif
562     usleep(100000); // wait
563     E.gti_sw_status = true;
564     ResetPI(&E.mode.pid);
565     ha_flag_vars_ss.runner = true;
566     E.fm80 = true;
567     E.dumpload = true;
568     E.iammeter = true;
569     E.homeassistant = true;
570     E.mode.in_pid_control = false; // shutdown auto energy control
571     E.mode.R = R_INIT;
572 }
573 if (ha_flag_vars_ss.receive_token) {
574     ha_flag_vars_ss.receive_token = false;
575 }
576 if (ha_flag_vars_sd.receive_token) {
577     ha_flag_vars_sd.receive_token = false;
578 }
579 break;
580 case E_RUN:
581     usleep(100);
582     switch (E.mode.R) {
583     case R_INIT:
584         E.once_ac = true;
585         E.once_gti = true;
586         E.ac_sw_on = true;
587         E.gti_sw_on = true;
588         E.mode.R = R_RUN;
589         E.mode.no_float = true;
590         break;
591     case R_FLOAT:
592         if (E.mode.no_float) {
593             E.once_ac = true;
594             E.once_gti = true;
595             E.ac_sw_on = true;
596             E.gti_sw_on = true;
597             E.gti_sw_status = false;
598             E.ac_sw_status = false;
599             E.mode.no_float = false;
600         }
601         if (!E.gti_sw_status) {
602             if (gti_test() > MIN_BAT_KW_GTI_HI) {
603                 mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
604                 E.gti_sw_status = true;
605                 fprintf(fout, "%s R_FLOAT DC switch true \r\n", log_time(false));
606             }
607         }
608         usleep(100000); // wait
609         if (!E.ac_sw_status) {
610             if (ac_test() > MIN_BAT_KW_AC_HI) {
611                 mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
612                 E.ac_sw_status = true;
613                 fprintf(fout, "%s R_FLOAT AC switch true \r\n", log_time(false));
614             }
615         }
616         E.mode.pv_bias = PV_BIAS;
617         fm80_float(true);
618         break;
619     case R_RUN:
620     default:
621         E.mode.R = R_RUN;
622         E.mode.no_float = true;
623         break;
624     }
625     /*
626     * main state-machine update sequence and control logic
627     */
628     /*

```

```

629 * check for idle/data errors flags from sensors and HA
630 */
631     if (!E.mode.data_error) {
632         bsoc_set_mode(E.mode.pv_bias, true, false);
633         if (E.gti_delay++ >= GTI_DELAY) {
634             char gti_str[SBUF_SIZ];
635             int32_t error_drive;
636
637             /*
638 * reset the control mode from simple switched power to PID control
639 */
640             if (!E.mode.in_pid_control) {
641                 mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
642                 E.gti_sw_status = true;
643                 usleep(100000); // wait
644                 mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
645                 E.ac_sw_status = true;
646                 E.mode.pv_bias = PV_BIAS;
647                 fprintf(fout, "%s in_pid_mode AC/DC switch true \r\n", log_time(false));
648                 fm80_float(true);
649             } else {
650                 if (!fm80_float(true)) {
651                     E.mode.pv_bias = (int32_t) E.mode.error - PV_BIAS;
652                 }
653             }
654             /*
655 * use PID style set-point error correction
656 */
657             E.mode.in_pid_control = true;
658             E.gti_delay = 0;
659             /*
660 * adjust power balance if battery charging energy is low
661 */
662             if (E.mvar[V_DPBAT] > PV_DL_BIAS_RATE) {
663                 error_drive = (int32_t) E.mode.error - E.mode.pv_bias; // PI feedback control
664             } else {
665                 error_drive = (int32_t) E.mode.error - PV_BIAS_RATE;
666             }
667             /*
668 * when main battery is in float, crank-up the power draw from the solar panels
669 */
670             if (fm80_float(true)) {
671                 error_drive = (int32_t) (E.mode.error + PV_BIAS);
672             }
673             /*
674 * don't drive to zero power
675 */
676             if (error_drive < 0) {
677                 error_drive = PV_BIAS_LOW; // control wide power swings
678                 if (!fm80_sleep()) { // check for using sleep bias
679                     if ((E.mvar[V_FBEKW] > MIN_BAT_KW_BSOC_SLP) && (E.mvar[V_PWA] > PWA_SLEEP))
680                         error_drive = PV_BIAS_SLEEP; // use higher power when we still have sun
681                     for better inverter efficiency
682                 }
683             }
684             /*
685 * reduce charging/diversion power to safe PS limits
686 */
687             if (E.mode.dl_mqtt_max > PV_DL_MPTT_MAX) {
688                 if (!E.dl_excess) {
689                     error_drive = PV_DL_MPTT_IDLE;
690                 } else {
691                     if (E.mode.dl_mqtt_max > PV_DL_MPTT_EXCESS) {
692                         error_drive = PV_DL_MPTT_IDLE;
693                     }
694                 }
695             } else {
696                 if (E.dl_excess) {
697                     error_drive = PV_DL_EXCESS + E.dl_excess_adj;
698                 }
699             }
700             /*
701 * shutdown GTI power at low DL battery Ah or Voltage
702 */
703             if ((E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) || (E.mvar[V_DVBAT] < PV_DL_B_V_LOW)) {
704                 error_drive = PV_BIAS_ZERO;
705             }
706             snprintf(gti_str, SBUF_SIZ - 1, "V%04dX", error_drive); // format for dumpload
707             controller gti power commands
708             mqtt_gti_power(E.client_p, TOPIC_P, gti_str, 2);
709
710         }
711     }

```



```

712
713 #ifndef FAKE_VPV
714     if (fm80_float(true) || ((acl_filter(E.mvar[V_BEN]) > BAL_MAX_ENERGY_AC) && (ac_test() >
715         MIN_BAT_KW_AC_HI))) {
716         ramp_up_ac(E.client_p, E.ac_sw_on); // use once control
717 #ifdef PSW_DEBUG
718         fprintf(fout, "%s MIN_BAT_KW_AC_HI AC switch %d \r\n", log_time(false), E.ac_sw_on);
719 #endif
720         E.ac_sw_on = false; // once flag
721     }
722 #endif
723     if (((ac2_filter(E.mvar[V_BEN]) < BAL_MIN_ENERGY_AC) || ((ac_test() < (MIN_BAT_KW_AC_LO
724         + E.ac_low_adj)))) {
725         if (!fm80_float(true)) {
726             ramp_down_ac(E.client_p, E.ac_sw_on);
727             if (log_timer()) {
728                 fprintf(fout, "%s RAMP DOWN AC, MIN_BAT_KW_AC_LO AC switch %d \r\n",
729                     log_time(false), E.ac_sw_on);
730             }
731         }
732         E.ac_sw_on = true;
733     }
734     /*
735     * Dump Load Excess testing
736     * send excess power into the home power grid taking care not to export energy to the utility grid
737     */
738     if (((dcl_filter(E.mvar[V_BEN]) > BAL_MAX_ENERGY_GTI) && (gti_test() >
739         MIN_BAT_KW_GTI_HI)) || E.dl_excess) {
740 #ifndef FAKE_VPV
741 #ifdef B_DLE_DEBUG
742         if (E.dl_excess) {
743             fprintf(fout, "%s DL excess ramp_up_gti, DC switch %d\r\n", log_time(false),
744                 E.gti_sw_on);
745         }
746 #endif
747         ramp_up_gti(E.client_p, E.gti_sw_on, E.dl_excess);
748         if (log_timer()) {
749             fprintf(fout, "%s RAMP DOWN DC, MIN_BAT_KW_GTI_HI DC switch %d \r\n",
750                 log_time(false), E.gti_sw_on);
751         }
752         E.gti_sw_on = false; // once flag
753     }
754 #endif
755     } else {
756         if ((dc2_filter(E.mvar[V_BEN]) < BAL_MIN_ENERGY_GTI) || (gti_test() <
757             (MIN_BAT_KW_GTI_LO + E.gti_low_adj))) {
758             if (!E.dl_excess) {
759                 if (log_timer()) {
760                     ramp_down_gti(E.client_p, true);
761                     #ifdef PSW_DEBUG
762                     fprintf(fout, "%s MIN_BAT_KW_GTI_LO DC switch %d \r\n", log_time(false),
763                         E.gti_sw_on);
764                     #endif
765                 }
766                 E.gti_sw_on = true;
767             }
768         }
769     }
770 };
771 #ifdef B_ADJ_DEBUG
772     fprintf(fout, "\r\n LO ADJ: AC %8.2fWh, GTI %8.2fWh\r\n", MIN_BAT_KW_AC_LO + E.ac_low_adj,
773         MIN_BAT_KW_GTI_LO + E.gti_low_adj);
774 #endif
775 #ifdef B_DLE_DEBUG
776     if (E.dl_excess) {
777         fprintf(fout, "%s DL excess vars from ha_energy %d %d : Flag %d\r\n", log_time(false),
778             E.mode.con4, E.mode.con5, E.dl_excess);
779     }
780 #endif
781
782     time(&rawtime);
783
784     if (E.im_delay++ >= IM_DELAY) {
785         E.im_delay = 0;
786         iammeter_read();
787     }
788     if (E.im_display++ >= IM_DISPLAY) {
789         char buffer[SYSLOG_SIZ];
790         uint32_t len;
791
792         E.im_display = 0;
793         mqttt_ha_pid(E.client_p, TOPIC_PPID);
794         if (!E.fm80 && E.dumpload && E.iammeter) {
795             if (!E.iammeter) {
796                 E.link.iammeter_error++;
797             }
798         }
799     }

```

```

789         } else {
790             E.link.mqtt_error++;
791         }
792         E.link.shutdown++;
793         fprintf(fout, "\r\n%s !!!! Source data update error !!!! , check FM80 %i, DUMPLOAD
%i, IAMMETER %i channels M %u,%u I %u,%u\r\n", log_time(false), E.fm80, E.dumpload, E.fm80,
794             E.link.mqtt_count, E.link.mqtt_error, E.link.iammeter_count,
E.link.iammeter_error);
795         fflush(fout);
796         snprintf(buffer, SYSLOG_SIZ - 1, "\r\n%s !!!! Source data update error !!!! ,
check FM80 %i, DUMPLOAD %i, IAMMETER %i channels M %u,%u I %u,%u\r\n", log_time(false), E.fm80,
E.dumpload, E.fm80,
797             E.link.mqtt_count, E.link.mqtt_error, E.link.iammeter_count,
E.link.iammeter_error);
798         syslog(LOG_NOTICE, buffer);
799         mqtt_ha_shutdown(E.client_p, TOPIC_SHUTDOWN);
800         E.mode.data_error = true;
801     } else {
802         E.mode.data_error = false;
803         E.link.shutdown = 0;
804     }
805     snprintf(buffer, RBUF_SIZ - 1, "%s", ctime(&rawtime));
806     len = strlen(buffer);
807     buffer[len - 1] = 0; // munge out the return character
808     fprintf(fout, "%s ", buffer);
809     fflush(fout);
810     E.fm80 = false;
811     E.dumpload = false;
812     E.homeassistant = false;
813     E.iammeter = false;
814     sync_ha();
815     print_im_vars();
816     print_mvar_vars();
817     fprintf(fout, "%s\r", ctime(&rawtime));
818 }
819 E.mode.E = E_WAIT;
820 fflush(fout);
821 if (E.mode.con6) {
822     E.mode.R = R_IDLE;
823 }
824 if (E.mode.con7) {
825     E.mode.E = E_STOP;
826 }
827 break;
828 case E_STOP:
829 default:
830     fflush(fout);
831     fprintf(fout, "\r\n%s HA Energy stopped and exited.\r\n", log_time(false));
832     fflush(fout);
833     return 0;
834     break;
835 }
836 }
837 }

```

4.1.2.9 ramp_down_ac()

```

void ramp_down_ac (
    MQTTClient client_p,
    bool sw_off )
937 {
938     if (sw_off) {
939         mqtt_ha_switch(client_p, TOPIC_PACC, false);
940         E.ac_sw_status = false;
941         usleep(500000);
942     }
943     E.once_ac = true;
944 }

```

4.1.2.10 ramp_down_gti()

```
void ramp_down_gti (
    MQTTClient client_p,
    bool sw_off )
904 {
905     if (sw_off) {
906         mqtt_ha_switch(client_p, TOPIC_PDCC, false);
907         E.once_gti_zero = true;
908         E.gti_sw_status = false;
909     }
910     E.once_gti = true;
911
912     if (E.once_gti_zero) {
913         mqtt_gti_power(client_p, TOPIC_P, "Z#", 7); // zero power
914         E.once_gti_zero = false;
915     }
916 }
```

4.1.2.11 ramp_up_ac()

```
void ramp_up_ac (
    MQTTClient client_p,
    bool start )
922 {
923
924     if (start) {
925         E.once_ac = true;
926     }
927
928     if (E.once_ac) {
929         E.once_ac = false;
930         mqtt_ha_switch(client_p, TOPIC_PACC, true);
931         E.ac_sw_status = true;
932         usleep(500000); // wait for voltage to ramp
933     }
934 }
```

4.1.2.12 ramp_up_gti()

```
void ramp_up_gti (
    MQTTClient client_p,
    bool start,
    bool excess )
843 {
844     static uint32_t sequence = 0;
845
846     if (start) {
847         E.once_gti = true;
848     }
849
850     if (E.once_gti) {
851         E.once_gti = false;
852         sequence = 0;
853         if (!excess) {
854             mqtt_ha_switch(client_p, TOPIC_PDCC, true);
855             E.gti_sw_status = true;
856             usleep(500000); // wait for voltage to ramp
857         } else {
858             sequence = 1;
859         }
860     }
861
862     switch (sequence) {
863     case 4:
864         E.once_gti_zero = true;
865         break;
```

```

866     case 3:
867     case 2:
868     case 1:
869         E.once_gti_zero = true;
870         if (bat_current_stable() || E.dl_excess) { // check battery current std dev, stop 'motorboating'
871             sequence++;
872             if (!mqtt_gti_power(client_p, TOPIC_P, "+#", 3)) {
873                 sequence = 0;
874             }; // +100W power
875         } else {
876             usleep(500000); // wait a bit more for power to be stable
877             sequence = 1; // do power ramps when ready
878             if (!mqtt_gti_power(client_p, TOPIC_P, "-#", 4)) {
879                 sequence = 0;
880             }; // - 100W power
881         }
882         break;
883     case 0:
884         sequence++;
885         if (E.once_gti_zero) {
886             mqtt_gti_power(client_p, TOPIC_P, "Z#", 5); // zero power
887             E.once_gti_zero = false;
888         }
889         break;
890     default:
891         if (E.once_gti_zero) {
892             mqtt_gti_power(client_p, TOPIC_P, "Z#", 6); // zero power
893             E.once_gti_zero = false;
894         }
895         sequence = 0;
896         break;
897     }
898 }

```

4.1.2.13 sanity_check()

```

bool sanity_check (
    void )
254 {
255     if (E.mvar[V_PWA] > PWA_SANE) {
256         E.sane = S_PWA;
257         return false;
258     }
259     if (E.mvar[V_PAMPS] > PAMPS_SANE) {
260         E.sane = S_PAMPS;
261         return false;
262     }
263     if (E.mvar[V_PVOLTS] > PVOLTS_SANE) {
264         E.sane = S_PVOLTS;
265         return false;
266     }
267     if (E.mvar[V_FBAMPS] > BAMPS_SANE) {
268         E.sane = S_FBAMPS;
269         return false;
270     }
271     return true;
272 }

```

4.1.2.14 showIP()

```

void showIP (
    void )
160 {
161     struct ifaddrs *ifaddr, *ifa;
162     int s;
163     char host[NI_MAXHOST];
164
165     if (getifaddrs(&ifaddr) == -1) {
166         perror("getifaddrs");
167         exit(EXIT_FAILURE);
168     }

```

```

169
170
171     for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
172         if (ifa->ifa_addr == NULL)
173             continue;
174
175         s = getnameinfo(ifa->ifa_addr, sizeof(struct sockaddr_in), host, NI_MAXHOST, NULL, 0,
NI_NUMERICHOST);
176
177         if (ifa->ifa_addr->sa_family == AF_INET) {
178             if (s != 0) {
179                 exit(EXIT_FAILURE);
180             }
181             printf("\tInterface : <%s>\n", ifa->ifa_name);
182             printf("\t  Address : <%s>\n", host);
183         }
184     }
185
186     freeifaddrs(ifaddr);
187 }

```

4.1.2.15 skeleton_daemon()

```

static void skeleton_daemon ( ) [static]
194 {
195     pid_t pid;
196
197     /* Fork off the parent process */
198     pid = fork();
199
200     /* An error occurred */
201     if (pid < 0) {
202         printf("\r\n%sDAEMON failure LOG Version %s : MQTT Version %s\r\n", log_time(false),
LOG_VERSION, MQTT_VERSION);
203         exit(EXIT_FAILURE);
204     }
205
206     /* Success: Let the parent terminate */
207     if (pid > 0) {
208         exit(EXIT_SUCCESS);
209     }
210
211     /* On success: The child process becomes session leader */
212     if (setsid() < 0) {
213         exit(EXIT_FAILURE);
214     }
215
216     /* Catch, ignore and handle signals */
217     /*TODO: Implement a working signal handler */
218     // signal(SIGCHLD, SIG_IGN);
219     // signal(SIGHUP, SIG_IGN);
220
221     /* Fork off for the second time*/
222     pid = fork();
223
224     /* An error occurred */
225     if (pid < 0) {
226         exit(EXIT_FAILURE);
227     }
228
229     /* Success: Let the parent terminate */
230     if (pid > 0) {
231         exit(EXIT_SUCCESS);
232     }
233
234     /* Set new file permissions */
235     umask(0);
236
237     /* Change the working directory to the root directory */
238     /* or another appropriated directory */
239     chdir("/");
240
241     /* Close all open file descriptors */
242     int x;
243     for (x = sysconf(_SC_OPEN_MAX); x >= 0; x--) {
244         close(x);
245     }
246
247 }

```

4.1.2.16 solar_shutdown()

```

static bool solar_shutdown (
    void ) [static]
977 {
978     static bool ret = false;
979
980     if (E.startup) {
981         ret = true;
982         E.startup = false;
983         return ret;
984     } else {
985         ret = false;
986
987         /*
988      * FIXME
989      *
990      */
991     }
992
993     if (E.solar_shutdown) {
994         ret = true;
995     } else {
996         ret = false;
997     }
998
999     if ((E.mvar[V_FBEKW] < BAT_CRITICAL) && !E.startup) { // special case for low battery
1000         if (!E.mode.bat_crit) {
1001             ret = true;
1002 #ifdef CRITICAL_SHUTDOWN_LOG
1003             fprintf(fout, "%s Solar BATTERY CRITICAL shutdown comms check ret = %d \r\n",
1004                 log_time(false), ret);
1005             fflush(fout);
1006 #endif
1007             E.mode.bat_crit = true;
1008             return ret;
1009         } else {
1010             E.mode.bat_crit = false;
1011         }
1012
1013         if (E.link.shutdown >= MAX_ERROR) {
1014             ret = true;
1015             if (E.fm80 && E.dumpload && E.iammeter) {
1016                 ret = false;
1017                 E.link.shutdown = 0;
1018             }
1019
1020 #ifdef DEBUG_SHUTDOWN
1021             fprintf(fout, "%s Solar shutdown comms check ret = %d \r\n", log_time(false), ret);
1022             fflush(fout);
1023 #endif
1024         }
1025         return ret;
1026     }

```

4.1.2.17 sync_ha()

```

bool sync_ha (
    void )
1062 {
1063     bool sync = false;
1064     if (E.gti_sw_status != (bool) ((int32_t) E.mvar[V_HDCSW])) {
1065         fprintf(fout, "DC_MM %d %d ", (bool) E.gti_sw_status, (bool) ((int32_t) E.mvar[V_HDCSW]));
1066         mqtt_ha_switch(E.client_p, TOPIC_PDCC, !E.gti_sw_status);
1067         E.dc_mismatch = true;
1068         fflush(fout);
1069         sync = true;
1070     } else {
1071         E.dc_mismatch = false;
1072     }
1073
1074     E.ac_sw_status = (bool) ((int32_t) E.mvar[V_HACSW]); // TEMP FIX for Mismatch errors
1075     if (E.ac_sw_status != (bool) ((int32_t) E.mvar[V_HACSW])) {
1076         fprintf(fout, "AC_MM %d %d ", (bool) E.ac_sw_status, (bool) ((int32_t) E.mvar[V_HACSW]));
1077         mqtt_ha_switch(E.client_p, TOPIC_PACC, !E.ac_sw_status);
1078         E.ac_mismatch = true;

```

```
1079         fflush(fout);
1080         sync = true;
1081     } else {
1082         E.ac_mismatch = false;
1083     }
1084     return sync;
1085 }
```

4.1.2.18 timer_callback()

```
void timer_callback (
    int32_t signum )
283 {
284     signal(signum, timer_callback);
285     ha_flag_vars_ss.runner = true;
286     E.ten_sec_clock++;
287     E.log_spam++;
288     E.log_time_reset++;
289     if (E.log_spam > MAX_LOG_SPAM) {
290         E.log_spam = 0;
291     }
292 }
```

4.1.3 Variable Documentation

4.1.3.1 board_name

```
const char* board_name = "NO_BOARD"
```

4.1.3.2 driver_name

```
const char * driver_name = "NO_DRIVER"
```

4.1.3.3 E

```
struct energy_type E
```

4.1.3.4 fout

```
FILE* fout
```

4.1.3.5 ha_flag_vars_ha

```
struct ha_flag_type ha_flag_vars_ha
```

Initial value:

```
= {  
    .runner = false,  
    .receivedtoken = false,  
    .deliveredtoken = false,  
    .rec_ok = false,  
    .ha_id = HA_ID,  
    .var_update = 0,  
}
```

4.1.3.6 ha_flag_vars_pc

```
struct ha_flag_type ha_flag_vars_pc
```

Initial value:

```
= {  
    .runner = false,  
    .receivedtoken = false,  
    .deliveredtoken = false,  
    .rec_ok = false,  
    .ha_id = P8055_ID,  
    .var_update = 0,  
}
```

4.1.3.7 ha_flag_vars_sd

```
struct ha_flag_type ha_flag_vars_sd
```

Initial value:

```
= {  
    .runner = false,  
    .receivedtoken = false,  
    .deliveredtoken = false,  
    .rec_ok = false,  
    .ha_id = DUMPLoad_ID,  
    .var_update = 0,  
}
```

4.1.3.8 ha_flag_vars_ss

```
struct ha_flag_type ha_flag_vars_ss
```

Initial value:

```
= {  
    .runner = false,  
    .receivedtoken = false,  
    .deliveredtoken = false,  
    .rec_ok = false,  
    .ha_id = FM80_ID,  
    .var_update = 0,  
    .energy_mode = NORM_MODE,  
}
```


4.3.1.1 ac0_filter()

```
double ac0_filter (
    const double raw )
376 {
377     static double accum = 0.0f;
378     static double coef = COEFF;
379     accum = accum - accum / coef + raw;
380     return accum / coef;
381 }
```

4.3.1.2 ac1_filter()

```
double ac1_filter (
    const double raw )
384 {
385     static double accum = 0.0f;
386     static double coef = COEF;
387     accum = accum - accum / coef + raw;
388     return accum / coef;
389 }
```

4.3.1.3 ac2_filter()

```
double ac2_filter (
    const double raw )
392 {
393     static double accum = 0.0f;
394     static double coef = COEF;
395     accum = accum - accum / coef + raw;
396     return accum / coef;
397 }
```

4.3.1.4 ac_test()

```
double ac_test (
    void )
191 {
192     return ac0_filter(L.ac_weight);
193 }
```

4.3.1.5 bat_current_stable()

```
bool bat_current_stable (
    void )
240 {
241     static double gap = 0.0f;
242
243     if (L.batc_std_dev <= (MAX_BATC_DEV + gap)) {
244         gap = MAX_BATC_DEV;
245         if (L.bat_c_std_dev[0] < BAT_C_DRAW) {
246             return true;
247         } else {
248             gap = 0.0f;
249             return false;
250         }
251     } else {
252         gap = 0.0f;
253         return false;
254     }
255 }
```

4.3.1.6 bsoc_ac()

```
double bsoc_ac (
    void )

136 {
137
138     return ac0_filter(L.ac_weight);
139 };
```

4.3.1.7 bsoc_data_collect()

```
bool bsoc_data_collect (
    void )

86 {
87     bool ret = false;
88     static uint32_t i = 0;
89     // lockout threaded updates
90     pthread_mutex_lock(&E.ha_lock); // lockout MQTT var updates
91
92     L.ac_weight = E.mvar[V_FBEKW];
93     L.gti_weight = E.mvar[V_FBEKW];
94 #ifndef FAKE_VPV // no DUMPLoad AC charger
95     if (E.gti_sw_on) {
96         pv_voltage = PV_V_NOM;
97     } else {
98         pv_voltage = PV_V_FAKE;
99     }
100     E.mvar[V_DVPV] = pv_voltage;
101 #else
102     L.pv_voltage = E.mvar[V_DVPV];
103 #endif
104     L.bat_voltage = E.mvar[V_DVBAT];
105     L.bat_current = E.mvar[V_DCMPPPT];
106     E.ac_low_adj = E.mvar[V_FSO] * -0.5f;
107     E.gti_low_adj = E.mvar[V_FACE] * -0.5f;
108     E.mode.dl_mqtt_max = E.mvar[V_DPMPPPT];
109
110     pthread_mutex_unlock(&E.ha_lock); // resume remote MQTT var updates
111
112     if (E.ac_low_adj < -2000.0f) {
113         E.ac_low_adj = -2000.0f;
114     }
115     if (E.gti_low_adj < -2000.0f) {
116         E.gti_low_adj = -2000.0f;
117     }
118
119     L.bat_c_std_dev[i++] = L.bat_current;
120     if (i >= DEV_SIZE) {
121         i = 0;
122     }
123
124     calculateStandardDeviation(DEV_SIZE, L.bat_c_std_dev);
125
126 #ifdef BSOC_DEBUG
127     fprintf(fout, "\r\nmqtt var bsoc update\r\n");
128 #endif
129     return ret;
130 }
```

4.3.1.8 bsoc_gti()

```
double bsoc_gti (
    void )

146 {
147 #ifdef BSOC_DEBUG
148     fprintf(fout, "pvp %f, gweight %f, aweight %f, batv %f, batc %f\r\n", pv_voltage, gti_weight,
149         ac_weight, bat_voltage, bat_current);
149 #endif
```

```

150     // check for 48VDC AC charger powered from the Solar battery bank AC inverter unless E.dl_excess is
    TRUE
151     if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
152         L.gti_weight = 0.0f; // reduce power to zero
153     } else {
154         if (E.dl_excess) {
155             if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
156                 L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
157             } else {
158                 L.gti_weight = 0.0f; // reduce power to zero
159             }
160         }
161     }
162
163
164     return dc0_filter(L.gti_weight);
165 };

```

4.3.1.9 bsoc_init()

```

bool bsoc_init (
    void )
61 {
62     L.ac_weight = 0.0f;
63     L.gti_weight = 0.0f;
64     // use MUTEX locks for message passing between remote programs
65     if (pthread_mutex_init(&E.ha_lock, NULL) != 0) {
66         fprintf(fout, "\n%s mutex init has failed\n", log_time(false));
67         return false;
68     }
69     return true;
70 };

```

4.3.1.10 bsoc_set_mode()

```

bool bsoc_set_mode (
    const double target,
    const bool mode,
    const bool init )
262 {
263     static bool bsoc_mode = false;
264     static bool bsoc_high = false, ha_ac_mode = true;
265     static double accum = 0.0f, vpwa = 0.0f;
266
267     if (init) {
268         bsoc_mode = false;
269         bsoc_high = false;
270         ha_ac_mode = true;
271         accum = 0.0f;
272         vpwa = 0.0f;
273         return true;
274     }
275     /*
276     * running avg filter
277     */
278     accum = accum - accum / COEFN + E.mvar[V_PWA];
279     vpwa = accum / COEFN;
280
281     if ((vpwa >= PV_FULL_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
282         if (!bsoc_mode) {
283             ResetPI(&E.mode.pid);
284         }
285         bsoc_mode = true;
286         bsoc_high = true;
287         if (!ha_ac_mode) {
288             ha_ac_on();
289             ha_ac_mode = true;
290         }
291     }
292     else {

```

```

293         if (bsoc_high) { // turn off at min limit power
294             if ((vpwa >= PV_MIN_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
295                 bsoc_mode = true;
296                 if (ha_ac_mode) {
297                     ha_ac_off();
298                     ha_ac_mode = false;
299                 }
300             } else {
301                 bsoc_high = false;
302                 ha_ac_mode = false;
303             }
304         }
305     }
306
307     E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) + E.mvar[V_DPPV]; // use as a temp variable
308     E.mode.total_system = (E.mvar[V_FLO] - E.mode.gti_dumpload) + E.mvar[V_DPPV] + (E.print_vars[L3_P]*
309     -1.0f);
310     E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) - E.mvar[V_DPPV]; // use this value
311
312     /*
313     * look at system energy balance for power control drive
314     */
315     if (mode) { // add GTI power from dumpload
316         E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + E.mode.gti_dumpload +
317         PBAL_OFFSET);
318     } else {
319         E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + PBAL_OFFSET);
320     }
321
322     if (E.mode.error > 0.0f) {
323         L.coef = COEF;
324     } else {
325         L.coef = COEFN;
326     }
327
328     E.mode.target = target;
329     E.mode.error = round(error_filter(E.mode.error));
330
331     /*
332     * check for idle flag from HA
333     */
334     if (E.mode.con6) {
335         ha_ac_mode = true;
336         bsoc_mode = false;
337     }
338
339     /*
340     * HA start excess button pressed
341     */
342     if (E.mode.con4) {
343         E.dl_excess = true;
344         E.mode.con4 = false;
345     }
346
347     /*
348     * HA stop excess button pressed
349     */
350     if (E.mode.con5) {
351         mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 9); // zero power at excess shutdown
352         E.dl_excess = false;
353         E.mode.con5 = false;
354     }
355
356     /*
357     * DL buffer battery low set-point excess load shutdown
358     */
359     if (E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) {
360         mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 10); // zero power at excess shutdown
361         E.dl_excess = false;
362         E.mode.con4 = false;
363         E.mode.con5 = false;
364     }
365
366     return bsoc_mode;
367 }

```

4.3.1.11 bsoc_set_std_dev()

```

void bsoc_set_std_dev (
    const double value,
    const uint32_t i )

```

```

76 {
77     L.bat_c_std_dev[i] = value;
78 }

```

4.3.1.12 calculateStandardDeviation()

```

double calculateStandardDeviation (
    const uint32_t N,
    const double data[] )
205 {
206     // variable to store sum of the given data
207     double sum = 0;
208
209     for (int i = 0; i < N; i++) {
210         sum += data[i];
211     }
212
213     // calculating mean
214     double mean = sum / N;
215
216     // temporary variable to store the summation of square
217     // of difference between individual data items and mean
218     double values = 0;
219
220     for (int i = 0; i < N; i++) {
221         values += pow(data[i] - mean, 2);
222     }
223
224     // variance is the square of standard deviation
225     double variance = values / N;
226
227     // calculating standard deviation by finding square root
228     // of variance
229     double standardDeviation = sqrt(variance);
230     L.batc_std_dev = standardDeviation;
231
232     #ifdef BSOC_DEBUG
233     // printing standard deviation
234     fprintf(fout, "STD DEV of Current %.2f\r\n", standardDeviation);
235     #endif
236     return standardDeviation;
237 }

```

4.3.1.13 dc0_filter()

```

double dc0_filter (
    const double raw )
400 {
401     static double accum = 0.0f;
402     static double coef = COEFF;
403     accum = accum - accum / coef + raw;
404     return accum / coef;
405 }

```

4.3.1.14 dc1_filter()

```

double dc1_filter (
    const double raw )
408 {
409     static double accum = 0.0f;
410     static double coef = COEF;
411     accum = accum - accum / coef + raw;
412     return accum / coef;
413 }

```

4.3.1.15 dc2_filter()

```
double dc2_filter (
    const double raw )
416 {
417     static double accum = 0.0f;
418     static double coef = COEF;
419     accum = accum - accum / coef + raw;
420     return accum / coef;
421 }
```

4.3.1.16 drive0_filter()

```
double drive0_filter (
    const double raw )
424 {
425     static double accum = 0.0f;
426     static double coef = COEF;
427     accum = accum - accum / coef + raw;
428     return accum / coef;
429 }
```

4.3.1.17 drive1_filter()

```
double drive1_filter (
    const double raw )
432 {
433     static double accum = 0.0f;
434     static double coef = COEFF;
435     accum = accum - accum / coef + raw;
436     return accum / coef;
437 }
```

4.3.1.18 error_filter()

```
static double error_filter (
    const double raw ) [static]
369 {
370     static double accum = 0.0f;
371     accum = accum - accum / L.coef + raw;
372     return accum / L.coef;
373 }
```

4.3.1.19 get_batc_dev()

```
double get_batc_dev (
    void )
196 {
197     return L.batc_std_dev;
198 }
```

4.3.1.20 gti_test()

```
double gti_test (
    void )
171 {
172     // check for 48VDC AC charger powered from the Solar battery bank AC inverter
173     if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
174         L.gti_weight = 0.0f; // reduce power to zero
175 #ifdef BSOC_DEBUG
176         fprintf(fout, "pvp %8.2f, gweight %8.2f, aweight %8.2f, batv %8.2f, batc %8.2f\r\n", pv_voltage,
            gti_weight, ac_weight, bat_voltage, bat_current);
177 #endif
178     } else {
179         if (E.dl_excess) {
180             if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
181                 L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
182             } else {
183                 L.gti_weight = 0.0f; // reduce power to zero
184             }
185         }
186     }
187     return dc0_filter(L.gti_weight);
188 }
```

4.3.2 Variable Documentation

4.3.2.1 L

```
struct local_type L [static]
```

Initial value:

```
= {
    .ac_weight = 0.0f,
    .bat_current = 0.0f,
    .bat_voltage = 0.0f,
    .batc_std_dev = 0.0f,
    .coef = COEF,
    .gti_weight = 0.0f,
    .pv_voltage = 0.0f,
}
```

4.3.2.2 mqtt_name

```
const char* mqtt_name[V_DLAST]
```

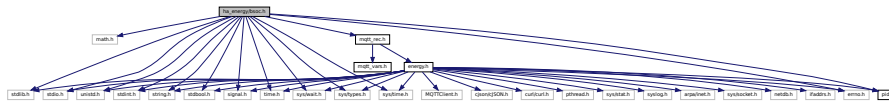
4.4 ha_energy/bsoc.h File Reference

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#include <signal.h>
```

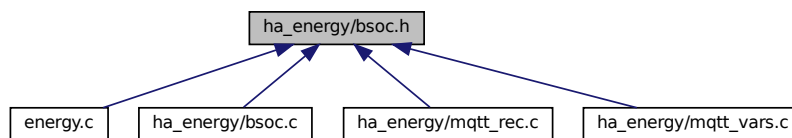


```
#include <time.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/time.h>
#include <errno.h>
#include "pid.h"
#include "mqtt_rec.h"
```

Include dependency graph for bsoc.h:



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



Macros

- #define [MIN_PV_VOLTS](#) 5.0f
- #define [MIN_BAT_VOLTS](#) 23.0f
- #define [MIN_BAT_KW](#) 4100.0f
- #define [DEV_SIZE](#) 10
- #define [MAX_BATC_DEV](#) 1.5f
- #define [BAT_C_DRAW](#) 3.0f
- #define [PBAL_OFFSET](#) -50.0f
- #define [PV_FULL_PWR](#) 300.0f
- #define [PV_MIN_PWR](#) 160.0f
- #define [PV_V_NOM](#) 60.0f
- #define [PV_V_FAKE](#) 0.336699f
- #define [COEF](#) 8.0f
- #define [COEFN](#) 4.0f
- #define [COEFF](#) 2.0f

Functions

- bool [bsoc_init](#) (void)
- bool [bsoc_data_collect](#) (void)
- double [bsoc_ac](#) (void)
- double [bsoc_gti](#) (void)
- double [gti_test](#) (void)
- double [ac_test](#) (void)
- double [get_batc_dev](#) (void)

- bool [bat_current_stable](#) (void)
- void [bsoc_set_std_dev](#) (const double, const uint32_t)
- double [calculateStandardDeviation](#) (const uint32_t, const double *)
- bool [bsoc_set_mode](#) (const double, const bool, const bool)
- double [ac0_filter](#) (const double)
- double [ac1_filter](#) (const double)
- double [ac2_filter](#) (const double)
- double [dc0_filter](#) (const double)
- double [dc1_filter](#) (const double)
- double [dc2_filter](#) (const double)
- double [drive0_filter](#) (const double)
- double [drive1_filter](#) (const double)

4.4.1 Macro Definition Documentation

4.4.1.1 BAT_C_DRAW

```
#define BAT_C_DRAW 3.0f
```

4.4.1.2 COEF

```
#define COEF 8.0f
```

4.4.1.3 COEFF

```
#define COEFF 2.0f
```

4.4.1.4 COEFN

```
#define COEFN 4.0f
```

4.4.1.5 DEV_SIZE

```
#define DEV_SIZE 10
```

4.4.1.6 MAX_BATC_DEV

```
#define MAX_BATC_DEV 1.5f
```

4.4.1.7 MIN_BAT_KW

```
#define MIN_BAT_KW 4100.0f
```

4.4.1.8 MIN_BAT_VOLTS

```
#define MIN_BAT_VOLTS 23.0f
```

4.4.1.9 MIN_PV_VOLTS

```
#define MIN_PV_VOLTS 5.0f
```

4.4.1.10 PBAL_OFFSET

```
#define PBAL_OFFSET -50.0f
```

4.4.1.11 PV_FULL_PWR

```
#define PV_FULL_PWR 300.0f
```

4.4.1.12 PV_MIN_PWR

```
#define PV_MIN_PWR 160.0f
```

4.4.1.13 PV_V_FAKE

```
#define PV_V_FAKE 0.336699f
```

4.4.1.14 PV_V_NOM

```
#define PV_V_NOM 60.0f
```

4.4.2 Function Documentation

4.4.2.1 ac0_filter()

```
double ac0_filter (
    const double raw )
376 {
377     static double accum = 0.0f;
378     static double coef = COEFF;
379     accum = accum - accum / coef + raw;
380     return accum / coef;
381 }
```

4.4.2.2 ac1_filter()

```
double ac1_filter (
    const double raw )
384 {
385     static double accum = 0.0f;
386     static double coef = COEF;
387     accum = accum - accum / coef + raw;
388     return accum / coef;
389 }
```

4.4.2.3 ac2_filter()

```
double ac2_filter (
    const double raw )
392 {
393     static double accum = 0.0f;
394     static double coef = COEF;
395     accum = accum - accum / coef + raw;
396     return accum / coef;
397 }
```

4.4.2.4 ac_test()

```
double ac_test (
    void )
191 {
192     return ac0_filter(L.ac_weight);
193 }
```

4.4.2.5 bat_current_stable()

```

bool bat_current_stable (
    void )
240 {
241     static double gap = 0.0f;
242
243     if (L.batc_std_dev <= (MAX_BATC_DEV + gap)) {
244         gap = MAX_BATC_DEV;
245         if (L.bat_c_std_dev[0] < BAT_C_DRAW) {
246             return true;
247         } else {
248             gap = 0.0f;
249             return false;
250         }
251     } else {
252         gap = 0.0f;
253         return false;
254     }
255 }

```

4.4.2.6 bsoc_ac()

```

double bsoc_ac (
    void )
136 {
137
138     return ac0_filter(L.ac_weight);
139 };

```

4.4.2.7 bsoc_data_collect()

```

bool bsoc_data_collect (
    void )
86 {
87     bool ret = false;
88     static uint32_t i = 0;
89     // lockout threaded updates
90     pthread_mutex_lock(&E.ha_lock); // lockout MQTT var updates
91
92     L.ac_weight = E.mvar[V_FBEKW];
93     L.gti_weight = E.mvar[V_FBEKW];
94 #ifndef FAKE_VPV // no DUMPLoad AC charger
95     if (E.gti_sw_on) {
96         pv_voltage = PV_V_NOM;
97     } else {
98         pv_voltage = PV_V_FAKE;
99     }
100     E.mvar[V_DVPV] = pv_voltage;
101 #else
102     L.pv_voltage = E.mvar[V_DVPV];
103 #endif
104     L.bat_voltage = E.mvar[V_DVBAT];
105     L.bat_current = E.mvar[V_DCMPPPT];
106     E.ac_low_adj = E.mvar[V_FSO] * -0.5f;
107     E.gti_low_adj = E.mvar[V_FACE] * -0.5f;
108     E.mode.dl_mqtt_max = E.mvar[V_DPMPPPT];
109
110     pthread_mutex_unlock(&E.ha_lock); // resume remote MQTT var updates
111
112     if (E.ac_low_adj < -2000.0f) {
113         E.ac_low_adj = -2000.0f;
114     }
115     if (E.gti_low_adj < -2000.0f) {
116         E.gti_low_adj = -2000.0f;
117     }
118
119     L.bat_c_std_dev[i++] = L.bat_current;
120     if (i >= DEV_SIZE) {

```

```

121         i = 0;
122     }
123
124     calculateStandardDeviation(DEV_SIZE, L.bat_c_std_dev);
125
126 #ifdef BSOC_DEBUG
127     fprintf(fout, "\r\nmqtt var bsoc update\r\n");
128 #endif
129     return ret;
130 }

```

4.4.2.8 bsoc_gti()

```

double bsoc_gti (
    void )
146 {
147 #ifdef BSOC_DEBUG
148     fprintf(fout, "pvp %f, gweight %f, aweight %f, batv %f, batc %f\r\n", pv_voltage, gti_weight,
149         ac_weight, bat_voltage, bat_current);
149 #endif
150     // check for 48VDC AC charger powered from the Solar battery bank AC inverter unless E.dl_excess is
    TRUE
151     if ((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
152         L.gti_weight = 0.0f; // reduce power to zero
153     } else {
154         if (E.dl_excess) {
155             if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
156                 L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
157             } else {
158                 L.gti_weight = 0.0f; // reduce power to zero
159             }
160         }
161     }
162
163     return dc0_filter(L.gti_weight);
164 }
165 };

```

4.4.2.9 bsoc_init()

```

bool bsoc_init (
    void )
61 {
62     L.ac_weight = 0.0f;
63     L.gti_weight = 0.0f;
64     // use MUTEX locks for message passing between remote programs
65     if (pthread_mutex_init(&E.ha_lock, NULL) != 0) {
66         fprintf(fout, "\n%s mutex init has failed\n", log_time(false));
67         return false;
68     }
69     return true;
70 };

```

4.4.2.10 bsoc_set_mode()

```

bool bsoc_set_mode (
    const double target,
    const bool mode,
    const bool init )
262 {
263     static bool bsoc_mode = false;
264     static bool bsoc_high = false, ha_ac_mode = true;

```

```

265     static double accum = 0.0f, vpwa = 0.0f;
266
267     if (init) {
268         bsoc_mode = false;
269         bsoc_high = false;
270         ha_ac_mode = true;
271         accum = 0.0f;
272         vpwa = 0.0f;
273         return true;
274     }
275     /*
276     * running avg filter
277     */
278     accum = accum - accum / COEFN + E.mvar[V_PWA];
279     vpwa = accum / COEFN;
280
281     if ((vpwa >= PV_FULL_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
282         if (!bsoc_mode) {
283             ResetPI(&E.mode.pid);
284         }
285         bsoc_mode = true;
286         bsoc_high = true;
287         if (!ha_ac_mode) {
288             ha_ac_on();
289             ha_ac_mode = true;
290         }
291     }
292     else {
293         if (bsoc_high) { // turn off at min limit power
294             if ((vpwa >= PV_MIN_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
295                 bsoc_mode = true;
296                 if (ha_ac_mode) {
297                     ha_ac_off();
298                     ha_ac_mode = false;
299                 }
300             } else {
301                 bsoc_high = false;
302                 ha_ac_mode = false;
303             }
304         }
305     }
306
307     E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) + E.mvar[V_DPPV]; // use as a temp variable
308     E.mode.total_system = (E.mvar[V_FLO] - E.mode.gti_dumpload) + E.mvar[V_DPPV] + (E.print_vars[L3_P]*
-1.0f);
309     E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) - E.mvar[V_DPPV]; // use this value
310
311     /*
312     * look at system energy balance for power control drive
313     */
314     if (mode) { // add GTI power from dumpload
315         E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + E.mode.gti_dumpload +
PBAL_OFFSET);
316     } else {
317         E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + PBAL_OFFSET);
318     }
319
320     if (E.mode.error > 0.0f) {
321         L.coef = COEF;
322     } else {
323         L.coef = COEFN;
324     }
325     E.mode.target = target;
326     E.mode.error = round(error_filter(E.mode.error));
327     /*
328     * check for idle flag from HA
329     */
330     if (E.mode.con6) {
331         ha_ac_mode = true;
332         bsoc_mode = false;
333     }
334
335     /*
336     * HA start excess button pressed
337     */
338     if (E.mode.con4) {
339         E.dl_excess = true;
340         E.mode.con4 = false;
341     }
342
343     /*
344     * HA stop excess button pressed
345     */
346     if (E.mode.con5) {
347         mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 9); // zero power at excess shutdown
348         E.dl_excess = false;
349         E.mode.con5 = false;

```

```

350     }
351
352     /*
353  * DL buffer battery low set-point excess load shutdown
354  */
355     if (E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) {
356         mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 10); // zero power at excess shutdown
357         E.dl_excess = false;
358         E.mode.con4 = false;
359         E.mode.con5 = false;
360     }
361
362     return bsoc_mode;
363 }

```

4.4.2.11 bsoc_set_std_dev()

```

void bsoc_set_std_dev (
    const double value,
    const uint32_t i )
76 {
77     L.bat_c_std_dev[i] = value;
78 }

```

4.4.2.12 calculateStandardDeviation()

```

double calculateStandardDeviation (
    const uint32_t ,
    const double * )

```

4.4.2.13 dc0_filter()

```

double dc0_filter (
    const double raw )
400 {
401     static double accum = 0.0f;
402     static double coef = COEFF;
403     accum = accum - accum / coef + raw;
404     return accum / coef;
405 }

```

4.4.2.14 dc1_filter()

```

double dc1_filter (
    const double raw )
408 {
409     static double accum = 0.0f;
410     static double coef = COEF;
411     accum = accum - accum / coef + raw;
412     return accum / coef;
413 }

```


4.4.2.15 dc2_filter()

```
double dc2_filter (
    const double raw )
416 {
417     static double accum = 0.0f;
418     static double coef = COEF;
419     accum = accum - accum / coef + raw;
420     return accum / coef;
421 }
```

4.4.2.16 drive0_filter()

```
double drive0_filter (
    const double raw )
424 {
425     static double accum = 0.0f;
426     static double coef = COEF;
427     accum = accum - accum / coef + raw;
428     return accum / coef;
429 }
```

4.4.2.17 drive1_filter()

```
double drive1_filter (
    const double raw )
432 {
433     static double accum = 0.0f;
434     static double coef = COEFF;
435     accum = accum - accum / coef + raw;
436     return accum / coef;
437 }
```

4.4.2.18 get_batc_dev()

```
double get_batc_dev (
    void )
196 {
197     return L.batc_std_dev;
198 }
```

4.4.2.19 gti_test()

```
double gti_test (
    void )
171 {
172     // check for 48VDC AC charger powered from the Solar battery bank AC inverter
173     if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
174         L.gti_weight = 0.0f; // reduce power to zero
175 #ifdef BSOC_DEBUG
176         fprintf(fout, "pvp %8.2f, gweight %8.2f, aweight %8.2f, batv %8.2f, batc %8.2f\r\n", pv_voltage,
            gti_weight, ac_weight, bat_voltage, bat_current);
177 #endif
178     } else {
179         if (E.dl_excess) {
180             if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
181                 L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
182             } else {
183                 L.gti_weight = 0.0f; // reduce power to zero
184             }
185         }
186     }
187     return dc0_filter(L.gti_weight);
188 }
```

4.5 bsoc.h

[Go to the documentation of this file.](#)

```

1 /*
2  * File:      bsoc.h
3  * Author:   root
4  *
5  * Created on February 10, 2024, 6:24 PM
6  */
7
8 #ifndef BSOC_H
9 #define BSOC_H
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14 #include <math.h>
15     // #define BSOC_DEBUG
16
17 #define MIN_PV_VOLTS      5.0f
18 #define MIN_BAT_VOLTS    23.0f
19 #define MIN_BAT_KW       4100.0f
20
21 #define DEV_SIZE          10
22 #define MAX_BATC_DEV      1.5f
23 #define BAT_C_DRAW        3.0f
24
25 #define PBAL_OFFSET       -50.0f // postive bias for control point
26 #define PV_FULL_PWR       300.0f
27 #define PV_MIN_PWR        160.0f
28 #define PV_V_NOM          60.0f
29 #define PV_V_FAKE         0.336699f
30
31 #define COEF              8.0f
32 #define COEFN             4.0f
33 #define COEFF             2.0f
34
35 #include <stdlib.h>
36 #include <stdio.h> /* for printf() */
37 #include <unistd.h>
38 #include <stdint.h>
39 #include <string.h>
40 #include <stdbool.h>
41 #include <signal.h>
42 #include <time.h>
43 #include <sys/wait.h>
44 #include <sys/types.h>
45 #include <sys/time.h>
46 #include <errno.h>
47 #include <math.h>
48 #include "pid.h"
49 #include "mqtt_rec.h"
50
51     bool bsoc_init(void);
52     bool bsoc_data_collect(void);
53     double bsoc_ac(void);
54     double bsoc_gti(void);
55     double gti_test(void);
56     double ac_test(void);
57     double get_batc_dev(void);
58     bool bat_current_stable(void);
59     void bsoc_set_std_dev(const double, const uint32_t);
60
61     double calculateStandardDeviation(const uint32_t, const double *);
62
63     bool bsoc_set_mode(const double, const bool, const bool);
64
65     double ac0_filter(const double);
66     double ac1_filter(const double);
67     double ac2_filter(const double);
68     double dc0_filter(const double);
69     double dc1_filter(const double);
70     double dc2_filter(const double);
71     double drive0_filter(const double);
72     double drive1_filter(const double);
73
74 #ifdef __cplusplus
75 }
76 #endif
77
78 #endif /* BSOC_H */
79

```

4.6 ha_energy/build/Debug/GNU-Linux/_ext/5c0/energy.o.d File Reference

4.7 ha_energy/build/Release/GNU-Linux/_ext/5c0/energy.o.d File Reference

4.8 ha_energy/build/Debug/GNU-Linux/bsoc.o.d File Reference

4.9 ha_energy/build/Release/GNU-Linux/bsoc.o.d File Reference

4.10 ha_energy/build/Debug/GNU-Linux/http_vars.o.d File Reference

4.11 ha_energy/build/Release/GNU-Linux/http_vars.o.d File Reference

4.12 ha_energy/build/Debug/GNU-Linux/mqtt_rec.o.d File Reference

4.13 ha_energy/build/Release/GNU-Linux/mqtt_rec.o.d File Reference

4.14 ha_energy/build/Debug/GNU-Linux/mqtt_vars.o.d File Reference

4.15 ha_energy/build/Release/GNU-Linux/mqtt_vars.o.d File Reference

4.16 ha_energy/build/Debug/GNU-Linux/pid.o.d File Reference

4.17 ha_energy/build/Release/GNU-Linux/pid.o.d File Reference

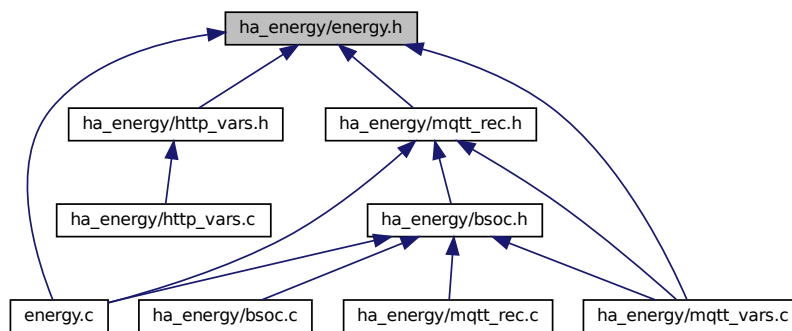
4.18 ha_energy/energy.h File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#include <signal.h>
#include <time.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/time.h>
#include <errno.h>
```

```
#include <cjson/cJSON.h>
#include <curl/curl.h>
#include <pthread.h>
#include <sys/stat.h>
#include <syslog.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netdb.h>
#include <ifaddrs.h>
#include "MQTTClient.h"
#include "pid.h"
Include dependency graph for energy.h:
```



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



Data Structures

- struct [link_type](#)
- struct [mode_type](#)
- struct [energy_type](#)

Macros

- #define [LOG_VERSION](#) "V0.73"
- #define [MQTT_VERSION](#) "V3.11"
- #define [TNAME](#) "maint9"
- #define [LADDRESS](#) "tcp://127.0.0.1:1883"
- #define [ADDRESS](#) "tcp://10.1.1.30:1883"
- #define [CLIENTID1](#) "Energy_Mqtt_HA1"
- #define [CLIENTID2](#) "Energy_Mqtt_HA2"
- #define [CLIENTID3](#) "Energy_Mqtt_HA3"
- #define [TOPIC_P](#) "mateq84/data/gticmd"
- #define [TOPIC_SPAM](#) "mateq84/data/spam"

- #define `TOPIC_PACA` "home-assistant/gtiac/availability"
- #define `TOPIC_PDCA` "home-assistant/gtidc/availability"
- #define `TOPIC_PACC` "home-assistant/gtiac/contact"
- #define `TOPIC_PDCC` "home-assistant/gtidc/contact"
- #define `TOPIC_PPID` "home-assistant/solar/pid"
- #define `TOPIC_SHUTDOWN` "home-assistant/solar/shutdown"
- #define `TOPIC_SS` "mateq84/data/solar"
- #define `TOPIC_SD` "mateq84/data/dumpload"
- #define `TOPIC_HA` "home-assistant/status/switch"
- #define `QOS` 1
- #define `TIMEOUT` 10000L
- #define `SPACING_USEC` 500 * 1000
- #define `USEC_SEC` 1000000L
- #define `DAQ_STR` 32
- #define `DAQ_STR_M` `DAQ_STR`-1
- #define `SBUF_SIZ` 16
- #define `RBUF_SIZ` 82
- #define `SYSLOG_SIZ` 512
- #define `MQTT_TIMEOUT` 900
- #define `SW_QOS` 1
- #define `NO_CYLON`
- #define `CRITIAL_SHUTDOWN_LOG`
- #define `UNIT_TEST` 2
- #define `NORM_MODE` 0
- #define `PID_MODE` 1
- #define `MAX_ERROR` 5
- #define `IAM_DELAY` 120
- #define `CMD_SEC` 10
- #define `TIME_SYNC_SEC` 30
- #define `BAT_M_KW` 5120.0f
- #define `BAT_SOC_TOP` 0.98f
- #define `BAT_SOC_HIGH` 0.95f
- #define `BAT_SOC_LOW` 0.64f
- #define `BAT_SOC_LOW_AC` 0.70f
- #define `BAT_CRITICAL` 200.0f
- #define `MIN_BAT_KW_BSOC_SLP` 4000.0f
- #define `MIN_BAT_KW_BSOC_HI` 4550.0f
- #define `MIN_BAT_KW_GTI_HI` `BAT_M_KW`*`BAT_SOC_TOP`
- #define `MIN_BAT_KW_GTI_LO` `BAT_M_KW`*`BAT_SOC_LOW`
- #define `MIN_BAT_KW_AC_HI` `BAT_M_KW`*`BAT_SOC_HIGH`
- #define `MIN_BAT_KW_AC_LO` `BAT_M_KW`*`BAT_SOC_LOW_AC`
- #define `PV_PGAIN` 0.85f
- #define `PV_IGAIN` 0.12f
- #define `PV_IMAX` 1400.0f
- #define `PV_BIAS` 288.0f
- #define `PV_BIAS_ZERO` 0.0f
- #define `PV_BIAS_LOW` 222.0f
- #define `PV_BIAS_FLOAT` 399.0f
- #define `PV_BIAS_SLEEP` 480.0f
- #define `PV_BIAS_RATE` 320.0f
- #define `PV_DL_MPTT_MAX` 1200.0f
- #define `PV_DL_MPTT_EXCESS` 1300.0f
- #define `PV_DL_MPTT_IDLE` 57.0f
- #define `PV_DL_BIAS_RATE` 75.0f
- #define `PV_DL_EXCESS` 500.0f

- #define PV_DL_B_AH_LOW 100.0f
- #define PV_DL_B_AH_MIN 150.0f
- #define PV_DL_B_V_LOW 23.8f
- #define PWA_SLEEP 200.0f
- #define DL_AC_DC_EFF 1.24f
- #define BAL_MIN_ENERGY_AC -200.0f
- #define BAL_MAX_ENERGY_AC 200.0f
- #define BAL_MIN_ENERGY_GTI -1400.0f
- #define BAL_MAX_ENERGY_GTI 200.0f
- #define LOG_TO_FILE "/store/logs/energy.log"
- #define LOG_TO_FILE_ALT "/tmp/energy.log"
- #define MAX_LOG_SPAM 60
- #define LOW_LOG_SPAM 2
- #define RESET_LOG_SPAM 120
- #define IM_DELAY 1
- #define IM_DISPLAY 1
- #define GTI_DELAY 1
- #define PWA_SANE 1700.0f
- #define PAMPS_SANE 16.0f
- #define PVOLTS_SANE 150.0f
- #define BAMPMS_SANE 70.0f
- #define MAX_IM_VAR IA_LAST*PHASE_LAST
- #define L1_P IA_POWER
- #define L2_P L1_P+IA_LAST
- #define L3_P L2_P+IA_LAST

Enumerations

- enum energy_state {
E_INIT , E_RUN , E_WAIT , E_IDLE ,
E_STOP , E_LAST }
- enum running_state {
R_INIT , R_FLOAT , R_SLEEP , R_RUN ,
R_IDLE , R_LAST }
- enum iammeter_phase { PHASE_A , PHASE_B , PHASE_C , PHASE_LAST }
- enum iammeter_id {
IA_VOLTAGE , IA_CURRENT , IA_POWER , IA_IMPORT ,
IA_EXPORT , IA_FREQ , IA_PF , IA_LAST }
- enum mqtt_vars {
V_FCCM , V_FBEKW , V_FRUNT , V_FBAMPS ,
V_FBV , V_FLO , V_FSO , V_FACE ,
V_BEN , V_PWA , V_PAMPS , V_PVOLTS ,
V_FLAST , V_HDCSW , V_HACSW , V_HSHUT ,
V_HMODE , V_HCON0 , V_HCON1 , V_HCON2 ,
V_HCON3 , V_HCON4 , V_HCON5 , V_HCON6 ,
V_HCON7 , V_DVPV , V_DPPV , V_DPBAT ,
V_DVBAT , V_DCMPTT , V_DPMPTT , V_DAHBAT ,
V_DCCMODE , V_DGTI , V_DLAST }
- enum sane_vars {
S_FCCM , S_FBEKW , S_FRUNT , S_FBAMPS ,
S_FBV , S_FLO , S_FSO , S_FACE ,
S_BEN , S_PWA , S_PAMPS , S_PVOLTS ,
S_FLAST , S_HDCSW , S_HACSW , S_HSHUT ,
S_HMODE , S_DVPV , S_DPPV , S_DPBAT ,
S_DVBAT , S_DCMPTT , S_DPMPTT , S_DAHBAT ,
S_DCCMODE , S_DGTI , S_DLAST }

Functions

- void [timer_callback](#) (int32_t)
- void [connlost](#) (void *, char *)
- void [ramp_up_gti](#) (MQTTClient, bool, bool)
- void [ramp_up_ac](#) (MQTTClient, bool)
- void [ramp_down_gti](#) (MQTTClient, bool)
- void [ramp_down_ac](#) (MQTTClient, bool)
- void [ha_ac_off](#) (void)
- void [ha_ac_on](#) (void)
- void [ha_dc_off](#) (void)
- void [ha_dc_on](#) (void)
- size_t [iammeter_write_callback](#) (char *, size_t, size_t, void *)
- void [iammeter_read](#) (void)
- void [print_im_vars](#) (void)
- void [print_mvar_vars](#) (void)
- bool [sanity_check](#) (void)
- char * [log_time](#) (bool)
- bool [sync_ha](#) (void)
- bool [log_timer](#) (void)

Variables

- struct [energy_type](#) E
- struct [ha_flag_type](#) ha_flag_vars_ss
- FILE * [fout](#)

4.18.1 Macro Definition Documentation

4.18.1.1 ADDRESS

```
#define ADDRESS "tcp://10.1.1.30:1883"
```

4.18.1.2 BAL_MAX_ENERGY_AC

```
#define BAL_MAX_ENERGY_AC 200.0f
```

4.18.1.3 BAL_MAX_ENERGY_GTI

```
#define BAL_MAX_ENERGY_GTI 200.0f
```

4.18.1.4 BAL_MIN_ENERGY_AC

```
#define BAL_MIN_ENERGY_AC -200.0f
```

4.18.1.5 BAL_MIN_ENERGY_GTI

```
#define BAL_MIN_ENERGY_GTI -1400.0f
```

4.18.1.6 BAMPS_SANE

```
#define BAMPS_SANE 70.0f
```

4.18.1.7 BAT_CRITICAL

```
#define BAT_CRITICAL 200.0f
```

4.18.1.8 BAT_M_KW

```
#define BAT_M_KW 5120.0f
```

4.18.1.9 BAT_SOC_HIGH

```
#define BAT_SOC_HIGH 0.95f
```

4.18.1.10 BAT_SOC_LOW

```
#define BAT_SOC_LOW 0.64f
```

4.18.1.11 BAT_SOC_LOW_AC

```
#define BAT_SOC_LOW_AC 0.70f
```


4.18.1.12 BAT_SOC_TOP

```
#define BAT_SOC_TOP 0.98f
```

4.18.1.13 CLIENTID1

```
#define CLIENTID1 "Energy_Mqtt_HA1"
```

4.18.1.14 CLIENTID2

```
#define CLIENTID2 "Energy_Mqtt_HA2"
```

4.18.1.15 CLIENTID3

```
#define CLIENTID3 "Energy_Mqtt_HA3"
```

4.18.1.16 CMD_SEC

```
#define CMD_SEC 10
```

4.18.1.17 CRITIAL_SHUTDOWN_LOG

```
#define CRITIAL_SHUTDOWN_LOG
```

4.18.1.18 DAQ_STR

```
#define DAQ_STR 32
```

4.18.1.19 DAQ_STR_M

```
#define DAQ_STR_M DAQ\_STR-1
```

4.18.1.20 DL_AC_DC_EFF

```
#define DL_AC_DC_EFF 1.24f
```

4.18.1.21 GTI_DELAY

```
#define GTI_DELAY 1
```

4.18.1.22 IAM_DELAY

```
#define IAM_DELAY 120
```

4.18.1.23 IM_DELAY

```
#define IM_DELAY 1
```

4.18.1.24 IM_DISPLAY

```
#define IM_DISPLAY 1
```

4.18.1.25 L1_P

```
#define L1_P IA\_POWER
```

4.18.1.26 L2_P

```
#define L2_P L1\_P+IA\_LAST
```

4.18.1.27 L3_P

```
#define L3_P L2\_P+IA\_LAST
```

4.18.1.28 LADDRESS

```
#define LADDRESS "tcp://127.0.0.1:1883"
```

4.18.1.29 LOG_TO_FILE

```
#define LOG_TO_FILE "/store/logs/energy.log"
```

4.18.1.30 LOG_TO_FILE_ALT

```
#define LOG_TO_FILE_ALT "/tmp/energy.log"
```

4.18.1.31 LOG_VERSION

```
#define LOG_VERSION "V0.73"
```

4.18.1.32 LOW_LOG_SPAM

```
#define LOW_LOG_SPAM 2
```

4.18.1.33 MAX_ERROR

```
#define MAX_ERROR 5
```

4.18.1.34 MAX_IM_VAR

```
#define MAX_IM_VAR IA_LAST*PHASE_LAST
```

4.18.1.35 MAX_LOG_SPAM

```
#define MAX_LOG_SPAM 60
```

4.18.1.36 MIN_BAT_KW_AC_HI

```
#define MIN_BAT_KW_AC_HI BAT_M_KW*BAT_SOC_HIGH
```

4.18.1.37 MIN_BAT_KW_AC_LO

```
#define MIN_BAT_KW_AC_LO BAT_M_KW*BAT_SOC_LOW_AC
```

4.18.1.38 MIN_BAT_KW_BSOC_HI

```
#define MIN_BAT_KW_BSOC_HI 4550.0f
```

4.18.1.39 MIN_BAT_KW_BSOC_SLP

```
#define MIN_BAT_KW_BSOC_SLP 4000.0f
```

4.18.1.40 MIN_BAT_KW_GTI_HI

```
#define MIN_BAT_KW_GTI_HI BAT_M_KW*BAT_SOC_TOP
```

4.18.1.41 MIN_BAT_KW_GTI_LO

```
#define MIN_BAT_KW_GTI_LO BAT_M_KW*BAT_SOC_LOW
```

4.18.1.42 MQTT_TIMEOUT

```
#define MQTT_TIMEOUT 900
```

4.18.1.43 MQTT_VERSION

```
#define MQTT_VERSION "V3.11"
```

4.18.1.44 NO_CYLON

```
#define NO_CYLON
```

4.18.1.45 NORM_MODE

```
#define NORM_MODE 0
```

4.18.1.46 PAMPS_SANE

```
#define PAMPS_SANE 16.0f
```

4.18.1.47 PID_MODE

```
#define PID_MODE 1
```

4.18.1.48 PV_BIAS

```
#define PV_BIAS 288.0f
```

4.18.1.49 PV_BIAS_FLOAT

```
#define PV_BIAS_FLOAT 399.0f
```

4.18.1.50 PV_BIAS_LOW

```
#define PV_BIAS_LOW 222.0f
```

4.18.1.51 PV_BIAS_RATE

```
#define PV_BIAS_RATE 320.0f
```

4.18.1.52 PV_BIAS_SLEEP

```
#define PV_BIAS_SLEEP 480.0f
```

4.18.1.53 PV_BIAS_ZERO

```
#define PV_BIAS_ZERO 0.0f
```

4.18.1.54 PV_DL_B_AH_LOW

```
#define PV_DL_B_AH_LOW 100.0f
```

4.18.1.55 PV_DL_B_AH_MIN

```
#define PV_DL_B_AH_MIN 150.0f
```

4.18.1.56 PV_DL_B_V_LOW

```
#define PV_DL_B_V_LOW 23.8f
```

4.18.1.57 PV_DL_BIAS_RATE

```
#define PV_DL_BIAS_RATE 75.0f
```

4.18.1.58 PV_DL_EXCESS

```
#define PV_DL_EXCESS 500.0f
```

4.18.1.59 PV_DL_MPTT_EXCESS

```
#define PV_DL_MPTT_EXCESS 1300.0f
```

4.18.1.60 PV_DL_MPTT_IDLE

```
#define PV_DL_MPTT_IDLE 57.0f
```

4.18.1.61 PV_DL_MPTT_MAX

```
#define PV_DL_MPTT_MAX 1200.0f
```

4.18.1.62 PV_IGAIN

```
#define PV_IGAIN 0.12f
```

4.18.1.63 PV_IMAX

```
#define PV_IMAX 1400.0f
```

4.18.1.64 PV_PGAIN

```
#define PV_PGAIN 0.85f
```

4.18.1.65 PVOLTS_SANE

```
#define PVOLTS_SANE 150.0f
```

4.18.1.66 PWA_SANE

```
#define PWA_SANE 1700.0f
```

4.18.1.67 PWA_SLEEP

```
#define PWA_SLEEP 200.0f
```

4.18.1.68 QOS

```
#define QOS 1
```

4.18.1.69 RBUF_SIZ

```
#define RBUF_SIZ 82
```

4.18.1.70 RESET_LOG_SPAM

```
#define RESET_LOG_SPAM 120
```

4.18.1.71 SBUF_SIZ

```
#define SBUF_SIZ 16
```

4.18.1.72 SPACING_USEC

```
#define SPACING_USEC 500 * 1000
```

4.18.1.73 SW_QOS

```
#define SW_QOS 1
```

4.18.1.74 SYSLOG_SIZ

```
#define SYSLOG_SIZ 512
```

4.18.1.75 TIME_SYNC_SEC

```
#define TIME_SYNC_SEC 30
```


4.18.1.76 TIMEOUT

```
#define TIMEOUT 10000L
```

4.18.1.77 TNAME

```
#define TNAME "maint9"
```

4.18.1.78 TOPIC_HA

```
#define TOPIC_HA "home-assistant/status/switch"
```

4.18.1.79 TOPIC_P

```
#define TOPIC_P "mateq84/data/gticmd"
```

4.18.1.80 TOPIC_PACA

```
#define TOPIC_PACA "home-assistant/gtiac/availability"
```

4.18.1.81 TOPIC_PACC

```
#define TOPIC_PACC "home-assistant/gtiac/contact"
```

4.18.1.82 TOPIC_PDCA

```
#define TOPIC_PDCA "home-assistant/gtidc/availability"
```

4.18.1.83 TOPIC_PDCC

```
#define TOPIC_PDCC "home-assistant/gtidc/contact"
```

4.18.1.84 TOPIC_PPID

```
#define TOPIC_PPID "home-assistant/solar/pid"
```

4.18.1.85 TOPIC_SD

```
#define TOPIC_SD "mateq84/data/dumpload"
```

4.18.1.86 TOPIC_SHUTDOWN

```
#define TOPIC_SHUTDOWN "home-assistant/solar/shutdown"
```

4.18.1.87 TOPIC_SPAM

```
#define TOPIC_SPAM "mateq84/data/spam"
```

4.18.1.88 TOPIC_SS

```
#define TOPIC_SS "mateq84/data/solar"
```

4.18.1.89 UNIT_TEST

```
#define UNIT_TEST 2
```

4.18.1.90 USEC_SEC

```
#define USEC_SEC 1000000L
```

4.18.2 Enumeration Type Documentation

4.18.2.1 energy_state

```
enum energy\_state
```

Enumerator

| | |
|--------|--|
| E_INIT | |
| E_RUN | |
| E_WAIT | |
| E_IDLE | |
| E_STOP | |
| E_LAST | |

```
194                                     {
195     E_INIT,
196     E_RUN,
197     E_WAIT,
198     E_IDLE,
199     E_STOP,
200     E_LAST,
201 };
```

4.18.2.2 iammeter_id

```
enum iammeter_id
```

Enumerator

| | |
|------------|--|
| IA_VOLTAGE | |
| IA_CURRENT | |
| IA_POWER | |
| IA_IMPORT | |
| IA_EXPORT | |
| IA_FREQ | |
| IA_PF | |
| IA_LAST | |

```
219                                     {
220     IA_VOLTAGE,
221     IA_CURRENT,
222     IA_POWER,
223     IA_IMPORT,
224     IA_EXPORT,
225     IA_FREQ,
226     IA_PF,
227     IA_LAST,
228 };
```

4.18.2.3 iammeter_phase

```
enum iammeter_phase
```

Enumerator

| | |
|------------|--|
| PHASE_A | |
| PHASE_B | |
| PHASE_C | |
| PHASE_LAST | |

```

212     {
213         PHASE_A,
214         PHASE_B,
215         PHASE_C,
216         PHASE_LAST,
217     };

```

4.18.2.4 mqtt_vars

```
enum mqtt_vars
```

Enumerator

| | |
|-----------|--|
| V_FCCM | |
| V_FBEKW | |
| V_FRUNT | |
| V_FBAMPS | |
| V_FBV | |
| V_FLO | |
| V_FSO | |
| V_FACE | |
| V_BEN | |
| V_PWA | |
| V_PAMPS | |
| V_PVOLTS | |
| V_FLAST | |
| V_HDCSW | |
| V_HACSW | |
| V_HSHUT | |
| V_HMODE | |
| V_HCON0 | |
| V_HCON1 | |
| V_HCON2 | |
| V_HCON3 | |
| V_HCON4 | |
| V_HCON5 | |
| V_HCON6 | |
| V_HCON7 | |
| V_DVPV | |
| V_DPPV | |
| V_DPBAT | |
| V_DVBAT | |
| V_DCMPPT | |
| V_DPMPPT | |
| V_DAHBAT | |
| V_DCCMODE | |
| V_DGTI | |
| V_DLAST | |

```

230     {
231         V_FCCM,
232         V_FBEKW,

```

```

233         V_FRUNT,
234         V_FBAMPS,
235         V_FBV,
236         V_FLO,
237         V_FSO,
238         V_FACE,
239         V_BEN,
240         V_PWA,
241         V_PAMPS,
242         V_PVOLTS,
243         V_FLAST,
244         V_HDCSW,
245         V_HACSW,
246         V_HSHUT,
247         V_HMODE,
248         V_HCON0,
249         V_HCON1,
250         V_HCON2,
251         V_HCON3,
252         V_HCON4,
253         V_HCON5,
254         V_HCON6,
255         V_HCON7,
256         // add other data ranges here
257         V_DVPV,
258         V_DPPV,
259         V_DPBAT,
260         V_DVBAT,
261         V_DCMPPPT,
262         V_DPMPPPT,
263         V_DAHBAT,
264         V_DCCMODE,
265         V_DGTI,
266         V_DLAST,
267     };

```

4.18.2.5 running_state

```
enum running_state
```

Enumerator

| | |
|---------|--|
| R_INIT | |
| R_FLOAT | |
| R_SLEEP | |
| R_RUN | |
| R_IDLE | |
| R_LAST | |

```

203     {
204         R_INIT,
205         R_FLOAT,
206         R_SLEEP,
207         R_RUN,
208         R_IDLE,
209         R_LAST,
210     };

```

4.18.2.6 sane_vars

```
enum sane_vars
```

Enumerator

| | |
|-----------|--|
| S_FCCM | |
| S_FBEKW | |
| S_FRUNT | |
| S_FBAMPS | |
| S_FBV | |
| S_FLO | |
| S_FSO | |
| S_FACE | |
| S_BEN | |
| S_PWA | |
| S_PAMPS | |
| S_PVOLTS | |
| S_FLAST | |
| S_HDCSW | |
| S_HACSW | |
| S_HSHUT | |
| S_HMODE | |
| S_DVPV | |
| S_DPPV | |
| S_DPBAT | |
| S_DVBAT | |
| S_DCMPPT | |
| S_DPMPPT | |
| S_DAHBAT | |
| S_DCCMODE | |
| S_DGTI | |
| S_DLAST | |

```

269     {
270         S_FCCM,
271         S_FBEKW,
272         S_FRUNT,
273         S_FBAMPS,
274         S_FBV,
275         S_FLO,
276         S_FSO,
277         S_FACE,
278         S_BEN,
279         S_PWA,
280         S_PAMPS,
281         S_PVOLTS,
282         S_FLAST,
283         S_HDCSW,
284         S_HACSW,
285         S_HSHUT,
286         S_HMODE,
287         // add other data ranges here
288         S_DVPV,
289         S_DPPV,
290         S_DPBAT,
291         S_DVBAT,
292         S_DCMPPT,
293         S_DPMPPT,
294         S_DAHBAT,
295         S_DCCMODE,
296         S_DGTI,
297         S_DLAST,
298     };

```

4.18.3 Function Documentation

4.18.3.1 connlost()

```
void connlost (
    void * context,
    char * cause )
298 {
299     struct ha_flag_type *ha_flag = context;
300     int32_t id_num;
301
302     // bug-out if no context variables passed to callback
303     if (context == NULL) {
304         id_num = -1;
305     } else {
306         id_num = ha_flag->ha_id;
307     }
308     fprintf(fout, "\n%s Connection lost, exit ha_energy program\n", log_time(false));
309     fprintf(fout, "%s      cause:  %s, %d\n", log_time(false), cause, id_num);
310     fprintf(fout, "%sDAEMON failure LOG Version %s : MQTT Version %s\n", log_time(false), LOG_VERSION,
MQTT_VERSION);
311     fflush(fout);
312     exit(EXIT_FAILURE);
313 }
```

4.18.3.2 ha_ac_off()

```
void ha_ac_off (
    void )
947 {
948     mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
949     E.ac_sw_status = false;
950 }
```

4.18.3.3 ha_ac_on()

```
void ha_ac_on (
    void )
953 {
954     mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
955     E.ac_sw_status = true;
956 }
```

4.18.3.4 ha_dc_off()

```
void ha_dc_off (
    void )
962 {
963     mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
964     E.gti_sw_status = false;
965 }
```

4.18.3.5 ha_dc_on()

```

void ha_dc_on (
    void )
968 {
969     mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
970     E.gti_sw_status = true;
971 }

```

4.18.3.6 iammeter_read()

```

void iammeter_read (
    void )
76 {
77
78     curl = curl_easy_init();
79     if (curl) {
80         E.link.iammeter_count++;
81         curl_easy_setopt(curl, CURLOPT_URL, "http://10.1.1.101/monitorjson");
82         curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, iammeter_write_callback);
83         curl_easy_setopt(curl, CURLOPT_WRITEDATA, E.print_vars); // external data array for iammeter
            values
84
85         res = curl_easy_perform(curl);
86         /* Check for errors */
87         if (res != CURLE_OK) {
88             fprintf(fout, "curl_easy_perform() failed in iammeter_read: %s\n",
89                 curl_easy_strerror(res));
90             E.iammeter = false;
91             E.link.iammeter_error++;
92         } else {
93             E.iammeter = true;
94         }
95         curl_easy_cleanup(curl);
96     }
97 }

```

4.18.3.7 iammeter_write_callback()

```

size_t iammeter_write_callback (
    char * buffer,
    size_t size,
    size_t nitems,
    void * stream )
14 {
15     cJSON *json = cJSON_ParseWithLength(buffer, strlen(buffer));
16     struct energy_type * e = stream;
17     uint32_t next_var = 0;
18
19     E.link.iammeter_count++;
20
21     if (json == NULL) {
22         const char *error_ptr = cJSON_GetErrorPtr();
23         E.link.iammeter_error++;
24         if (error_ptr != NULL) {
25             fprintf(fout, "Error in iammeter_write_callback %u: %s\n", E.link.iammeter_error,
                error_ptr);
26         }
27         goto iammeter_exit;
28     }
29 #ifdef IM_DEBUG
30     fprintf(fout, "\n iammeter_read_callback %s \n", buffer);
31 #endif
32
33     cJSON *data_result = cJSON_GetObjectItemCaseSensitive(json, "Datas");
34
35     if (!data_result) {

```



```

36         size = 0;
37         nitems = 0;
38         goto iammeter_exit;
39     }
40
41     cJSON *jname;
42     uint32_t phase = PHASE_A;
43
44     cJSON_ArrayForEach(jname, data_result)
45     {
46         cJSON *ianame;
47 #ifdef IM_DEBUG
48         fprintf(fout, "\n iammeter variables ");
49 #endif
50
51         cJSON_ArrayForEach(ianame, jname)
52         {
53             uint32_t phase_var = IA_VOLTAGE;
54             iammeter_get_data(ianame->valuedouble, phase_var, phase);
55             e->print_vars[next_var++] = ianame->valuedouble;
56 #ifdef IM_DEBUG
57             fprintf(fout, "%8.2f ", im_vars[phase_var][phase]);
58 #endif
59             phase_var++;
60         }
61         phase++;
62     }
63 #ifdef IM_DEBUG
64     fprintf(fout, "\n");
65 #endif
66
67 iammeter_exit:
68     cJSON_Delete(json);
69     return size * nitems;
70 }

```

4.18.3.8 log_time()

```

char * log_time (
    bool log )
1032 {
1033     static char time_log[RBUF_SIZ] = {0};
1034     static uint32_t len = 0, sync_time = TIME_SYNC_SEC - 1;
1035     time_t rawtime_log;
1036
1037     tzset();
1038     timezone = 0;
1039     daylight = 0;
1040     time(&rawtime_log);
1041     if (sync_time++ > TIME_SYNC_SEC) {
1042         sync_time = 0;
1043         snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
1044         time commands
1045         mqtt_gti_time(E.client_p, TOPIC_P, time_log);
1046     }
1047     sprintf(time_log, "%s", ctime(&rawtime_log));
1048     len = strlen(time_log);
1049     time_log[len - 1] = 0; // munge out the return character
1050     if (log) {
1051         fprintf(fout, "%s ", time_log);
1052         fflush(fout);
1053     }
1054
1055     return time_log;
1056 }

```

4.18.3.9 log_timer()

```

bool log_timer (
    void )

```

```

1091 {
1092     bool itstime = false;
1093
1094     if (E.log_spam < LOW_LOG_SPAM) {
1095         E.log_time_reset = 0;
1096         itstime = true;
1097     }
1098     if (E.log_time_reset > RESET_LOG_SPAM) {
1099         E.log_spam = 0;
1100         itstime = true;
1101     }
1102     return itstime;
1103 }

```

4.18.3.10 print_im_vars()

```

void print_im_vars (
    void )
{
111 {
112     static char time_log[RBUFF_SIZ] = {0};
113     static uint32_t sync_time = TIME_SYNC_SEC - 1;
114     time_t rawtime_log;
115     char imvars[SYSLOG_SIZ];
116
117     fflush(fout);
118     snprintf(imvars, SYSLOG_SIZ-1, "House L1 %7.2fW, House L2 %7.2fW, GTI L1 %7.2fW",
E.print_vars[L1_P], E.print_vars[L2_P], E.print_vars[L3_P]);
119     fprintf(fout, "%s", imvars);
120     fflush(fout);
121     time(&rawtime_log);
122     if (sync_time++ > TIME_SYNC_SEC) {
123         sync_time = 0;
124         snprintf(time_log, RBUFF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
time commands
125         mqtt_gti_time(E.client_p, TOPIC_P, time_log);
126     }
127 }

```

4.18.3.11 print_mvar_vars()

```

void print_mvar_vars (
    void )
{
237 {
238     fprintf(fout, ", AC Inverter %7.2fW, BAT Energy %7.2fWh, Solar E %7.2fWh, AC E %7.2fWh, PERR %7.2fW,
PBAL %7.2fW, ST %7.2f, GDL %7.2f, %d,%d,%d %d,%d,%d\r",
239         E.mvar[V_FLO], E.mvar[V_FBEKW], E.mvar[V_FSO], E.mvar[V_FACE], E.mode.error, E.mvar[V_BEN],
E.mode.total_system, E.mode.gti_dumpload, (int32_t) E.mvar[V_HDCSW], (int32_t) E.mvar[V_HACSW],
(int32_t) E.mvar[V_HMODE],
240         (int32_t) E.dc_mismatch, (int32_t) E.ac_mismatch, (int32_t) E.mode_mismatch);
241 }

```

4.18.3.12 ramp_down_ac()

```

void ramp_down_ac (
    MQTTClient client_p,
    bool sw_off )
{
937 {
938     if (sw_off) {
939         mqtt_ha_switch(client_p, TOPIC_PACC, false);
940         E.ac_sw_status = false;
941         usleep(500000);
942     }
943     E.once_ac = true;
944 }

```

4.18.3.13 ramp_down_gti()

```
void ramp_down_gti (
    MQTTClient client_p,
    bool sw_off )
904 {
905     if (sw_off) {
906         mqtt_ha_switch(client_p, TOPIC_PDCC, false);
907         E.once_gti_zero = true;
908         E.gti_sw_status = false;
909     }
910     E.once_gti = true;
911
912     if (E.once_gti_zero) {
913         mqtt_gti_power(client_p, TOPIC_P, "Z#", 7); // zero power
914         E.once_gti_zero = false;
915     }
916 }
```

4.18.3.14 ramp_up_ac()

```
void ramp_up_ac (
    MQTTClient client_p,
    bool start )
922 {
923
924     if (start) {
925         E.once_ac = true;
926     }
927
928     if (E.once_ac) {
929         E.once_ac = false;
930         mqtt_ha_switch(client_p, TOPIC_PACC, true);
931         E.ac_sw_status = true;
932         usleep(500000); // wait for voltage to ramp
933     }
934 }
```

4.18.3.15 ramp_up_gti()

```
void ramp_up_gti (
    MQTTClient client_p,
    bool start,
    bool excess )
843 {
844     static uint32_t sequence = 0;
845
846     if (start) {
847         E.once_gti = true;
848     }
849
850     if (E.once_gti) {
851         E.once_gti = false;
852         sequence = 0;
853         if (!excess) {
854             mqtt_ha_switch(client_p, TOPIC_PDCC, true);
855             E.gti_sw_status = true;
856             usleep(500000); // wait for voltage to ramp
857         } else {
858             sequence = 1;
859         }
860     }
861
862     switch (sequence) {
863     case 4:
864         E.once_gti_zero = true;
865         break;
```

```

866     case 3:
867     case 2:
868     case 1:
869         E.once_gti_zero = true;
870         if (bat_current_stable() || E.dl_excess) { // check battery current std dev, stop 'motorboating'
871             sequence++;
872             if (!mqtt_gti_power(client_p, TOPIC_P, "+#", 3)) {
873                 sequence = 0;
874             }; // +100W power
875         } else {
876             usleep(500000); // wait a bit more for power to be stable
877             sequence = 1; // do power ramps when ready
878             if (!mqtt_gti_power(client_p, TOPIC_P, "-#", 4)) {
879                 sequence = 0;
880             }; // - 100W power
881         }
882         break;
883     case 0:
884         sequence++;
885         if (E.once_gti_zero) {
886             mqtt_gti_power(client_p, TOPIC_P, "Z#", 5); // zero power
887             E.once_gti_zero = false;
888         }
889         break;
890     default:
891         if (E.once_gti_zero) {
892             mqtt_gti_power(client_p, TOPIC_P, "Z#", 6); // zero power
893             E.once_gti_zero = false;
894         }
895         sequence = 0;
896         break;
897     }
898 }

```

4.18.3.16 sanity_check()

```

bool sanity_check (
    void )
254 {
255     if (E.mvar[V_PWA] > PWA_SANE) {
256         E.sane = S_PWA;
257         return false;
258     }
259     if (E.mvar[V_PAMPS] > PAMPS_SANE) {
260         E.sane = S_PAMPS;
261         return false;
262     }
263     if (E.mvar[V_PVOLTS] > PVOLTS_SANE) {
264         E.sane = S_PVOLTS;
265         return false;
266     }
267     if (E.mvar[V_FBAMPS] > BAMPS_SANE) {
268         E.sane = S_FBAMPS;
269         return false;
270     }
271     return true;
272 }

```

4.18.3.17 sync_ha()

```

bool sync_ha (
    void )
1062 {
1063     bool sync = false;
1064     if (E.gti_sw_status != (bool) ((int32_t) E.mvar[V_HDCSW])) {
1065         fprintf(fout, "DC_MM %d %d ", (bool) E.gti_sw_status, (bool) ((int32_t) E.mvar[V_HDCSW]));
1066         mqtt_ha_switch(E.client_p, TOPIC_PDCC, !E.gti_sw_status);
1067         E.dc_mismatch = true;
1068         fflush(fout);
1069         sync = true;
1070     } else {

```

```

1071     E.dc_mismatch = false;
1072 }
1073
1074 E.ac_sw_status = (bool) ((int32_t) E.mvar[V_HACSW]); // TEMP FIX for Mismatch errors
1075 if (E.ac_sw_status != (bool) ((int32_t) E.mvar[V_HACSW])) {
1076     fprintf(fout, "AC_MM %d %d ", (bool) E.ac_sw_status, (bool) ((int32_t) E.mvar[V_HACSW]));
1077     mqtt_ha_switch(E.client_p, TOPIC_PACC, !E.ac_sw_status);
1078     E.ac_mismatch = true;
1079     fflush(fout);
1080     sync = true;
1081 } else {
1082     E.ac_mismatch = false;
1083 }
1084 return sync;
1085 }

```

4.18.3.18 timer_callback()

```

void timer_callback (
    int32_t signum )
283 {
284     signal(signum, timer_callback);
285     ha_flag_vars_ss.runner = true;
286     E.ten_sec_clock++;
287     E.log_spam++;
288     E.log_time_reset++;
289     if (E.log_spam > MAX_LOG_SPAM) {
290         E.log_spam = 0;
291     }
292 }

```

4.18.4 Variable Documentation

4.18.4.1 E

```
struct energy_type E [extern]
```

4.18.4.2 fout

```
FILE* fout [extern]
```

4.18.4.3 ha_flag_vars_ss

```
struct ha_flag_type ha_flag_vars_ss [extern]
```

4.19 energy.h

[Go to the documentation of this file.](#)

```

1  /*
2  * File:      bmc.h
3  * Author:   root
4  *
5  * Created on September 21, 2012, 12:54 PM
6  */
7
8  #ifndef BMC_H
9  #define BMC_H
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14 #include <stdlib.h>
15 #include <stdio.h> /* for printf() */
16 #include <unistd.h>
17 #include <stdint.h>
18 #include <string.h>
19 #include <stdbool.h>
20 #include <signal.h>
21 #include <time.h>
22 #include <sys/wait.h>
23 #include <sys/types.h>
24 #include <sys/time.h>
25 #include <errno.h>
26 #include <cjson/cJSON.h>
27 #include <curl/curl.h>
28 #include <pthread.h>
29 #include <sys/stat.h>
30 #include <syslog.h>
31 #include <arpa/inet.h>
32 #include <sys/socket.h>
33 #include <netdb.h>
34 #include <ifaddrs.h>
35 #include "MQTTClient.h"
36 #include "pid.h"
37
38
39 #define LOG_VERSION      "V0.73"
40 #define MQTT_VERSION     "V3.11"
41 #define TNAME            "maint9"
42 #define LADDRESS         "tcp://127.0.0.1:1883"
43 #ifdef __amd64
44 #define ADDRESS          "tcp://10.1.1.172:1883"
45 #else
46 #define ADDRESS          "tcp://10.1.1.30:1883"
47 #endif
48 #define CLIENTID1        "Energy_Mqtt_HA1"
49 #define CLIENTID2        "Energy_Mqtt_HA2"
50 #define CLIENTID3        "Energy_Mqtt_HA3"
51 #define TOPIC_P           "mateq84/data/gticmd"
52 #define TOPIC_SPAM       "mateq84/data/spam"
53 #define TOPIC_PACA        "home-assistant/gtiac/availability"
54 #define TOPIC_PDCA        "home-assistant/gtidc/availability"
55 #define TOPIC_PACC        "home-assistant/gtiac/contact"
56 #define TOPIC_PDCC        "home-assistant/gtidc/contact"
57 #define TOPIC_PPID        "home-assistant/solar/pid"
58 #define TOPIC_SHUTDOWN    "home-assistant/solar/shutdown"
59 #define TOPIC_SS          "mateq84/data/solar"
60 #define TOPIC_SD          "mateq84/data/dumpload"
61 #define TOPIC_HA          "home-assistant/status/switch"
62 #define QOS               1
63 #define TIMEOUT           10000L
64 #define SPACING_USEC      500 * 1000
65 #define USEC_SEC          1000000L
66
67 #define DAQ_STR           32
68 #define DAQ_STR_M         DAQ_STR-1
69
70 #define SBUF_SIZ          16 // short buffer string size
71 #define RBUF_SIZ          82
72 #define SYSLOG_SIZ        512
73
74 #define MQTT_TIMEOUT      900
75 #define SW_QOS            1
76
77 #define NO_CYLON
78 #define CRITIAL_SHUTDOWN_LOG
79
80 #define UNIT_TEST         2
81 #define NORM_MODE         0
82 #define PID_MODE          1

```

```

83 #define MAX_ERROR          5
84 #define IAM_DELAY          120
85
86 #define CMD_SEC            10
87 #define TIME_SYNC_SEC     30
88
89 /*
90 * Battery SoC cycle limits parameters
91 */
92 #define BAT_M_KW            5120.0f
93 #define BAT_SOC_TOP         0.98f
94 #define BAT_SOC_HIGH        0.95f
95 #define BAT_SOC_LOW         0.64f
96 #define BAT_SOC_LOW_AC      0.70f
97 #define BAT_CRITICAL        200.0f
98 #define MIN_BAT_KW_BSOC_SLP 4000.0f
99 #define MIN_BAT_KW_BSOC_HI  4550.0f
100
101 #define MIN_BAT_KW_GTI_HI    BAT_M_KW*BAT_SOC_TOP
102 #define MIN_BAT_KW_GTI_LO    BAT_M_KW*BAT_SOC_LOW
103
104 #define MIN_BAT_KW_AC_HI     BAT_M_KW*BAT_SOC_HIGH
105 #define MIN_BAT_KW_AC_LO     BAT_M_KW*BAT_SOC_LOW_AC
106
107 /*
108 * PV panel cycle limits parameters
109 */
110 #define PV_PGAIN             0.85f
111 #define PV_IGAIN             0.12f
112 #define PV_IMAX              1400.0f
113 #define PV_BIAS              288.0f
114 #define PV_BIAS_ZERO         0.0f
115 #define PV_BIAS_LOW          222.0f
116 #define PV_BIAS_FLOAT        399.0f
117 #define PV_BIAS_SLEEP        480.0f
118 #define PV_BIAS_RATE          320.0f
119 #define PV_DL_MPTT_MAX       1200.0f
120 #define PV_DL_MPTT_EXCESS    1300.0f
121 #define PV_DL_MPTT_IDLE      57.0f
122 #define PV_DL_BIAS_RATE      75.0f
123 #define PV_DL_EXCESS         500.0f
124 #define PV_DL_B_AH_LOW       100.0f
125 #define PV_DL_B_AH_MIN       150.0f // DL battery should be at least 175Ah
126 #define PV_DL_B_V_LOW        23.8f // Battery low-voltage cutoff
127 #define PWA_SLEEP            200.0f
128 #define DL_AC_DC_EFF         1.24f
129
130 /*
131 * Energy control loop parameters
132 */
133 #define BAL_MIN_ENERGY_AC     -200.0f
134 #define BAL_MAX_ENERGY_AC     200.0f
135 #define BAL_MIN_ENERGY_GTI    -1400.0f
136 #define BAL_MAX_ENERGY_GTI    200.0f
137
138 #define LOG_TO_FILE           "/store/logs/energy.log"
139 #define LOG_TO_FILE_ALT       "/tmp/energy.log"
140
141 #define MAX_LOG_SPAM          60
142 #define LOW_LOG_SPAM          2
143 #define RESET_LOG_SPAM       120
144
145 // #define IM_DEBUG              // WEM3080T LOGGING
146 // #define B_ADJ_DEBUG           // debug printing
147 // #define FAKE_VPV              // NO AC CHARGER for DUMPLoad, batteries are cross-connected
148 // #define PSW_DEBUG             to a parallel bank
149 // #define DEBUG_SHUTDOWN
150
151 // #define AUTO_CHARGE            // turn on dumptload charger during restarts
152 // #define B_DLE_DEBUG           // Dump Load debugging
153 // #define BSOC_DEGUB
154 // #define DEBUG_HA_CMD
155
156 #define IM_DELAY              1 // tens of second updates
157 #define IM_DISPLAY            1
158 #define GTI_DELAY             1
159
160 /*
161 * sane limits for system data elements
162 */
163 #define PWA_SANE               1700.0f
164 #define PAMPS_SANE            16.0f
165 #define PVOLTS_SANE           150.0f
166 #define BAMPMS_SANE           70.0f
167
168 /*

```

```

169 Three Phase WiFi Energy Meter (WEM3080T)
170 name      Unit      Description
171 wem3080t_voltage_a  V    A phase voltage
172 wem3080t_current_a  A    A phase current
173 wem3080t_power_a    W    A phase active power
174 wem3080t_importenergy_a kWh A phase import energy
175 wem3080t_exportgrid_a kWh A phase export energy
176 wem3080t_frequency_a kWh A phase frequency
177 wem3080t_pf_a      kWh A phase power factor
178 wem3080t_voltage_b  V    B phase voltage
179 wem3080t_current_b  A    B phase current
180 wem3080t_power_b    W    B phase active power
181 wem3080t_importenergy_b kWh B phase import energy
182 wem3080t_exportgrid_b kWh B phase export energy
183 wem3080t_frequency_b kWh B phase frequency
184 wem3080t_pf_b      kWh B phase power factor
185 wem3080t_voltage_c  V    C phase voltage
186 wem3080t_current_c  A    C phase current
187 wem3080t_power_c    W    C phase active power
188 wem3080t_importenergy_c kWh C phase import energy
189 wem3080t_exportgrid_c kWh C phase export energy
190 wem3080t_frequency_c kWh C phase frequency
191 wem3080t_pf_c      kWh C phase power factor
192 */
193
194     enum energy_state {
195         E_INIT,
196         E_RUN,
197         E_WAIT,
198         E_IDLE,
199         E_STOP,
200         E_LAST,
201     };
202
203     enum running_state {
204         R_INIT,
205         R_FLOAT,
206         R_SLEEP,
207         R_RUN,
208         R_IDLE,
209         R_LAST,
210     };
211
212     enum iammeter_phase {
213         PHASE_A,
214         PHASE_B,
215         PHASE_C,
216         PHASE_LAST,
217     };
218
219     enum iammeter_id {
220         IA_VOLTAGE,
221         IA_CURRENT,
222         IA_POWER,
223         IA_IMPORT,
224         IA_EXPORT,
225         IA_FREQ,
226         IA_PF,
227         IA_LAST,
228     };
229
230     enum mqtt_vars {
231         V_FCCM,
232         V_FBEKW,
233         V_FRUNT,
234         V_FBAMPS,
235         V_FBV,
236         V_FLO,
237         V_FSO,
238         V_FACE,
239         V_BEN,
240         V_PWA,
241         V_PAMPS,
242         V_PVOLTS,
243         V_FLAST,
244         V_HDCSW,
245         V_HACSW,
246         V_HSHUT,
247         V_HMODE,
248         V_HCON0,
249         V_HCON1,
250         V_HCON2,
251         V_HCON3,
252         V_HCON4,
253         V_HCON5,
254         V_HCON6,
255         V_HCON7,

```



```

256         // add other data ranges here
257         V_DVPV,
258         V_DPPV,
259         V_DPBAT,
260         V_DVBAT,
261         V_DCMPPT,
262         V_DMPMPPT,
263         V_DAHBAT,
264         V_DCCMODE,
265         V_DGTI,
266         V_DLAST,
267     };
268
269     enum sane_vars {
270         S_FCCM,
271         S_FBEKW,
272         S_FRUNT,
273         S_FBAMPS,
274         S_FBV,
275         S_FLO,
276         S_FSO,
277         S_FACE,
278         S_BEN,
279         S_PWA,
280         S_PAMPS,
281         S_PVOLTS,
282         S_FLAST,
283         S_HDCSW,
284         S_HACSW,
285         S_HSHUT,
286         S_HMODE,
287         // add other data ranges here
288         S_DVPV,
289         S_DPPV,
290         S_DPBAT,
291         S_DVBAT,
292         S_DCMPPT,
293         S_DMPMPPT,
294         S_DAHBAT,
295         S_DCCMODE,
296         S_DGTI,
297         S_DLAST,
298     };
299
300 #define MAX_IM_VAR   IA_LAST+PHASE_LAST
301
302 #define L1_P         IA_POWER
303 #define L2_P         L1_P+IA_LAST
304 #define L3_P         L2_P+IA_LAST
305
306     struct link_type {
307         volatile uint32_t iammeter_error, iammeter_count;
308         volatile uint32_t mqtt_error, mqtt_count;
309         volatile uint32_t shutdown;
310     };
311
312     struct mode_type {
313         volatile double error, target, total_system, gti_dumpload, pv_bias, dl_mqtt_max, off_grid,
sequence;
314         volatile bool mode, in_pid_control, con0, con1, con2, con3, con4, con5, con6, con7, no_float,
data_error, bat_crit;
315         volatile uint32_t mode_tmr;
316         volatile struct SPid pid;
317         volatile enum energy_state E;
318         volatile enum running_state R;
319     };
320
321     struct energy_type {
322         volatile double print_vars[MAX_IM_VAR];
323         volatile double im_vars[IA_LAST][PHASE_LAST];
324         volatile double mvar[V_DLAST + 1];
325         volatile bool once_gti, once_ac, iammeter, fm80, dumpload, homeassistant, once_gti_zero;
326         volatile double gti_low_adj, ac_low_adj, dl_excess_adj;
327         volatile bool ac_sw_on, gti_sw_on, ac_sw_status, gti_sw_status, solar_shutdown, solar_mode,
startup, ac_mismatch, dc_mismatch, mode_mismatch, dl_excess;
328         volatile uint32_t speed_go, im_delay, im_display, gti_delay;
329         volatile int32_t rc, sane;
330         volatile uint32_t ten_sec_clock, log_spam, log_time_reset;
331         pthread_mutex_t ha_lock;
332         struct mode_type mode;
333         struct link_type link;
334         MQTTClient client_p, client_sd, client_ha;
335     };
336
337     extern struct energy_type E;
338     extern struct ha_flag_type ha_flag_vars_ss;
339     extern FILE* fout;

```


4.20.1.1 iammeter_get_data()

```

static void iammeter_get_data (
    const double valuedouble,
    const uint32_t i,
    const uint32_t j ) [static]
103 {
104     E.im_vars[i][j] = valuedouble;
105 }

```

4.20.1.2 iammeter_read()

```

void iammeter_read (
    void )
76 {
77
78     curl = curl_easy_init();
79     if (curl) {
80         E.link.iammeter_count++;
81         curl_easy_setopt(curl, CURLOPT_URL, "http://10.1.1.101/monitorjson");
82         curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, iammeter_write_callback);
83         curl_easy_setopt(curl, CURLOPT_WRITEDATA, E.print_vars); // external data array for iammeter
            values
84
85         res = curl_easy_perform(curl);
86         /* Check for errors */
87         if (res != CURLE_OK) {
88             fprintf(fout, "curl_easy_perform() failed in iammeter_read: %s\n",
89                 curl_easy_strerror(res));
90             E.iammeter = false;
91             E.link.iammeter_error++;
92         } else {
93             E.iammeter = true;
94         }
95         curl_easy_cleanup(curl);
96     }
97 }

```

4.20.1.3 iammeter_write_callback()

```

size_t iammeter_write_callback (
    char * buffer,
    size_t size,
    size_t nitems,
    void * stream )
14 {
15     cJSON *json = cJSON_ParseWithLength(buffer, strlen(buffer));
16     struct energy_type * e = stream;
17     uint32_t next_var = 0;
18
19     E.link.iammeter_count++;
20
21     if (json == NULL) {
22         const char *error_ptr = cJSON_GetErrorPtr();
23         E.link.iammeter_error++;
24         if (error_ptr != NULL) {
25             fprintf(fout, "Error in iammeter_write_callback %u: %s\n", E.link.iammeter_error,
                error_ptr);
26         }
27         goto iammeter_exit;
28     }
29 #ifdef IM_DEBUG
30     fprintf(fout, "\n iammeter_read_callback %s \n", buffer);
31 #endif
32
33     cJSON *data_result = cJSON_GetObjectItemCaseSensitive(json, "Datas");

```

```

34
35     if (!data_result) {
36         size = 0;
37         nitems = 0;
38         goto iammeter_exit;
39     }
40
41     cJSON *jname;
42     uint32_t phase = PHASE_A;
43
44     cJSON_ArrayForEach(jname, data_result)
45     {
46         cJSON *ianame;
47 #ifdef IM_DEBUG
48         fprintf(fout, "\n iammeter variables ");
49 #endif
50
51         cJSON_ArrayForEach(ianame, jname)
52         {
53             uint32_t phase_var = IA_VOLTAGE;
54             iammeter_get_data(ianame->valuedouble, phase_var, phase);
55             e->print_vars[next_var++] = ianame->valuedouble;
56 #ifdef IM_DEBUG
57             fprintf(fout, "%.2f ", im_vars[phase_var][phase]);
58 #endif
59             phase_var++;
60         }
61         phase++;
62     }
63 #ifdef IM_DEBUG
64     fprintf(fout, "\n");
65 #endif
66
67 iammeter_exit:
68     cJSON_Delete(json);
69     return size * nitems;
70 }

```

4.20.1.4 mqtt_gti_time()

```

bool mqtt_gti_time (
    MQTTClient client_p,
    const char * topic_p,
    char * msg )
249 {
250     bool ret = true;
251     MQTTClient_message pubmsg = MQTTClient_message_initializer;
252     MQTTClient_deliveryToken token;
253     ha_flag_vars_ss.deliveredtoken = 0;
254
255     E.link.mqtt_count++;
256     pubmsg.payload = msg;
257     pubmsg.payloadlen = strlen(msg);
258     pubmsg.qos = QOS;
259     pubmsg.retained = 0;
260
261     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run time commands
262
263     // a busy, wait loop for the async delivery thread to complete
264     {
265         uint32_t waiting = 0;
266         while (ha_flag_vars_ss.deliveredtoken != token) {
267             usleep(GTI_TOKEN_DELAY);
268             if (waiting++ > MQTT_TIMEOUT) {
269                 fprintf(fout, "\r\n%s GTI Time Still Waiting, timeout\r\n", log_time(false));
270                 break;
271             }
272         };
273     }
274     usleep(HA_SW_DELAY);
275     return ret;
276 }

```

4.20.1.5 print_im_vars()

```
void print_imvars (
    void )

111 {
112     static char time_log[RBUF_SIZ] = {0};
113     static uint32_t sync_time = TIME_SYNC_SEC - 1;
114     time_t rawtime_log;
115     char imvars[SYSLOG_SIZ];
116
117     fflush(fout);
118     snprintf(imvars, SYSLOG_SIZ-1, "House L1 %7.2fW, House L2 %7.2fW, GTI L1 %7.2fW",
E.print_vars[L1_P], E.print_vars[L2_P], E.print_vars[L3_P]);
119     fprintf(fout, "%s", imvars);
120     fflush(fout);
121     time(&rawtime_log);
122     if (sync_time++ > TIME_SYNC_SEC) {
123         sync_time = 0;
124         snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
time commands
125         mqtt_gti_time(E.client_p, TOPIC_P, time_log);
126     }
127 }
```

4.20.2 Variable Documentation

4.20.2.1 curl

CURL* curl [static]

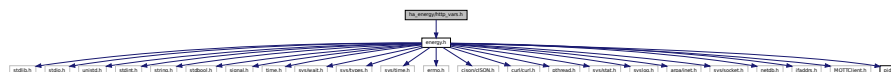
4.20.2.2 res

```
CURLcode res [static]
```

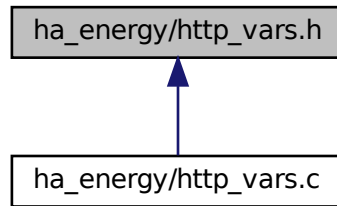
4.21 ha_energy/http_vars.h File Reference

```
#include "energy.h"
```

Include dependency graph for http_vars.h:



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



Variables

- FILE * `fout`

4.21.1 Variable Documentation

4.21.1.1 fout

```
FILE* fout [extern]
```

4.22 http_vars.h

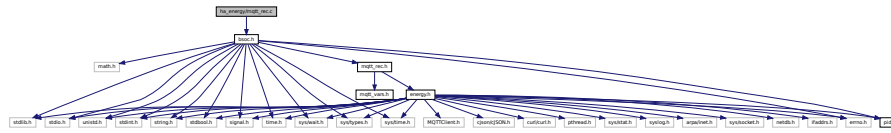
[Go to the documentation of this file.](#)

```
1 /*
2  * File:      http_vars.h
3  * Author:   root
4  *
5  * Created on February 16, 2024, 8:37 AM
6  */
7
8 #ifndef HTTP_VARS_H
9 #define HTTP_VARS_H
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14
15 #include "energy.h"
16
17     extern FILE* fout;
18
19 #ifdef __cplusplus
20 }
21 #endif
22
23 #endif /* HTTP_VARS_H */
24
```

4.23 ha_energy/mqtt_rec.c File Reference

```
#include "bsoc.h"
```

Include dependency graph for mqtt_rec.c:



Functions

- `int32_t msgarrvd` (void *context, char *topicName, int topicLen, MQTTClient_message *message)
- `void delivered` (void *context, MQTTClient_deliveryToken dt)
- `bool json_get_data` (cJSON *json_src, const char *data_id, cJSON *name, uint32_t i)
- `void print_mvar_vars` (void)
- `bool fm80_float` (const bool set_bias)
- `bool fm80_sleep` (void)

4.23.1 Function Documentation

4.23.1.1 delivered()

```
void delivered (
    void * context,
    MQTTClient_deliveryToken dt )
123 {
124     struct ha_flag_type *ha_flag = context;
125
126     // bug-out if no context variables passed to callback
127     if (context == NULL) {
128         return;
129     }
130     ha_flag->deliveredtoken = dt;
131 }
```

4.23.1.2 fm80_float()

```
bool fm80_float (
    const bool set_bias )
{
    247 if ((uint32_t) E.mvar[V_FCCM] == FLOAT_CODE) {
    248     if (set_bias) {
    249         E.mode.pv_bias = PV_BIAS_FLOAT;
    250     }
    251     if (E.mode.R != R_IDLE) {
    252         E.mode.R = R_FLOAT;
    253     }
    254     return true;
    255 } else {
    256     if (E.mode.R == R_FLOAT) {
    257         E.mode.R = R_RUN;
    258     }
    259 }
    260 return false;
    261 }
    262 }
```

4.23.1.3 fm80_sleep()

```
bool fm80_sleep (
    void )
269 {
270     if ((uint32_t) E.mvar[V_FCCM] == SLEEP_CODE) {
271         return true;
272     }
273     return false;
274 }
```

4.23.1.4 json_get_data()

```
bool json_get_data (
    cJSON * json_src,
    const char * data_id,
    cJSON * name,
    uint32_t i )
138 {
139     bool ret = false;
140     static uint32_t j = 0;
141
142     // access the JSON data using the lookup string passed in data_id
143     name = cJSON_GetObjectItemCaseSensitive(json_src, data_id);
144
145     /*
146  * process string values
147  */
148     if (cJSON_IsString(name) && (name->valstring != NULL)) {
149 #ifdef GET_DEBUG
150         fprintf(fout, "%s Name:  %s\n", data_id, name->valstring);
151 #endif
152         ret = true;
153     }
154
155     /*
156  * process numeric values
157  */
158     if (cJSON_IsNumber(name)) {
159 #ifdef GET_DEBUG
160         fprintf(fout, "%s Value:  %f\n", data_id, name->valuedouble);
161 #endif
162         if (i > V_DLAST) { // check for out-of-range index
163             i = V_DLAST;
164         }
165
166         // lock the main value array during updates
167         pthread_mutex_lock(&E.ha_lock);
168         E.mvar[i] = name->valuedouble;
169         pthread_mutex_unlock(&E.ha_lock);
170
171         /*
172  * special processing for variable data received
173  */
174         if (i == V_DCMPPPT) {
175             /*
176  * load battery current standard deviation array bat_c_std_dev with data
177  */
178             bsoc_set_std_dev(E.mvar[i], j++);
179             if (j >= RDEV_SIZE) {
180                 j = 0;
181             }
182         }
183         /*
184  * update local MATTER switch status from HA
185  */
186         if (i == V_HDCSW) {
187             E.gti_sw_status = (bool) ((int32_t) E.mvar[i]);
188             E.dc_mismatch = false;
189         }
190
191         if (i == V_HACSW) {
192             E.ac_sw_status = (bool) ((int32_t) E.mvar[i]);
193             E.ac_mismatch = false;
194         }
195     }
```



```

195
196     // command HA_ENERGY to shutdown mode
197     if (i == V_HSHUT) {
198         E.solar_shutdown = (bool) ((int32_t) E.mvar[i]);
199     }
200     // set HA_ENERGY energy processing mode
201     if (i == V_HMODE) {
202         ha_flag_vars_ss.energy_mode = (bool) ((int32_t) E.mvar[i]);
203     }
204     if (i == V_HCON0) {
205         E.mode.con0 = (bool) ((int32_t) E.mvar[i]);
206     }
207     if (i == V_HCON1) {
208         E.mode.con1 = (bool) ((int32_t) E.mvar[i]);
209     }
210     if (i == V_HCON2) {
211         E.mode.con2 = (bool) ((int32_t) E.mvar[i]);
212     }
213     if (i == V_HCON3) {
214         E.mode.con3 = (bool) ((int32_t) E.mvar[i]);
215     }
216     if (i == V_HCON4) { // set DL GTI excess load MODE
217         E.mode.con4 = (bool) ((int32_t) E.mvar[i]);
218     }
219     if (i == V_HCON5) { // clear DL GTI excess load MODE
220         E.mode.con5 = (bool) ((int32_t) E.mvar[i]);
221     }
222     if (i == V_HCON6) { // HA Energy program idle
223         E.mode.con6 = (bool) ((int32_t) E.mvar[i]);
224     }
225     if (i == V_HCON7) { // HA Energy program exit
226         E.mode.con7 = (bool) ((int32_t) E.mvar[i]);
227     }
228     ret = true;
229 }
230 return ret;
231 }

```

4.23.1.5 msgarrvd()

```

int32_t msgarrvd (
    void * context,
    char * topicName,
    int topicLen,
    MQTTClient_message * message )
{
    8 {
    9     int32_t i, ret = 1;
    10     const char* payloadptr;
    11     char buffer[MBMQTT];
    12     struct ha_flag_type *ha_flag = context;
    13
    14     E.link.mqtt_count++;
    15     // bug-out if no context variables passed to callback
    16     if (context == NULL) {
    17         ret = -1;
    18         goto null_exit;
    19     }
    20
    21 #ifdef DEBUG_REC
    22     fprintf(fout, "Message arrived\n");
    23 #endif
    24     /*
    25     * move the received message into a processing holding buffer
    26     */
    27     payloadptr = message->payload;
    28     for (i = 0; i < message->payloadlen; i++) {
    29         buffer[i] = *payloadptr++;
    30     }
    31     buffer[i] = 0; // make a null terminated C string
    32
    33     // parse the JSON data in the holding buffer
    34     cJSON *json = cJSON_ParseWithLength(buffer, message->payloadlen);
    35     if (json == NULL) {
    36         const char *error_ptr = cJSON_GetErrorPtr();
    37         if (error_ptr != NULL) {
    38             fprintf(fout, "%s Error: %s NULL cJSON pointer\n", log_time(false), error_ptr);
    39         }
    40         ret = -1;

```

```

41     ha_flag->rec_ok = false;
42     E.fm80 = false;
43     E.dumpload = false;
44     E.homeassistant = false;
45     E.link.mqtt_error++;
46     goto error_exit;
47 }
48
49 /*
50 * MQTT messages from the FM80 Q84 interface
51 */
52 if (ha_flag->ha_id == FM80_ID) {
53 #ifdef DEBUG_REC
54     fprintf(fout, "FM80 MQTT data\r\n");
55 #endif
56     cJSON *data_result = json;
57
58     for (uint32_t ii = V_FCCM; ii < V_FLAST; ii++) {
59         if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
60             ha_flag->var_update++;
61         }
62     }
63     E.fm80 = true;
64 }
65
66 /*
67 * MQTT messages from the K42 dumpload/gti interface
68 */
69 if (ha_flag->ha_id == DUMPLOAD_ID) {
70 #ifdef DEBUG_REC
71     fprintf(fout, "DUMPLOAD MQTT data\r\n");
72 #endif
73     cJSON *data_result = json;
74
75     for (uint32_t ii = V_HDCSW; ii < V_DLAST; ii++) {
76         if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
77             ha_flag->var_update++;
78         }
79     }
80     E.dumpload = true;
81 }
82
83 /*
84 * MQTT messages from the Linux HA_ENERGY interface
85 */
86 if (ha_flag->ha_id == HA_ID) {
87 #ifdef DEBUG_REC
88     fprintf(fout, "Home Assistant MQTT data\r\n");
89 #endif
90     cJSON *data_result = json;
91
92     if (json_get_data(json, mqtt_name[V_HACSW], data_result, V_HACSW)) {
93         ha_flag->var_update++;
94     }
95     data_result = json;
96     if (json_get_data(json, mqtt_name[V_HDCSW], data_result, V_HDCSW)) {
97         ha_flag->var_update++;
98     }
99
100     E.homeassistant = true;
101 }
102
103 // done with processing MQTT async message, set state flags
104 ha_flag->receivedtoken = true;
105 ha_flag->rec_ok = true;
106 /*
107 * exit and delete/free resources. In steps depending of possible error conditions
108 */
109 error_exit:
110     // delete the JSON object
111     cJSON_Delete(json);
112 null_exit:
113     // free the MQTT objects
114     MQTTClient_freeMessage(&message);
115     MQTTClient_free(topicName);
116     return ret;
117 }

```

4.23.1.6 print_mvar_vars()

```

void print_mvar_vars (
    void )

```

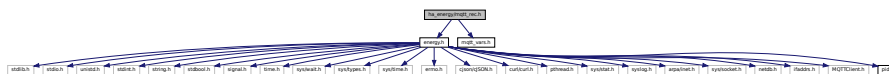
```

237 {
238     fprintf(fout, ", AC Inverter %7.2fW, BAT Energy %7.2fWh, Solar E %7.2fWh, AC E %7.2fWh, PERR %7.2fW,
PBAL %7.2fW, ST %7.2E, GDL %7.2f, %d,%d,%d %d,%d,%d\r",
239         E.mvar[V_FLO], E.mvar[V_PBEKW], E.mvar[V_FSO], E.mvar[V_FACE], E.mode.error, E.mvar[V_BEN],
E.mode.total_system, E.mode.gti_dumpload, (int32_t) E.mvar[V_HDCSW], (int32_t) E.mvar[V_HACSW],
(int32_t) E.mvar[V_HMODE],
240         (int32_t) E.dc_mismatch, (int32_t) E.ac_mismatch, (int32_t) E.mode_mismatch);
241 }

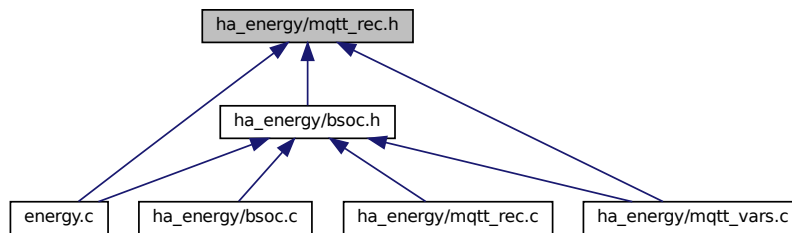
```

4.24 ha_energy/mqtt_rec.h File Reference

```
#include "energy.h"
#include "mqtt_vars.h"
Include dependency graph for mqtt_rec.h:
```



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



Data Structures

- struct `ha_flag_type`

Macros

- #define RDEV_SIZE 10
- #define SLEEP_CODE 0
- #define FLOAT_CODE 1
- #define MBMQTT 1024

Enumerations

- enum mqtt_id {
P8055_ID , FM80_ID , DUMPLoad_ID , HA_ID ,
LAST MQTT_ID }

Functions

- int32_t [msgarrvd](#) (void *, char *, int, MQTTClient_message *)
- void [delivered](#) (void *, MQTTClient_deliveryToken)
- bool [json_get_data](#) (cJSON *, const char *, cJSON *, uint32_t)
- bool [fm80_float](#) (const bool set_bias)
- bool [fm80_sleep](#) (void)

Variables

- FILE * [fout](#)

4.24.1 Macro Definition Documentation

4.24.1.1 FLOAT_CODE

```
#define FLOAT_CODE 1
```

4.24.1.2 MBMQTT

```
#define MBMQTT 1024
```

4.24.1.3 RDEV_SIZE

```
#define RDEV_SIZE 10
```

4.24.1.4 SLEEP_CODE

```
#define SLEEP_CODE 0
```

4.24.2 Enumeration Type Documentation

4.24.2.1 mqtt_id

```
enum mqtt\_id
```

Enumerator

| | |
|--------------|--|
| P8055_ID | |
| FM80_ID | |
| DUMPLOAD_ID | |
| HA_ID | |
| LAST_MQTT_ID | |

```

27         {
28             P8055_ID,
29             FM80_ID,
30             DUMPLOAD_ID,
31             HA_ID,
32             LAST_MQTT_ID,
33         };

```

4.24.3 Function Documentation

4.24.3.1 delivered()

```

void delivered (
    void * context,
    MQTTClient_deliveryToken dt )
123 {
124     struct ha_flag_type *ha_flag = context;
125
126     // bug-out if no context variables passed to callback
127     if (context == NULL) {
128         return;
129     }
130     ha_flag->deliveredtoken = dt;
131 }

```

4.24.3.2 fm80_float()

```

bool fm80_float (
    const bool set_bias )
247 {
248     if ((uint32_t) E.mvar[V_FCCM] == FLOAT_CODE) {
249         if (set_bias) {
250             E.mode.pv_bias = PV_BIAS_FLOAT;
251         }
252         if (E.mode.R != R_IDLE) {
253             E.mode.R = R_FLOAT;
254         }
255         return true;
256     } else {
257         if (E.mode.R == R_FLOAT) {
258             E.mode.R = R_RUN;
259         }
260     }
261     return false;
262 }

```

4.24.3.3 fm80_sleep()

```
bool fm80_sleep (
    void )
269 {
270     if ((uint32_t) E.mvar[V_FCCM] == SLEEP_CODE) {
271         return true;
272     }
273     return false;
274 }
```

4.24.3.4 json_get_data()

```
bool json_get_data (
    cJSON * json_src,
    const char * data_id,
    cJSON * name,
    uint32_t i )
138 {
139     bool ret = false;
140     static uint32_t j = 0;
141
142     // access the JSON data using the lookup string passed in data_id
143     name = cJSON_GetObjectItemCaseSensitive(json_src, data_id);
144
145     /*
146  * process string values
147  */
148     if (cJSON_IsString(name) && (name->valstring != NULL)) {
149 #ifdef GET_DEBUG
150         fprintf(fout, "%s Name:  %s\n", data_id, name->valstring);
151 #endif
152         ret = true;
153     }
154
155     /*
156  * process numeric values
157  */
158     if (cJSON_IsNumber(name)) {
159 #ifdef GET_DEBUG
160         fprintf(fout, "%s Value:  %f\n", data_id, name->valuedouble);
161 #endif
162         if (i > V_DLAST) { // check for out-of-range index
163             i = V_DLAST;
164         }
165
166         // lock the main value array during updates
167         pthread_mutex_lock(&E.ha_lock);
168         E.mvar[i] = name->valuedouble;
169         pthread_mutex_unlock(&E.ha_lock);
170
171         /*
172  * special processing for variable data received
173  */
174         if (i == V_DCMPPPT) {
175             /*
176  * load battery current standard deviation array bat_c_std_dev with data
177  */
178             bsoc_set_std_dev(E.mvar[i], j++);
179             if (j >= RDEV_SIZE) {
180                 j = 0;
181             }
182         }
183         /*
184  * update local MATTER switch status from HA
185  */
186         if (i == V_HDCSW) {
187             E.gti_sw_status = (bool) ((int32_t) E.mvar[i]);
188             E.dc_mismatch = false;
189         }
190
191         if (i == V_HACSW) {
192             E.ac_sw_status = (bool) ((int32_t) E.mvar[i]);
193             E.ac_mismatch = false;
194         }
195     }
```

```

195
196     // command HA_ENERGY to shutdown mode
197     if (i == V_HSHUT) {
198         E.solar_shutdown = (bool) ((int32_t) E.mvar[i]);
199     }
200     // set HA_ENERGY energy processing mode
201     if (i == V_HMODE) {
202         ha_flag_vars_ss.energy_mode = (bool) ((int32_t) E.mvar[i]);
203     }
204     if (i == V_HCON0) {
205         E.mode.con0 = (bool) ((int32_t) E.mvar[i]);
206     }
207     if (i == V_HCON1) {
208         E.mode.con1 = (bool) ((int32_t) E.mvar[i]);
209     }
210     if (i == V_HCON2) {
211         E.mode.con2 = (bool) ((int32_t) E.mvar[i]);
212     }
213     if (i == V_HCON3) {
214         E.mode.con3 = (bool) ((int32_t) E.mvar[i]);
215     }
216     if (i == V_HCON4) { // set DL GTI excess load MODE
217         E.mode.con4 = (bool) ((int32_t) E.mvar[i]);
218     }
219     if (i == V_HCON5) { // clear DL GTI excess load MODE
220         E.mode.con5 = (bool) ((int32_t) E.mvar[i]);
221     }
222     if (i == V_HCON6) { // HA Energy program idle
223         E.mode.con6 = (bool) ((int32_t) E.mvar[i]);
224     }
225     if (i == V_HCON7) { // HA Energy program exit
226         E.mode.con7 = (bool) ((int32_t) E.mvar[i]);
227     }
228     ret = true;
229 }
230 return ret;
231 }

```

4.24.3.5 msgarrvd()

```

int32_t msgarrvd (
    void * context,
    char * topicName,
    int topicLen,
    MQTTClient_message * message )
{
    8 {
    9     int32_t i, ret = 1;
    10     const char* payloadptr;
    11     char buffer[MBMQTT];
    12     struct ha_flag_type *ha_flag = context;
    13
    14     E.link.mqtt_count++;
    15     // bug-out if no context variables passed to callback
    16     if (context == NULL) {
    17         ret = -1;
    18         goto null_exit;
    19     }
    20
    21 #ifdef DEBUG_REC
    22     fprintf(fout, "Message arrived\n");
    23 #endif
    24     /*
    25     * move the received message into a processing holding buffer
    26     */
    27     payloadptr = message->payload;
    28     for (i = 0; i < message->payloadlen; i++) {
    29         buffer[i] = *payloadptr++;
    30     }
    31     buffer[i] = 0; // make a null terminated C string
    32
    33     // parse the JSON data in the holding buffer
    34     cJSON *json = cJSON_ParseWithLength(buffer, message->payloadlen);
    35     if (json == NULL) {
    36         const char *error_ptr = cJSON_GetErrorPtr();
    37         if (error_ptr != NULL) {
    38             fprintf(fout, "%s Error: %s NULL cJSON pointer\n", log_time(false), error_ptr);
    39         }
    40         ret = -1;
    41     }

```

```

41     ha_flag->rec_ok = false;
42     E.fm80 = false;
43     E.dumpload = false;
44     E.homeassistant = false;
45     E.link.mqtt_error++;
46     goto error_exit;
47 }
48
49 /*
50 * MQTT messages from the FM80 Q84 interface
51 */
52 if (ha_flag->ha_id == FM80_ID) {
53 #ifdef DEBUG_REC
54     fprintf(fout, "FM80 MQTT data\r\n");
55 #endif
56     cJSON *data_result = json;
57
58     for (uint32_t ii = V_FCCM; ii < V_FLAST; ii++) {
59         if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
60             ha_flag->var_update++;
61         }
62     }
63     E.fm80 = true;
64 }
65
66 /*
67 * MQTT messages from the K42 dumpload/gti interface
68 */
69 if (ha_flag->ha_id == DUMPLOAD_ID) {
70 #ifdef DEBUG_REC
71     fprintf(fout, "DUMPLOAD MQTT data\r\n");
72 #endif
73     cJSON *data_result = json;
74
75     for (uint32_t ii = V_HDCSW; ii < V_DLAST; ii++) {
76         if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
77             ha_flag->var_update++;
78         }
79     }
80     E.dumpload = true;
81 }
82
83 /*
84 * MQTT messages from the Linux HA_ENERGY interface
85 */
86 if (ha_flag->ha_id == HA_ID) {
87 #ifdef DEBUG_REC
88     fprintf(fout, "Home Assistant MQTT data\r\n");
89 #endif
90     cJSON *data_result = json;
91
92     if (json_get_data(json, mqtt_name[V_HACSW], data_result, V_HACSW)) {
93         ha_flag->var_update++;
94     }
95     data_result = json;
96     if (json_get_data(json, mqtt_name[V_HDCSW], data_result, V_HDCSW)) {
97         ha_flag->var_update++;
98     }
99
100     E.homeassistant = true;
101 }
102
103 // done with processing MQTT async message, set state flags
104 ha_flag->receivedtoken = true;
105 ha_flag->rec_ok = true;
106 /*
107 * exit and delete/free resources. In steps depending of possible error conditions
108 */
109 error_exit:
110     // delete the JSON object
111     cJSON_Delete(json);
112 null_exit:
113     // free the MQTT objects
114     MQTTClient_freeMessage(&message);
115     MQTTClient_free(topicName);
116     return ret;
117 }

```

4.24.4 Variable Documentation

4.24.4.1 fout

FILE* fout [extern]

4.25 mqtt_rec.h

[Go to the documentation of this file.](#)

```

1 /*
2  * File:      mqtt_rec.h
3  * Author:   root
4  *
5  * Created on February 5, 2024, 2:54 PM
6  */
7
8 #ifndef MQTT_REC_H
9 #define MQTT_REC_H
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14
15 #include "energy.h"
16 #include "mqtt_vars.h"
17
18 #define RDEV_SIZE      10
19
20 #define SLEEP_CODE      0
21 #define FLOAT_CODE      1
22     //#define DEBUG_REC
23     //#define GET_DEBUG
24
25 #define MBMQTT          1024
26
27     enum mqtt_id {
28         P8055_ID,
29         FM80_ID,
30         DUMpload_ID,
31         HA_ID,
32         LAST_MQTT_ID,
33     };
34
35     struct ha_flag_type {
36         volatile MQTTClient_deliveryToken deliveredtoken, receivedtoken;
37         volatile bool runner, rec_ok;
38         int32_t ha_id;
39         volatile int32_t var_update, energy_mode;
40     };
41
42     extern FILE* fout;
43
44     int32_t msgarrvd(void *, char *, int, MQTTClient_message *);
45     void delivered(void *, MQTTClient_deliveryToken);
46
47     bool json_get_data(cJSON *, const char *, cJSON *, uint32_t);
48     bool fm80_float(const bool set_bias);
49     bool fm80_sleep(void);
50
51 #ifdef __cplusplus
52 }
53 #endif
54
55 #endif /* MQTT_REC_H */
56

```

4.26 ha_energy/mqtt_vars.c File Reference

```

#include "mqtt_rec.h"
#include "energy.h"

```


4.26.2.1 mqtt_gti_power()

```

bool mqtt_gti_power (
    MQTTClient client_p,
    const char * topic_p,
    char * msg,
    uint32_t trace )
187 {
188     bool ret = true;
189     MQTTClient_message pubmsg = MQTTClient_message_initializer;
190     MQTTClient_deliveryToken token;
191     ha_flag_vars_ss.deliveredtoken = 0;
192     static bool spam = false;
193
194     E.link.mqtt_count++;
195     pubmsg.payload = msg;
196     pubmsg.payloadlen = strlen(msg);
197     pubmsg.qos = QOS;
198     pubmsg.retained = 0;
199
200     if (E.dl_excess) { // always run excess commands
201         spam = false;
202     }
203 #ifdef GTI_NO_POWER
204     MQTTClient_publishMessage(client_p, "mateq84/data/gticmd_nopower", &pubmsg, &token);
205 #else
206     if (bsoc_gti() > MIN_BAT_KW || E.dl_excess) {
207 #ifdef DEBUG_HA_CMD
208         log_time(true);
209         fprintf(fout, "HA GTI power command %s, SDEV %5.2f trace %u\r\n", msg, get_batc_dev(), trace);
210         fflush(fout);
211         spam = true;
212 #endif
213         MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run power commands
214     } else {
215         ret = false;
216         pubmsg.payload = "Z#";
217         pubmsg.payloadlen = strlen("Z#");
218         if (!spam) {
219             MQTTClient_publishMessage(client_p, TOPIC_SPAM, &pubmsg, &token);
220         } else {
221             MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // only shutdown GTI power
222         }
223 #ifdef DEBUG_HA_CMD
224         if (spam) {
225             log_time(true);
226             fprintf(fout, "HA GTI power set to zero, trace %u\r\n", trace);
227             fflush(fout);
228             spam = false;
229         }
230 #endif
231     }
232 #endif
233     // a busy, wait loop for the async delivery thread to complete
234     {
235         uint32_t waiting = 0;
236         while (ha_flag_vars_ss.deliveredtoken != token) {
237             usleep(TOKEN_DELAY);
238             if (waiting++ > MQTT_TIMEOUT) {
239                 fprintf(fout, "\r\n%s GTI Power Still Waiting, timeout\r\n", log_time(false));
240                 break;
241             }
242         };
243     }
244     usleep(HA_SW_DELAY);
245     return ret;
246 }

```

4.26.2.2 mqtt_gti_time()

```

bool mqtt_gti_time (
    MQTTClient client_p,
    const char * topic_p,
    char * msg )

```

```

249 {
250     bool ret = true;
251     MQTTClient_message pubmsg = MQTTClient_message_initializer;
252     MQTTClient_deliveryToken token;
253     ha_flag_vars_ss.deliveredtoken = 0;
254
255     E.link.mqtt_count++;
256     pubmsg.payload = msg;
257     pubmsg.payloadlen = strlen(msg);
258     pubmsg.qos = QOS;
259     pubmsg.retained = 0;
260
261     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run time commands
262
263     // a busy, wait loop for the async delivery thread to complete
264     {
265         uint32_t waiting = 0;
266         while (ha_flag_vars_ss.deliveredtoken != token) {
267             usleep(GTI_TOKEN_DELAY);
268             if (waiting++ > MQTT_TIMEOUT) {
269                 fprintf(fout, "\r\n%s GTI Time Still Waiting, timeout\r\n", log_time(false));
270                 break;
271             }
272         };
273     }
274     usleep(HA_SW_DELAY);
275     return ret;
276 }

```

4.26.2.3 mqtt_ha_pid()

```

void mqtt_ha_pid (
    MQTTClient client_p,
    const char * topic_p )
{
46 {
47     cJSON *json;
48     time_t rawtime;
49
50     MQTTClient_message pubmsg = MQTTClient_message_initializer;
51     MQTTClient_deliveryToken token;
52     ha_flag_vars_ss.deliveredtoken = 0;
53
54     E.link.mqtt_count++;
55     E.mode.sequence++;
56     json = cJSON_CreateObject();
57     cJSON_AddStringToObject(json, "name", CLIENTID1);
58     cJSON_AddNumberToObject(json, "sequence", E.mode.sequence);
59     cJSON_AddNumberToObject(json, "mqtt_count", (double) E.link.mqtt_count);
60     cJSON_AddNumberToObject(json, "http_count", (double) E.link.iammeter_count);
61     cJSON_AddNumberToObject(json, "piderror", E.mode.error);
62     cJSON_AddNumberToObject(json, "totalsystem", E.mode.total_system);
63     cJSON_AddNumberToObject(json, "gtinet", E.mode.gti_dumpload);
64     cJSON_AddNumberToObject(json, "energy_state", (double) E.mode.E);
65     cJSON_AddNumberToObject(json, "run_state", (double) E.mode.R);
66     // correct for power sensed by GTI metering
67     E.mode.off_grid = (E.mvar[V_FLO] - (E.mvar[V_DPPV] * DL_AC_DC_EFF));
68     E.mode.off_grid = drive1_filter(E.mode.off_grid);
69     if (E.mode.off_grid < 0.0f) { // only see power removed from grid usage
70         E.mode.off_grid = 0.0f;
71     }
72     cJSON_AddNumberToObject(json, "off_grid", E.mode.off_grid);
73     cJSON_AddNumberToObject(json, "excess_mode", (double) E.dl_excess);
74     cJSON_AddStringToObject(json, "build_date", FW_Date);
75     cJSON_AddStringToObject(json, "build_time", FW_Time);
76     time(&rawtime);
77     cJSON_AddNumberToObject(json, "sequence_time", (double) rawtime);
78     // convert the cJSON object to a JSON string
79     char *json_str = cJSON_Print(json);
80
81     pubmsg.payload = json_str;
82     pubmsg.payloadlen = strlen(json_str);
83     pubmsg.qos = QOS;
84     pubmsg.retained = 0;
85
86     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
87     // a busy, wait loop for the async delivery thread to complete
88     {
89         uint32_t waiting = 0;
90         while (ha_flag_vars_ss.deliveredtoken != token) {

```

```

91         usleep(TOKEN_DELAY);
92         if (waiting++ > MQTT_TIMEOUT) {
93             fprintf(fout, "\r\n%s SW Still Waiting, timeout\r\n", log_time(false));
94             break;
95         }
96     };
97 }
98
99 cJSON_free(json_str);
100 cJSON_Delete(json);
101 }

```

4.26.2.4 mqtt_ha_shutdown()

```

void mqtt_ha_shutdown (
    MQTTClient client_p,
    const char * topic_p )
13 {
14     cJSON *json;
15     MQTTClient_message pubmsg = MQTTClient_message_initializer;
16     MQTTClient_deliveryToken token;
17     ha_flag_vars_ss.deliveredtoken = 0;
18
19     json = cJSON_CreateObject();
20     cJSON_AddStringToObject(json, "shutdown", CLIENTID1);
21     char *json_str = cJSON_Print(json);
22
23     pubmsg.payload = json_str;
24     pubmsg.payloadlen = strlen(json_str);
25     pubmsg.qos = QOS;
26     pubmsg.retained = 0;
27
28     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
29     // a busy, wait loop for the async delivery thread to complete
30     {
31         uint32_t waiting = 0;
32         while (ha_flag_vars_ss.deliveredtoken != token) {
33             usleep(TOKEN_DELAY);
34             if (waiting++ > MQTT_TIMEOUT) {
35                 fprintf(fout, "\r\n%s SW Still Waiting, timeout\r\n", log_time(false));
36                 break;
37             }
38         };
39     }
40 }

```

4.26.2.5 mqtt_ha_switch()

```

void mqtt_ha_switch (
    MQTTClient client_p,
    const char * topic_p,
    const bool sw_state )
107 {
108     cJSON *json;
109     #ifdef DEBUG_HA_CMD
110         static bool spam = false;
111         static uint32_t less_spam = 0;
112     #endif
113
114     MQTTClient_message pubmsg = MQTTClient_message_initializer;
115     MQTTClient_deliveryToken token;
116     ha_flag_vars_ss.deliveredtoken = 0;
117
118     E.link.mqtt_count++;
119     json = cJSON_CreateObject();
120     if (sw_state) {
121         cJSON_AddStringToObject(json, "state", "ON");
122     }
123     #ifdef DEBUG_HA_CMD
124         spam = true;
125     #endif

```

```

124         less_spam = 0;
125 #endif
126     } else {
127         if ((uint32_t) E.mvar[V_FCCM] != FLOAT_CODE) { // use max power in FLOAT mode
128             cJSON_AddStringToObject(json, "state", "OFF");
129         } else {
130             cJSON_AddStringToObject(json, "state", "ON");
131 #ifdef DEBUG_HA_CMD
132             spam = true;
133             less_spam = 0;
134 #endif
135         }
136     }
137     // convert the cJSON object to a JSON string
138     char *json_str = cJSON_Print(json);
139
140     pubmsg.payload = json_str;
141     pubmsg.payloadlen = strlen(json_str);
142     pubmsg.qos = QOS;
143     pubmsg.retained = 0;
144
145 #ifdef DEBUG_HA_CMD
146     if (spam) {
147         log_time(true);
148         fflush(fout);
149         fprintf(fout, "HA switch command %s, %d\r\n", topic_p, sw_state);
150         fflush(fout);
151         if (!sw_state) {
152             if (less_spam++ > 3) {
153                 spam = false;
154                 less_spam = 0;
155             }
156         }
157     }
158 #endif
159
160     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
161     // a busy, wait loop for the async delivery thread to complete
162     {
163         uint32_t waiting = 0;
164         while (ha_flag_vars_ss.deliveredtoken != token) {
165             usleep(TOKEN_DELAY);
166             if (waiting++ > MQTT_TIMEOUT) {
167 #ifdef DEBUG_HA_CMD
168                 fflush(fout);
169                 fprintf(fout, "\r\nSW Still Waiting, timeout\r\n");
170                 fflush(fout);
171 #endif
172                 break;
173             }
174         };
175     }
176
177     cJSON_free(json_str);
178     cJSON_Delete(json);
179     usleep(HA_SW_DELAY);
180     fflush(fout);
181 }

```

4.26.3 Variable Documentation

4.26.3.1 FW_Date

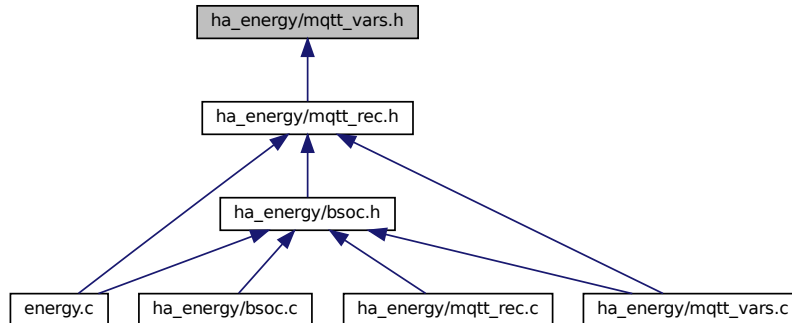
```
const char* const FW_Date = __DATE__ [static]
```

4.26.3.2 FW_Time

```
const char* const FW_Time = __TIME__ [static]
```

4.27 ha_energy/mqtt_vars.h File Reference

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



Macros

- `#define HA_SW_DELAY 400000`
- `#define TOKEN_DELAY 250`
- `#define GTI_TOKEN_DELAY 300`
- `#define QOS 1`

Functions

- void `mqtt_ha_switch` (MQTTClient, const char *, const bool)
- void `mqtt_ha_pid` (MQTTClient, const char *)
- void `mqtt_ha_shutdown` (MQTTClient, const char *)
- bool `mqtt_gti_power` (MQTTClient, const char *, char *, uint32_t)
- bool `mqtt_gti_time` (MQTTClient, const char *, char *)

Variables

- const char * `mqtt_name` [V_DLAST]

4.27.1 Macro Definition Documentation

4.27.1.1 GTI_TOKEN_DELAY

```
#define GTI_TOKEN_DELAY 300
```

4.27.1.2 HA_SW_DELAY

```
#define HA_SW_DELAY 400000
```

4.27.1.3 QOS

```
#define QOS 1
```

4.27.1.4 TOKEN_DELAY

```
#define TOKEN_DELAY 250
```

4.27.2 Function Documentation

4.27.2.1 mqtt_gti_power()

```
bool mqtt_gti_power (
    MQTTClient client_p,
    const char * topic_p,
    char * msg,
    uint32_t trace )
187 {
188     bool ret = true;
189     MQTTClient_message pubmsg = MQTTClient_message_initializer;
190     MQTTClient_deliveryToken token;
191     ha_flag_vars_ss.deliveredtoken = 0;
192     static bool spam = false;
193
194     E.link.mqtt_count++;
195     pubmsg.payload = msg;
196     pubmsg.payloadlen = strlen(msg);
197     pubmsg.qos = QOS;
198     pubmsg.retained = 0;
199
200     if (E.dl_excess) { // always run excess commands
201         spam = false;
202     }
203 #ifdef GTI_NO_POWER
204     MQTTClient_publishMessage(client_p, "mateq84/data/gticmd_nopower", &pubmsg, &token);
205 #else
206     if (bsoc_gti() > MIN_BAT_KW || E.dl_excess) {
207 #ifdef DEBUG_HA_CMD
208         log_time(true);
209         fprintf(fout, "HA GTI power command %s, SDEV %5.2f trace %u\r\n", msg, get_batc_dev(), trace);
210         fflush(fout);
211         spam = true;
212 #endif
213         MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run power commands
214     } else {
215         ret = false;
216         pubmsg.payload = "Z#";
217         pubmsg.payloadlen = strlen("Z#");
218         if (!spam) {
219             MQTTClient_publishMessage(client_p, TOPIC_SPAM, &pubmsg, &token);
220         } else {
221             MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // only shutdown GTI power
222         }
223     }
224 }
```



```

223 #ifdef DEBUG_HA_CMD
224     if (spam) {
225         log_time(true);
226         fprintf(fout, "HA GTI power set to zero, trace %u\r\n", trace);
227         fflush(fout);
228         spam = false;
229     }
230 #endif
231 }
232 #endif
233 // a busy, wait loop for the async delivery thread to complete
234 {
235     uint32_t waiting = 0;
236     while (ha_flag_vars_ss.deliveredtoken != token) {
237         usleep(TOKEN_DELAY);
238         if (waiting++ > MQTT_TIMEOUT) {
239             fprintf(fout, "\r\n%s GTI Power Still Waiting, timeout\r\n", log_time(false));
240             break;
241         }
242     };
243 }
244 usleep(HA_SW_DELAY);
245 return ret;
246 }

```

4.27.2.2 mqtt_gti_time()

```

bool mqtt_gti_time (
    MQTTClient client_p,
    const char * topic_p,
    char * msg )
249 {
250     bool ret = true;
251     MQTTClient_message pubmsg = MQTTClient_message_initializer;
252     MQTTClient_deliveryToken token;
253     ha_flag_vars_ss.deliveredtoken = 0;
254
255     E.link.mqtt_count++;
256     pubmsg.payload = msg;
257     pubmsg.payloadlen = strlen(msg);
258     pubmsg.qos = QOS;
259     pubmsg.retained = 0;
260
261     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run time commands
262
263     // a busy, wait loop for the async delivery thread to complete
264     {
265         uint32_t waiting = 0;
266         while (ha_flag_vars_ss.deliveredtoken != token) {
267             usleep(GTI_TOKEN_DELAY);
268             if (waiting++ > MQTT_TIMEOUT) {
269                 fprintf(fout, "\r\n%s GTI Time Still Waiting, timeout\r\n", log_time(false));
270                 break;
271             }
272         };
273     }
274     usleep(HA_SW_DELAY);
275     return ret;
276 }

```

4.27.2.3 mqtt_ha_pid()

```

void mqtt_ha_pid (
    MQTTClient client_p,
    const char * topic_p )
46 {
47     cJSON *json;
48     time_t rawtime;
49

```

```

50 MQTTClient_message pubmsg = MQTTClient_message_initializer;
51 MQTTClient_deliveryToken token;
52 ha_flag_vars_ss.deliveredtoken = 0;
53
54 E.link.mqtt_count++;
55 E.mode.sequence++;
56 json = cJSON_CreateObject();
57 cJSON_AddStringToObject(json, "name", CLIENTID1);
58 cJSON_AddNumberToObject(json, "sequence", E.mode.sequence);
59 cJSON_AddNumberToObject(json, "mqtt_count", (double) E.link.mqtt_count);
60 cJSON_AddNumberToObject(json, "http_count", (double) E.link.iammeter_count);
61 cJSON_AddNumberToObject(json, "piderror", E.mode.error);
62 cJSON_AddNumberToObject(json, "totalsystem", E.mode.total_system);
63 cJSON_AddNumberToObject(json, "gtinet", E.mode.gti_dumpload);
64 cJSON_AddNumberToObject(json, "energy_state", (double) E.mode.E);
65 cJSON_AddNumberToObject(json, "run_state", (double) E.mode.R);
66 // correct for power sensed by GTI metering
67 E.mode.off_grid = (E.mvar[V_FLO] - (E.mvar[V_DPPV] * DL_AC_DC_EFF));
68 E.mode.off_grid = drive1_filter(E.mode.off_grid);
69 if (E.mode.off_grid < 0.0f) { // only see power removed from grid usage
70     E.mode.off_grid = 0.0f;
71 }
72 cJSON_AddNumberToObject(json, "off_grid", E.mode.off_grid);
73 cJSON_AddNumberToObject(json, "excess_mode", (double) E.dl_excess);
74 cJSON_AddStringToObject(json, "build_date", FW_Date);
75 cJSON_AddStringToObject(json, "build_time", FW_Time);
76 time(&rawtime);
77 cJSON_AddNumberToObject(json, "sequence_time", (double) rawtime);
78 // convert the cJSON object to a JSON string
79 char *json_str = cJSON_Print(json);
80
81 pubmsg.payload = json_str;
82 pubmsg.payloadlen = strlen(json_str);
83 pubmsg.qos = QOS;
84 pubmsg.retained = 0;
85
86 MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
87 // a busy, wait loop for the async delivery thread to complete
88 {
89     uint32_t waiting = 0;
90     while (ha_flag_vars_ss.deliveredtoken != token) {
91         usleep(TOKEN_DELAY);
92         if (waiting++ > MQTT_TIMEOUT) {
93             fprintf(fout, "\r\n%s SW Still Waiting, timeout\r\n", log_time(false));
94             break;
95         }
96     };
97 }
98
99 cJSON_free(json_str);
100 cJSON_Delete(json);
101 }

```

4.27.2.4 mqtt_ha_shutdown()

```

void mqtt_ha_shutdown (
    MQTTClient client_p,
    const char * topic_p )
13 {
14     cJSON *json;
15     MQTTClient_message pubmsg = MQTTClient_message_initializer;
16     MQTTClient_deliveryToken token;
17     ha_flag_vars_ss.deliveredtoken = 0;
18
19     json = cJSON_CreateObject();
20     cJSON_AddStringToObject(json, "shutdown", CLIENTID1);
21     char *json_str = cJSON_Print(json);
22
23     pubmsg.payload = json_str;
24     pubmsg.payloadlen = strlen(json_str);
25     pubmsg.qos = QOS;
26     pubmsg.retained = 0;
27
28     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
29     // a busy, wait loop for the async delivery thread to complete
30     {
31         uint32_t waiting = 0;
32         while (ha_flag_vars_ss.deliveredtoken != token) {
33             usleep(TOKEN_DELAY);

```

```

34         if (waiting++ > MQTT_TIMEOUT) {
35             fprintf(fout, "\r\n%s SW Still Waiting, timeout\r\n", log_time(false));
36             break;
37         }
38     };
39 }
40 }

```

4.27.2.5 mqtt_ha_switch()

```

void mqtt_ha_switch (
    MQTTClient client_p,
    const char * topic_p,
    const bool sw_state )
107 {
108     cJSON *json;
109 #ifdef DEBUG_HA_CMD
110     static bool spam = false;
111     static uint32_t less_spam = 0;
112 #endif
113     MQTTClient_message pubmsg = MQTTClient_message_initializer;
114     MQTTClient_deliveryToken token;
115     ha_flag_vars_ss.deliveredtoken = 0;
116
117     E.link.mqtt_count++;
118     json = cJSON_CreateObject();
119     if (sw_state) {
120         cJSON_AddStringToObject(json, "state", "ON");
121     #ifdef DEBUG_HA_CMD
122         spam = true;
123         less_spam = 0;
124     #endif
125     } else {
126         if ((uint32_t) E.mvar[V_FCCM] != FLOAT_CODE) { // use max power in FLOAT mode
127             cJSON_AddStringToObject(json, "state", "OFF");
128         } else {
129             cJSON_AddStringToObject(json, "state", "ON");
130         }
131     #ifdef DEBUG_HA_CMD
132         spam = true;
133         less_spam = 0;
134     #endif
135     }
136     // convert the cJSON object to a JSON string
137     char *json_str = cJSON_Print(json);
138     pubmsg.payload = json_str;
139     pubmsg.payloadlen = strlen(json_str);
140     pubmsg.qos = QOS;
141     pubmsg.retained = 0;
142
143 #ifdef DEBUG_HA_CMD
144     if (spam) {
145         log_time(true);
146         fflush(fout);
147         fprintf(fout, "HA switch command %s, %d\r\n", topic_p, sw_state);
148         fflush(fout);
149         if (!sw_state) {
150             if (less_spam++ > 3) {
151                 spam = false;
152                 less_spam = 0;
153             }
154         }
155     }
156 #endif
157     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
158     // a busy, wait loop for the async delivery thread to complete
159     {
160         uint32_t waiting = 0;
161         while (ha_flag_vars_ss.deliveredtoken != token) {
162             usleep(TOKEN_DELAY);
163             if (waiting++ > MQTT_TIMEOUT) {
164                 #ifdef DEBUG_HA_CMD
165                     fflush(fout);
166                     fprintf(fout, "\r\nSW Still Waiting, timeout\r\n");
167                     fflush(fout);

```

```

171 #endif
172         break;
173     }
174 };
175 }
176
177 cJSON_free(json_str);
178 cJSON_Delete(json);
179 usleep(HA_SW_DELAY);
180 fflush(fout);
181 }

```

4.27.3 Variable Documentation

4.27.3.1 mqtt_name

```
const char* mqtt_name[V_DLAST] [extern]
```

4.28 mqtt_vars.h

[Go to the documentation of this file.](#)

```

1 /*
2  * File:      mqtt_vars.h
3  * Author:   root
4  *
5  * Created on February 9, 2024, 6:50 AM
6  */
7
8 #ifndef MQTT_VARS_H
9 #define MQTT_VARS_H
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14
15     // #define GTI_NO_POWER      // do we actually run power commands
16
17     // #define DEBUG_HA_CMD      // show debug text
18 #define HA_SW_DELAY      400000 // usecs
19 #define TOKEN_DELAY      250
20 #define GTI_TOKEN_DELAY 300
21
22 #define QOS      1
23
24     extern const char* mqtt_name[V_DLAST];
25
26     void mqtt_ha_switch(MQTTClient, const char *, const bool);
27     void mqtt_ha_pid(MQTTClient, const char *);
28     void mqtt_ha_shutdown(MQTTClient, const char *);
29     bool mqtt_gti_power(MQTTClient, const char *, char *, uint32_t);
30     bool mqtt_gti_time(MQTTClient, const char *, char *);
31
32 #ifdef __cplusplus
33 }
34 #endif
35
36 #endif /* MQTT_VARS_H */
37

```

4.29 ha_energy/nbproject/private/c_standard_headers_indexer.c File Reference

```
#include <assert.h>
#include <ctype.h>
#include <errno.h>
#include <float.h>
#include <limits.h>
#include <locale.h>
#include <math.h>
#include <setjmp.h>
#include <signal.h>
#include <stdarg.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <iso646.h>
#include <wchar.h>
#include <wctype.h>
```

Include dependency graph for c_standard_headers_indexer.c:



4.30 ha_energy/nbproject/private/cpp_standard_headers_indexer.cpp File Reference

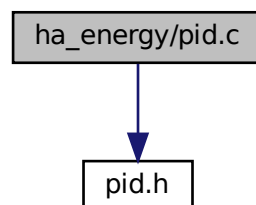
```
#include <cstdlib>
#include <csignal>
#include <csetjmp>
#include <cstdarg>
#include <typeinfo>
#include <bitset>
#include <functional>
#include <utility>
#include <ctime>
#include <cstddef>
#include <new>
#include <memory>
#include <climits>
#include <cfloat>
#include <limits>
#include <exception>
#include <stdexcept>
#include <cassert>
#include <cerrno>
#include <cctype>
#include <cwctype>
#include <cstring>
```

```
#include <cwchar>
#include <string>
#include <vector>
#include <deque>
#include <list>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <algorithm>
#include <iterator>
#include <cmath>
#include <complex>
#include <valarray>
#include <numeric>
#include <iosfwd>
#include <ios>
#include <istream>
#include <ostream>
#include <iostream>
#include <fstream>
#include <sstream>
#include <strstream>
#include <iomanip>
#include <streambuf>
#include <cstdio>
#include <locale>
#include <clocale>
#include <ciso646>
```

4.31 ha_energy/pid.c File Reference

```
#include "pid.h"
```

Include dependency graph for pid.c:



Functions

- double [UpdatePI](#) (volatile struct [SPid](#) *const pid, double const error)
- void [ResetPI](#) (volatile struct [SPid](#) *const pid)

4.31.1 Function Documentation

4.31.1.1 ResetPI()

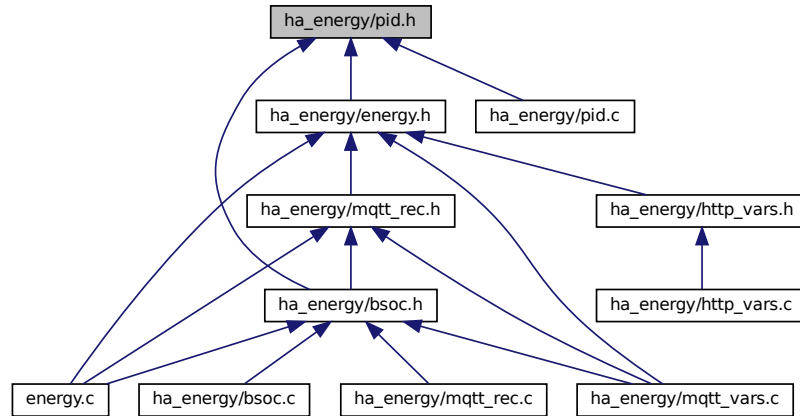
```
void ResetPI (
    volatile struct SPid *const pid )
30
31     pid->dState = 0.0; // not used but cleared
32     pid->iState = 0.0;
33 }
```

4.31.1.2 UpdatePI()

```
double UpdatePI (
    volatile struct SPid *const pid,
    double const error )
6
7     double pTerm, iTerm;
8
9     pTerm = pid->pGain * error; // calculate the proportional term
10    // calculate the integral state with appropriate limiting
11    pid->iState += error;
12
13    if (pid->iState > pid->iMax) {
14        pid->iState = pid->iMax;
15    } else if (pid->iState < pid->iMin) {
16        pid->iState = pid->iMin;
17    } else {
18
19    }
20
21    iTerm = (pid->iGain * pid->iState); // calculate the integral term
22
23    if ((pTerm + iTerm) > pid->iMax) {
24        iTerm = 0.0f;
25        pTerm = pid->iMax;
26    }
27    return pTerm + iTerm;
28 }
```

4.32 ha_energy/pid.h File Reference

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



Data Structures

- struct [SPid](#)

Functions

- double [UpdatePI](#) (volatile struct [SPid](#) *const, const double)
- void [ResetPI](#) (volatile struct [SPid](#) *const)

4.32.1 Function Documentation

4.32.1.1 ResetPI()

```

void ResetPI (
    volatile struct SPid * const pid )
30     {
31     pid->dState = 0.0; // not used but cleared
32     pid->iState = 0.0;
33 }

```


4.32.1.2 UpdatePI()

```

double UpdatePI (
    volatile struct SPid * const pid,
    const double error )
6
7     double pTerm, iTerm;
8
9     pTerm = pid->pGain * error; // calculate the proportional term
10    // calculate the integral state with appropriate limiting
11    pid->iState += error;
12
13    if (pid->iState > pid->iMax) {
14        pid->iState = pid->iMax;
15    } else if (pid->iState < pid->iMin) {
16        pid->iState = pid->iMin;
17    } else {
18
19    }
20
21    iTerm = (pid->iGain * pid->iState); // calculate the integral term
22
23    if ((pTerm + iTerm) > pid->iMax) {
24        iTerm = 0.0f;
25        pTerm = pid->iMax;
26    }
27    return pTerm + iTerm;
28 }

```

4.33 pid.h

[Go to the documentation of this file.](#)

```

1 /*
2  * File:      pid.h
3  * Author:   root
4  *
5  * Created on March 6, 2024, 7:03 AM
6  */
7
8 #ifndef PID_H
9 #define PID_H
10
11 #ifdef __cplusplus
12 extern "C" {
13 #endif
14
15     struct SPid {
16         double dState; // Last position input
17         double iState; // Integrator state
18         double iMax, iMin; // Maximum and minimum allowable integrator state
19         double iGain, // integral gain
20         pGain, // proportional gain
21         dGain; // derivative gain
22     };
23
24     double UpdatePI(volatile struct SPid * const, const double);
25     void ResetPI(volatile struct SPid * const);
26
27 #ifdef __cplusplus
28 }
29 #endif
30 #endif
31
32 #endif /* PID_H */
33

```


Index

`_DEFAULT_SOURCE`
 [energy.c, 22](#)
 [mqtt_vars.c, 102](#)

`ac0_filter`
 [bsoc.c, 37](#)
 [bsoc.h, 48](#)
`ac1_filter`
 [bsoc.c, 38](#)
 [bsoc.h, 48](#)
`ac2_filter`
 [bsoc.c, 38](#)
 [bsoc.h, 48](#)
`ac_low_adj`
 [energy_type, 6](#)
`ac_mismatch`
 [energy_type, 6](#)
`ac_sw_on`
 [energy_type, 6](#)
`ac_sw_status`
 [energy_type, 7](#)
`ac_test`
 [bsoc.c, 38](#)
 [bsoc.h, 48](#)
`ac_weight`
 [local_type, 14](#)
`ADDRESS`
 [energy.h, 59](#)

`BAL_MAX_ENERGY_AC`
 [energy.h, 59](#)
`BAL_MAX_ENERGY_GTI`
 [energy.h, 59](#)
`BAL_MIN_ENERGY_AC`
 [energy.h, 59](#)
`BAL_MIN_ENERGY_GTI`
 [energy.h, 60](#)
`BAMPS_SANE`
 [energy.h, 60](#)
`BAT_C_DRAW`
 [bsoc.h, 46](#)
`bat_c_std_dev`
 [local_type, 14](#)
`bat_crit`
 [mode_type, 16](#)
`BAT_CRITICAL`
 [energy.h, 60](#)
`bat_current`
 [local_type, 14](#)
`bat_current_stable`

[bsoc.c, 38](#)
 [bsoc.h, 48](#)
`BAT_M_KW`
 [energy.h, 60](#)
`BAT_SOC_HIGH`
 [energy.h, 60](#)
`BAT_SOC_LOW`
 [energy.h, 60](#)
`BAT_SOC_LOW_AC`
 [energy.h, 60](#)
`BAT_SOC_TOP`
 [energy.h, 60](#)
`bat_voltage`
 [local_type, 14](#)
`batc_std_dev`
 [local_type, 14](#)
`board_name`
 [energy.c, 35](#)
`bsoc.c`
 [ac0_filter, 37](#)
 [ac1_filter, 38](#)
 [ac2_filter, 38](#)
 [ac_test, 38](#)
 [bat_current_stable, 38](#)
 [bsoc_ac, 38](#)
 [bsoc_data_collect, 39](#)
 [bsoc_gti, 39](#)
 [bsoc_init, 40](#)
 [bsoc_set_mode, 40](#)
 [bsoc_set_std_dev, 41](#)
 [calculateStandardDeviation, 42](#)
 [dc0_filter, 42](#)
 [dc1_filter, 42](#)
 [dc2_filter, 42](#)
 [drive0_filter, 43](#)
 [drive1_filter, 43](#)
 [error_filter, 43](#)
 [get_batc_dev, 43](#)
 [gti_test, 43](#)
 [L, 44](#)
 [mqtt_name, 44](#)
`bsoc.h`
 [ac0_filter, 48](#)
 [ac1_filter, 48](#)
 [ac2_filter, 48](#)
 [ac_test, 48](#)
 [BAT_C_DRAW, 46](#)
 [bat_current_stable, 48](#)
 [bsoc_ac, 49](#)

- bsoc_data_collect, 49
- bsoc_gti, 50
- bsoc_init, 50
- bsoc_set_mode, 50
- bsoc_set_std_dev, 52
- calculateStandardDeviation, 52
- COEF, 46
- COEFF, 46
- COEFN, 46
- dc0_filter, 52
- dc1_filter, 52
- dc2_filter, 52
- DEV_SIZE, 46
- drive0_filter, 53
- drive1_filter, 53
- get_batc_dev, 53
- gti_test, 53
- MAX_BATC_DEV, 46
- MIN_BAT_KW, 47
- MIN_BAT_VOLTS, 47
- MIN_PV_VOLTS, 47
- PBAL_OFFSET, 47
- PV_FULL_PWR, 47
- PV_MIN_PWR, 47
- PV_V_FAKE, 47
- PV_V_NOM, 47
- bsoc_ac
 - bsoc.c, 38
 - bsoc.h, 49
- bsoc_data_collect
 - bsoc.c, 39
 - bsoc.h, 49
- bsoc_gti
 - bsoc.c, 39
 - bsoc.h, 50
- bsoc_init
 - bsoc.c, 40
 - bsoc.h, 50
- bsoc_set_mode
 - bsoc.c, 40
 - bsoc.h, 50
- bsoc_set_std_dev
 - bsoc.c, 41
 - bsoc.h, 52
- calculateStandardDeviation
 - bsoc.c, 42
 - bsoc.h, 52
- client_ha
 - energy_type, 7
- client_p
 - energy_type, 7
- client_sd
 - energy_type, 7
- CLIENTID1
 - energy.h, 61
- CLIENTID2
 - energy.h, 61
- CLIENTID3
 - energy.h, 61
- CMD_SEC
 - energy.h, 61
- COEF
 - bsoc.h, 46
- coef
 - local_type, 14
- COEFF
 - bsoc.h, 46
- COEFN
 - bsoc.h, 46
- con0
 - mode_type, 16
- con1
 - mode_type, 16
- con2
 - mode_type, 16
- con3
 - mode_type, 17
- con4
 - mode_type, 17
- con5
 - mode_type, 17
- con6
 - mode_type, 17
- con7
 - mode_type, 17
- connlost
 - energy.c, 22
 - energy.h, 74
- CRITIAL_SHUTDOWN_LOG
 - energy.h, 61
- curl
 - http_vars.c, 89
- DAQ_STR
 - energy.h, 61
- DAQ_STR_M
 - energy.h, 61
- data_error
 - mode_type, 17
- dc0_filter
 - bsoc.c, 42
 - bsoc.h, 52
- dc1_filter
 - bsoc.c, 42
 - bsoc.h, 52
- dc2_filter
 - bsoc.c, 42
 - bsoc.h, 52
- dc_mismatch
 - energy_type, 7
- delivered
 - mqtt_rec.c, 91
 - mqtt_rec.h, 97
- deliveredtoken
 - ha_flag_type, 12
- DEV_SIZE
 - bsoc.h, 46

- dGain
 - SPid, [20](#)
- DL_AC_DC_EFF
 - energy.h, [61](#)
- dl_excess
 - energy_type, [7](#)
- dl_excess_adj
 - energy_type, [7](#)
- dl_mqtt_max
 - mode_type, [17](#)
- drive0_filter
 - bsoc.c, [43](#)
 - bsoc.h, [53](#)
- drive1_filter
 - bsoc.c, [43](#)
 - bsoc.h, [53](#)
- driver_name
 - energy.c, [35](#)
- dState
 - SPid, [20](#)
- dumpload
 - energy_type, [7](#)
- DUMPLOAD_ID
 - mqtt_rec.h, [97](#)
- E
 - energy.c, [35](#)
 - energy.h, [81](#)
 - mode_type, [17](#)
- E_IDLE
 - energy.h, [71](#)
- E_INIT
 - energy.h, [71](#)
- E_LAST
 - energy.h, [71](#)
- E_RUN
 - energy.h, [71](#)
- E_STOP
 - energy.h, [71](#)
- E_WAIT
 - energy.h, [71](#)
- energy.c, [21](#)
 - _DEFAULT_SOURCE, [22](#)
 - board_name, [35](#)
 - connlost, [22](#)
 - driver_name, [35](#)
 - E, [35](#)
 - fout, [35](#)
 - ha_ac_off, [22](#)
 - ha_ac_on, [22](#)
 - ha_dc_off, [23](#)
 - ha_dc_on, [23](#)
 - ha_flag_vars_ha, [35](#)
 - ha_flag_vars_pc, [36](#)
 - ha_flag_vars_sd, [36](#)
 - ha_flag_vars_ss, [36](#)
 - log_time, [23](#)
 - log_timer, [23](#)
 - main, [24](#)
 - ramp_down_ac, [30](#)
 - ramp_down_gti, [30](#)
 - ramp_up_ac, [31](#)
 - ramp_up_gti, [31](#)
 - sanity_check, [32](#)
 - showIP, [32](#)
 - skeleton_daemon, [33](#)
 - solar_shutdown, [33](#)
 - sync_ha, [34](#)
 - timer_callback, [35](#)
- energy.h
 - ADDRESS, [59](#)
 - BAL_MAX_ENERGY_AC, [59](#)
 - BAL_MAX_ENERGY_GTI, [59](#)
 - BAL_MIN_ENERGY_AC, [59](#)
 - BAL_MIN_ENERGY_GTI, [60](#)
 - BAMPS_SANE, [60](#)
 - BAT_CRITICAL, [60](#)
 - BAT_M_KW, [60](#)
 - BAT_SOC_HIGH, [60](#)
 - BAT_SOC_LOW, [60](#)
 - BAT_SOC_LOW_AC, [60](#)
 - BAT_SOC_TOP, [60](#)
 - CLIENTID1, [61](#)
 - CLIENTID2, [61](#)
 - CLIENTID3, [61](#)
 - CMD_SEC, [61](#)
 - connlost, [74](#)
 - CRITICAL_SHUTDOWN_LOG, [61](#)
 - DAQ_STR, [61](#)
 - DAQ_STR_M, [61](#)
 - DL_AC_DC_EFF, [61](#)
 - E, [81](#)
 - E_IDLE, [71](#)
 - E_INIT, [71](#)
 - E_LAST, [71](#)
 - E_RUN, [71](#)
 - E_STOP, [71](#)
 - E_WAIT, [71](#)
 - energy_state, [70](#)
 - fout, [81](#)
 - GTI_DELAY, [62](#)
 - ha_ac_off, [75](#)
 - ha_ac_on, [75](#)
 - ha_dc_off, [75](#)
 - ha_dc_on, [75](#)
 - ha_flag_vars_ss, [81](#)
 - IA_CURRENT, [71](#)
 - IA_EXPORT, [71](#)
 - IA_FREQ, [71](#)
 - IA_IMPORT, [71](#)
 - IA_LAST, [71](#)
 - IA_PF, [71](#)
 - IA_POWER, [71](#)
 - IA_VOLTAGE, [71](#)
 - IAM_DELAY, [62](#)
 - iammeter_id, [71](#)
 - iammeter_phase, [71](#)

iammeter_read, 76
iammeter_write_callback, 76
IM_DELAY, 62
IM_DISPLAY, 62
L1_P, 62
L2_P, 62
L3_P, 62
LADDRESS, 62
log_time, 77
log_timer, 77
LOG_TO_FILE, 63
LOG_TO_FILE_ALT, 63
LOG_VERSION, 63
LOW_LOG_SPAM, 63
MAX_ERROR, 63
MAX_IM_VAR, 63
MAX_LOG_SPAM, 63
MIN_BAT_KW_AC_HI, 63
MIN_BAT_KW_AC_LO, 64
MIN_BAT_KW_BSOC_HI, 64
MIN_BAT_KW_BSOC_SLP, 64
MIN_BAT_KW_GTI_HI, 64
MIN_BAT_KW_GTI_LO, 64
MQTT_TIMEOUT, 64
mqtt_vars, 72
MQTT_VERSION, 64
NO_CYLON, 64
NORM_MODE, 65
PAMPS_SANE, 65
PHASE_A, 71
PHASE_B, 71
PHASE_C, 71
PHASE_LAST, 71
PID_MODE, 65
print_im_vars, 78
print_mvar_vars, 78
PV_BIAS, 65
PV_BIAS_FLOAT, 65
PV_BIAS_LOW, 65
PV_BIAS_RATE, 65
PV_BIAS_SLEEP, 65
PV_BIAS_ZERO, 66
PV_DL_B_AH_LOW, 66
PV_DL_B_AH_MIN, 66
PV_DL_B_V_LOW, 66
PV_DL_BIAS_RATE, 66
PV_DL_EXCESS, 66
PV_DL_MPTT_EXCESS, 66
PV_DL_MPTT_IDLE, 66
PV_DL_MPTT_MAX, 67
PV_IGAIN, 67
PV_IMAX, 67
PV_PGAIN, 67
PVOLTS_SANE, 67
PWA_SANE, 67
PWA_SLEEP, 67
QOS, 67
R_FLOAT, 73
R_IDLE, 73
R_INIT, 73
R_LAST, 73
R_RUN, 73
R_SLEEP, 73
ramp_down_ac, 78
ramp_down_gti, 78
ramp_up_ac, 79
ramp_up_gti, 79
RBUF_SIZ, 68
RESET_LOG_SPAM, 68
running_state, 73
S_BEN, 74
S_DAHBAT, 74
S_DCCMODE, 74
S_DCMPPPT, 74
S_DGTI, 74
S_DLAST, 74
S_DPBAT, 74
S_DPMPPT, 74
S_DPPV, 74
S_DVBAT, 74
S_DVPV, 74
S_FACE, 74
S_FBAMPS, 74
S_FBEKW, 74
S_FBV, 74
S_FCCM, 74
S_FLAST, 74
S_FLO, 74
S_FRUNT, 74
S_FSO, 74
S_HACSW, 74
S_HDCSW, 74
S_HMODE, 74
S_HSHUT, 74
S_PAMPS, 74
S_PVOLTS, 74
S_PWA, 74
sane_vars, 73
sanity_check, 80
SBUF_SIZ, 68
SPACING_USEC, 68
SW_QOS, 68
sync_ha, 80
SYSLOG_SIZ, 68
TIME_SYNC_SEC, 68
TIMEOUT, 68
timer_callback, 81
TNAME, 69
TOPIC_HA, 69
TOPIC_P, 69
TOPIC_PACA, 69
TOPIC_PACC, 69
TOPIC_PDCA, 69
TOPIC_PDCC, 69
TOPIC_PPID, 69
TOPIC_SD, 70

- TOPIC_SHUTDOWN, 70
- TOPIC_SPAM, 70
- TOPIC_SS, 70
- UNIT_TEST, 70
- USEC_SEC, 70
- V_BEN, 72
- V_DAHBAT, 72
- V_DCCMODE, 72
- V_DCMPTT, 72
- V_DGTI, 72
- V_DLAST, 72
- V_DPBAT, 72
- V_DPMPTT, 72
- V_DPPV, 72
- V_DVBAT, 72
- V_DVPV, 72
- V_FACE, 72
- V_FBAMPS, 72
- V_FBEKW, 72
- V_FBV, 72
- V_FCCM, 72
- V_FLAST, 72
- V_FLO, 72
- V_FRUNT, 72
- V_FSO, 72
- V_HACSW, 72
- V_HCON0, 72
- V_HCON1, 72
- V_HCON2, 72
- V_HCON3, 72
- V_HCON4, 72
- V_HCON5, 72
- V_HCON6, 72
- V_HCON7, 72
- V_HDCSW, 72
- V_HMODE, 72
- V_HSHUT, 72
- V_PAMPS, 72
- V_PVOLTS, 72
- V_PWA, 72
- energy_mode
 - ha_flag_type, 12
- energy_state
 - energy.h, 70
- energy_type, 5
 - ac_low_adj, 6
 - ac_mismatch, 6
 - ac_sw_on, 6
 - ac_sw_status, 7
 - client_ha, 7
 - client_p, 7
 - client_sd, 7
 - dc_mismatch, 7
 - dl_excess, 7
 - dl_excess_adj, 7
 - dumpload, 7
 - fm80, 8
 - gti_delay, 8
 - gti_low_adj, 8
 - gti_sw_on, 8
 - gti_sw_status, 8
 - ha_lock, 8
 - homeassistant, 8
 - iammeter, 8
 - im_delay, 9
 - im_display, 9
 - im_vars, 9
 - link, 9
 - log_spam, 9
 - log_time_reset, 9
 - mode, 9
 - mode_mismatch, 9
 - mvar, 10
 - once_ac, 10
 - once_gti, 10
 - once_gti_zero, 10
 - print_vars, 10
 - rc, 10
 - sane, 10
 - solar_mode, 10
 - solar_shutdown, 11
 - speed_go, 11
 - startup, 11
 - ten_sec_clock, 11
- error
 - mode_type, 18
- error_filter
 - bsoc.c, 43
- FLOAT_CODE
 - mqtt_rec.h, 96
- fm80
 - energy_type, 8
- fm80_float
 - mqtt_rec.c, 91
 - mqtt_rec.h, 97
- FM80_ID
 - mqtt_rec.h, 97
- fm80_sleep
 - mqtt_rec.c, 91
 - mqtt_rec.h, 97
- fout
 - energy.c, 35
 - energy.h, 81
 - http_vars.h, 90
 - mqtt_rec.h, 100
- FW_Date
 - mqtt_vars.c, 106
- FW_Time
 - mqtt_vars.c, 106
- get_batc_dev
 - bsoc.c, 43
 - bsoc.h, 53
- GTI_DELAY
 - energy.h, 62
- gti_delay

- energy_type, 8
- gti_dumpload
 - mode_type, 18
- gti_low_adj
 - energy_type, 8
- gti_sw_on
 - energy_type, 8
- gti_sw_status
 - energy_type, 8
- gti_test
 - bsoc.c, 43
 - bsoc.h, 53
- GTI_TOKEN_DELAY
 - mqtt_vars.h, 107
- gti_weight
 - local_type, 15
- ha_ac_off
 - energy.c, 22
 - energy.h, 75
- ha_ac_on
 - energy.c, 22
 - energy.h, 75
- ha_dc_off
 - energy.c, 23
 - energy.h, 75
- ha_dc_on
 - energy.c, 23
 - energy.h, 75
- ha_energy/.dep.inc, 37
- ha_energy/bsoc.c, 37
- ha_energy/bsoc.h, 44, 54
- ha_energy/build/Debug/GNU-Linux/_ext/5c0/energy.o.d, 55
- ha_energy/build/Debug/GNU-Linux/bsoc.o.d, 55
- ha_energy/build/Debug/GNU-Linux/http_vars.o.d, 55
- ha_energy/build/Debug/GNU-Linux/mqtt_rec.o.d, 55
- ha_energy/build/Debug/GNU-Linux/mqtt_vars.o.d, 55
- ha_energy/build/Debug/GNU-Linux/pid.o.d, 55
- ha_energy/build/Release/GNU-Linux/_ext/5c0/energy.o.d, 55
- ha_energy/build/Release/GNU-Linux/bsoc.o.d, 55
- ha_energy/build/Release/GNU-Linux/http_vars.o.d, 55
- ha_energy/build/Release/GNU-Linux/mqtt_rec.o.d, 55
- ha_energy/build/Release/GNU-Linux/mqtt_vars.o.d, 55
- ha_energy/build/Release/GNU-Linux/pid.o.d, 55
- ha_energy/energy.h, 55, 82
- ha_energy/http_vars.c, 86
- ha_energy/http_vars.h, 89, 90
- ha_energy/mqtt_rec.c, 91
- ha_energy/mqtt_rec.h, 95, 101
- ha_energy/mqtt_vars.c, 101
- ha_energy/mqtt_vars.h, 107, 112
- ha_energy/nbproject/private/c_standard_headers_indexer.c, 113
- ha_energy/nbproject/private/cpp_standard_headers_indexer.cpp, 113
- ha_energy/pid.c, 114
- ha_energy/pid.h, 116, 117
- ha_flag_type, 11
 - deliveredtoken, 12
 - energy_mode, 12
 - ha_id, 12
 - rec_ok, 12
 - receivedtoken, 12
 - runner, 12
 - var_update, 12
- ha_flag_vars_ha
 - energy.c, 35
- ha_flag_vars_pc
 - energy.c, 36
- ha_flag_vars_sd
 - energy.c, 36
- ha_flag_vars_ss
 - energy.c, 36
 - energy.h, 81
- HA_ID
 - mqtt_rec.h, 97
- ha_id
 - ha_flag_type, 12
- ha_lock
 - energy_type, 8
- HA_SW_DELAY
 - mqtt_vars.h, 107
- homeassistant
 - energy_type, 8
- http_vars.c
 - curl, 89
 - iammeter_get_data, 86
 - iammeter_read, 87
 - iammeter_write_callback, 87
 - mqtt_gti_time, 88
 - print_im_vars, 88
 - res, 89
- http_vars.h
 - fout, 90
- IA_CURRENT
 - energy.h, 71
- IA_EXPORT
 - energy.h, 71
- IA_FREQ
 - energy.h, 71
- IA_IMPORT
 - energy.h, 71
- IA_LAST
 - energy.h, 71
- IA_PF
 - energy.h, 71
- IA_POWER
 - energy.h, 71
- IA_VOLTAGE
 - energy.h, 71
- IAM_DELAY
 - energy.h, 62
- iammeter
 - energy_type, 8
- iammeter_count

- link_type, 13
- iammeter_error
 - link_type, 13
- iammeter_get_data
 - http_vars.c, 86
- iammeter_id
 - energy.h, 71
- iammeter_phase
 - energy.h, 71
- iammeter_read
 - energy.h, 76
 - http_vars.c, 87
- iammeter_write_callback
 - energy.h, 76
 - http_vars.c, 87
- iGain
 - SPid, 20
- IM_DELAY
 - energy.h, 62
- im_delay
 - energy_type, 9
- IM_DISPLAY
 - energy.h, 62
- im_display
 - energy_type, 9
- im_vars
 - energy_type, 9
- iMax
 - SPid, 20
- iMin
 - SPid, 20
- in_pid_control
 - mode_type, 18
- iState
 - SPid, 20
- json_get_data
 - mqtt_rec.c, 92
 - mqtt_rec.h, 98
- L
 - bsoc.c, 44
- L1_P
 - energy.h, 62
- L2_P
 - energy.h, 62
- L3_P
 - energy.h, 62
- LADDRESS
 - energy.h, 62
- LAST_MQTT_ID
 - mqtt_rec.h, 97
- link
 - energy_type, 9
- link_type, 13
 - iammeter_count, 13
 - iammeter_error, 13
 - mqtt_count, 13
 - mqtt_error, 13
 - shutdown, 13
- local_type, 14
 - ac_weight, 14
 - bat_c_std_dev, 14
 - bat_current, 14
 - bat_voltage, 14
 - batc_std_dev, 14
 - coef, 14
 - gti_weight, 15
 - pv_voltage, 15
- log_spam
 - energy_type, 9
- log_time
 - energy.c, 23
 - energy.h, 77
- log_time_reset
 - energy_type, 9
- log_timer
 - energy.c, 23
 - energy.h, 77
- LOG_TO_FILE
 - energy.h, 63
- LOG_TO_FILE_ALT
 - energy.h, 63
- LOG_VERSION
 - energy.h, 63
- LOW_LOG_SPAM
 - energy.h, 63
- main
 - energy.c, 24
- MAX_BATC_DEV
 - bsoc.h, 46
- MAX_ERROR
 - energy.h, 63
- MAX_IM_VAR
 - energy.h, 63
- MAX_LOG_SPAM
 - energy.h, 63
- MBMQTT
 - mqtt_rec.h, 96
- MIN_BAT_KW
 - bsoc.h, 47
- MIN_BAT_KW_AC_HI
 - energy.h, 63
- MIN_BAT_KW_AC_LO
 - energy.h, 64
- MIN_BAT_KW_BSOC_HI
 - energy.h, 64
- MIN_BAT_KW_BSOC_SLP
 - energy.h, 64
- MIN_BAT_KW_GTI_HI
 - energy.h, 64
- MIN_BAT_KW_GTI_LO
 - energy.h, 64
- MIN_BAT_VOLTS
 - bsoc.h, 47
- MIN_PV_VOLTS
 - bsoc.h, 47

- mode
 - energy_type, 9
 - mode_type, 18
- mode_mismatch
 - energy_type, 9
- mode_tmr
 - mode_type, 18
- mode_type, 15
 - bat_crit, 16
 - con0, 16
 - con1, 16
 - con2, 16
 - con3, 17
 - con4, 17
 - con5, 17
 - con6, 17
 - con7, 17
 - data_error, 17
 - dl_mqtt_max, 17
 - E, 17
 - error, 18
 - gti_dumpload, 18
 - in_pid_control, 18
 - mode, 18
 - mode_tmr, 18
 - no_float, 18
 - off_grid, 18
 - pid, 18
 - pv_bias, 19
 - R, 19
 - sequence, 19
 - target, 19
 - total_system, 19
- mqtt_count
 - link_type, 13
- mqtt_error
 - link_type, 13
- mqtt_gti_power
 - mqtt_vars.c, 102
 - mqtt_vars.h, 108
- mqtt_gti_time
 - http_vars.c, 88
 - mqtt_vars.c, 103
 - mqtt_vars.h, 109
- mqtt_ha_pid
 - mqtt_vars.c, 104
 - mqtt_vars.h, 109
- mqtt_ha_shutdown
 - mqtt_vars.c, 105
 - mqtt_vars.h, 110
- mqtt_ha_switch
 - mqtt_vars.c, 105
 - mqtt_vars.h, 111
- mqtt_id
 - mqtt_rec.h, 96
- mqtt_name
 - bsoc.c, 44
 - mqtt_vars.h, 112
- mqtt_rec.c
 - delivered, 91
 - fm80_float, 91
 - fm80_sleep, 91
 - json_get_data, 92
 - msgarrvd, 93
 - print_mvar_vars, 94
- mqtt_rec.h
 - delivered, 97
 - DUMPLOAD_ID, 97
 - FLOAT_CODE, 96
 - fm80_float, 97
 - FM80_ID, 97
 - fm80_sleep, 97
 - fout, 100
 - HA_ID, 97
 - json_get_data, 98
 - LAST_MQTT_ID, 97
 - MBMQTT, 96
 - mqtt_id, 96
 - msgarrvd, 99
 - P8055_ID, 97
 - RDEV_SIZE, 96
 - SLEEP_CODE, 96
- MQTT_TIMEOUT
 - energy.h, 64
- mqtt_vars
 - energy.h, 72
- mqtt_vars.c
 - _DEFAULT_SOURCE, 102
 - FW_Date, 106
 - FW_Time, 106
 - mqtt_gti_power, 102
 - mqtt_gti_time, 103
 - mqtt_ha_pid, 104
 - mqtt_ha_shutdown, 105
 - mqtt_ha_switch, 105
- mqtt_vars.h
 - GTI_TOKEN_DELAY, 107
 - HA_SW_DELAY, 107
 - mqtt_gti_power, 108
 - mqtt_gti_time, 109
 - mqtt_ha_pid, 109
 - mqtt_ha_shutdown, 110
 - mqtt_ha_switch, 111
 - mqtt_name, 112
 - QOS, 108
 - TOKEN_DELAY, 108
- MQTT_VERSION
 - energy.h, 64
- msgarrvd
 - mqtt_rec.c, 93
 - mqtt_rec.h, 99
- mvar
 - energy_type, 10
- NO_CYLON
 - energy.h, 64
- no_float

- mode_type, 18
- NORM_MODE
 - energy.h, 65
- off_grid
 - mode_type, 18
- once_ac
 - energy_type, 10
- once_gti
 - energy_type, 10
- once_gti_zero
 - energy_type, 10
- P8055_ID
 - mqtt_rec.h, 97
- PAMPS_SANE
 - energy.h, 65
- PBAL_OFFSET
 - bsoc.h, 47
- pGain
 - SPid, 20
- PHASE_A
 - energy.h, 71
- PHASE_B
 - energy.h, 71
- PHASE_C
 - energy.h, 71
- PHASE_LAST
 - energy.h, 71
- pid
 - mode_type, 18
- pid.c
 - ResetPI, 115
 - UpdatePI, 115
- pid.h
 - ResetPI, 116
 - UpdatePI, 116
- PID_MODE
 - energy.h, 65
- print_im_vars
 - energy.h, 78
 - http_vars.c, 88
- print_mvar_vars
 - energy.h, 78
 - mqtt_rec.c, 94
- print_vars
 - energy_type, 10
- PV_BIAS
 - energy.h, 65
- pv_bias
 - mode_type, 19
- PV_BIAS_FLOAT
 - energy.h, 65
- PV_BIAS_LOW
 - energy.h, 65
- PV_BIAS_RATE
 - energy.h, 65
- PV_BIAS_SLEEP
 - energy.h, 65
- PV_BIAS_ZERO
 - energy.h, 66
- PV_DL_B_AH_LOW
 - energy.h, 66
- PV_DL_B_AH_MIN
 - energy.h, 66
- PV_DL_B_V_LOW
 - energy.h, 66
- PV_DL_BIAS_RATE
 - energy.h, 66
- PV_DL_EXCESS
 - energy.h, 66
- PV_DL_MPTT_EXCESS
 - energy.h, 66
- PV_DL_MPTT_IDLE
 - energy.h, 66
- PV_DL_MPTT_MAX
 - energy.h, 67
- PV_FULL_PWR
 - bsoc.h, 47
- PV_IGAIN
 - energy.h, 67
- PV_IMAX
 - energy.h, 67
- PV_MIN_PWR
 - bsoc.h, 47
- PV_PGAIN
 - energy.h, 67
- PV_V_FAKE
 - bsoc.h, 47
- PV_V_NOM
 - bsoc.h, 47
- pv_voltage
 - local_type, 15
- PVOLTS_SANE
 - energy.h, 67
- PWA_SANE
 - energy.h, 67
- PWA_SLEEP
 - energy.h, 67
- QOS
 - energy.h, 67
 - mqtt_vars.h, 108
- R
 - mode_type, 19
- R_FLOAT
 - energy.h, 73
- R_IDLE
 - energy.h, 73
- R_INIT
 - energy.h, 73
- R_LAST
 - energy.h, 73
- R_RUN
 - energy.h, 73
- R_SLEEP
 - energy.h, 73

ramp_down_ac
 energy.c, 30
 energy.h, 78
 ramp_down_gti
 energy.c, 30
 energy.h, 78
 ramp_up_ac
 energy.c, 31
 energy.h, 79
 ramp_up_gti
 energy.c, 31
 energy.h, 79
 RBUF_SIZ
 energy.h, 68
 rc
 energy_type, 10
 RDEV_SIZE
 mqtt_rec.h, 96
 rec_ok
 ha_flag_type, 12
 receivedtoken
 ha_flag_type, 12
 res
 http_vars.c, 89
 RESET_LOG_SPAM
 energy.h, 68
 ResetPI
 pid.c, 115
 pid.h, 116
 runner
 ha_flag_type, 12
 running_state
 energy.h, 73

 S_BEN
 energy.h, 74
 S_DAHBAT
 energy.h, 74
 S_DCCMODE
 energy.h, 74
 S_DCMPTT
 energy.h, 74
 S_DGTI
 energy.h, 74
 S_DLAST
 energy.h, 74
 S_DPBAT
 energy.h, 74
 S_DPMPPT
 energy.h, 74
 S_DPPV
 energy.h, 74
 S_DVBAT
 energy.h, 74
 S_DVPV
 energy.h, 74
 S_FACE
 energy.h, 74
 S_FBAMPS
 energy.h, 74
 S_FBEKW
 energy.h, 74
 S_FBV
 energy.h, 74
 S_FCCM
 energy.h, 74
 S_FLAST
 energy.h, 74
 S_FLO
 energy.h, 74
 S_FRUNT
 energy.h, 74
 S_FSO
 energy.h, 74
 S_HACSW
 energy.h, 74
 S_HDCSW
 energy.h, 74
 S_HMODE
 energy.h, 74
 S_HSHUT
 energy.h, 74
 S_PAMPS
 energy.h, 74
 S_PVOLTS
 energy.h, 74
 S_PWA
 energy.h, 74
 sane
 energy_type, 10
 sane_vars
 energy.h, 73
 sanity_check
 energy.c, 32
 energy.h, 80
 SBUF_SIZ
 energy.h, 68
 sequence
 mode_type, 19
 showIP
 energy.c, 32
 shutdown
 link_type, 13
 skeleton_daemon
 energy.c, 33
 SLEEP_CODE
 mqtt_rec.h, 96
 solar_mode
 energy_type, 10
 solar_shutdown
 energy.c, 33
 energy_type, 11
 SPACING_USEC
 energy.h, 68
 speed_go
 energy_type, 11
 SPid, 19

- dGain, 20
- dState, 20
- iGain, 20
- iMax, 20
- iMin, 20
- iState, 20
- pGain, 20
- startup
 - energy_type, 11
- SW_QOS
 - energy.h, 68
- sync_ha
 - energy.c, 34
 - energy.h, 80
- SYSLOG_SIZ
 - energy.h, 68
- target
 - mode_type, 19
- ten_sec_clock
 - energy_type, 11
- TIME_SYNC_SEC
 - energy.h, 68
- TIMEOUT
 - energy.h, 68
- timer_callback
 - energy.c, 35
 - energy.h, 81
- TNAME
 - energy.h, 69
- TOKEN_DELAY
 - mqtt_vars.h, 108
- TOPIC_HA
 - energy.h, 69
- TOPIC_P
 - energy.h, 69
- TOPIC_PACA
 - energy.h, 69
- TOPIC_PACC
 - energy.h, 69
- TOPIC_PDCA
 - energy.h, 69
- TOPIC_PDCC
 - energy.h, 69
- TOPIC_PPID
 - energy.h, 69
- TOPIC_SD
 - energy.h, 70
- TOPIC_SHUTDOWN
 - energy.h, 70
- TOPIC_SPAM
 - energy.h, 70
- TOPIC_SS
 - energy.h, 70
- total_system
 - mode_type, 19
- UNIT_TEST
 - energy.h, 70
- UpdatePI
 - pid.c, 115
 - pid.h, 116
- USEC_SEC
 - energy.h, 70
- V_BEN
 - energy.h, 72
- V_DAHBAT
 - energy.h, 72
- V_DCCMODE
 - energy.h, 72
- V_DCMPPT
 - energy.h, 72
- V_DGTI
 - energy.h, 72
- V_DLAST
 - energy.h, 72
- V_DPBAT
 - energy.h, 72
- V_DPMPPT
 - energy.h, 72
- V_DPPV
 - energy.h, 72
- V_DVBAT
 - energy.h, 72
- V_DVPV
 - energy.h, 72
- V_FACE
 - energy.h, 72
- V_FBAMPS
 - energy.h, 72
- V_FBEKW
 - energy.h, 72
- V_FBV
 - energy.h, 72
- V_FCCM
 - energy.h, 72
- V_FLAST
 - energy.h, 72
- V_FLO
 - energy.h, 72
- V_FRUNT
 - energy.h, 72
- V_FSO
 - energy.h, 72
- V_HACSW
 - energy.h, 72
- V_HCON0
 - energy.h, 72
- V_HCON1
 - energy.h, 72
- V_HCON2
 - energy.h, 72
- V_HCON3
 - energy.h, 72
- V_HCON4
 - energy.h, 72
- V_HCON5

- energy.h, [72](#)
- V_HCON6
 - energy.h, [72](#)
- V_HCON7
 - energy.h, [72](#)
- V_HDCSW
 - energy.h, [72](#)
- V_HMODE
 - energy.h, [72](#)
- V_HSHUT
 - energy.h, [72](#)
- V_PAMPS
 - energy.h, [72](#)
- V_PVOLTS
 - energy.h, [72](#)
- V_PWA
 - energy.h, [72](#)
- var_update
 - ha_flag_type, [12](#)