HA Energy

Solar Project #1

Generated by Doxygen 1.14.0

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

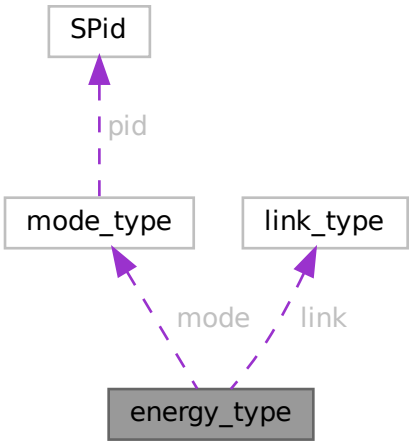
Here is a list of all documented files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 energy_type Struct Reference

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 double **bat_runtime**
- 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**

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

- ha_energy/energy.h

3.2 ha_flag_type Struct Reference

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**
- enum client_id **cid**

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

- ha_energy/mqtt_rec.h

3.3 link_type Struct Reference

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**

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

- ha_energy/energy.h

3.4 local_type Struct Reference

Data Fields

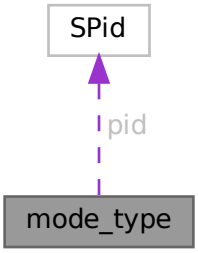
- volatile double **ac_weight**
- volatile double **gti_weight**
- volatile double **pv_voltage**
- volatile double **bat_current**
- volatile double **batc_std_dev**
- volatile double **bat_voltage**
- volatile double **bat_runtime**
- double **bat_c_std_dev** [DEV_SIZE]
- double **coef**

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

- ha_energy/bsoc.c

3.5 mode_type Struct Reference

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**

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

- ha_energy/energy.h

3.6 SPid Struct Reference

Data Fields

- double **dState**
- double **iState**
- double **iMax**
- double **iMin**
- double **iGain**
- double **pGain**
- double **dGain**

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

- ha_energy/pid.h

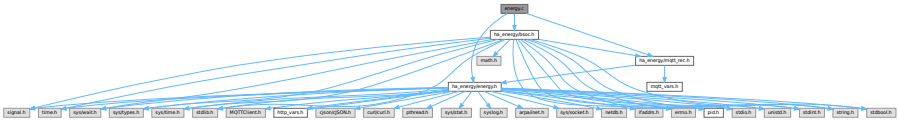
Chapter 4

File Documentation

4.1 energy.c File Reference

```
#include "ha_energy/energy.h"
#include "ha_energy/mqtt_rec.h"
#include "ha_energy/bsoc.h"
```

Include dependency graph for energy.c:



Functions

- 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 Detailed Description

V0.25 add Home Assistant Matter controlled utility power control switching V0.26 BSOC weights for system condition for power diversion V0.27 -> V0.28 GTI power ramps stability using battery current STD DEV V0.29 log date-time and spam control V0.30 add iammeter http data reading and processing V0.31 refactor http code and a few vars V0.32 AC and GTI power triggers reworked V0.33 refactor system parms into energy structure energy_type E V0.↩ 34 GTI and AC Inverter battery energy run down limits adjustments per energy usage and solar production V0.35 more refactors and global variable consolidation V0.36 more command repeat fixes for ramp up/down dumpload commands V0.37 Power feedback to use PV power to GTI and AC loads V0.38 signal filters to smooth large power swings in control optimization V0.39 fix optimizer bugs and add AC load switching set-points in BSOC control V0.↩ 40 shutdown and restart fixes V0.41 fix errors and warning per cppcheck V0.42 fake ac charger for dumpload using FAKE_VPV define V0.43 adjust PV_BIAS per float or charging status V0.44 tune for spring/summer solar conditions V0.50 convert main loop code to FSM V0.51 logging time additions V0.52 tune GTI inverter levels for better conversion efficiency V0.53 sync to HA back-end switch status V0.54 data source shutdown functions V0.55 off-grid inverter power tracking for HA V0.56 run as Daemon in background V0.62 adjust battery critical to keep making energy calculations V0.63 add IP address logging V0.64 Dump Load excess load mode programming V.065 DL excess logic tuning and power adjustments V.066 -> V.068 Various timing fixes to reduce spamming commands and logs V.069 send MQTT showdown commands to HA when critical energy conditions are meet V.070 process Home Assistant MQTT commands sent from automation's

V.071 comment additions, logging improvements and code cleanups V.072 -> V.073 fine tune GTI and AC power lower limits V.074 Doxygen comments added V.075 connection lost logging and Keep Alive fixes V.076 control setpoints tuning for main battery runtime limits

4.1.2 Function Documentation

4.1.2.1 connlost()

```
void connlost (
            void * context,
            char * cause)
```

trouble in River-city

```
00302 {
00303     struct ha_flag_type *ha_flag = context;
00304     int32_t id_num = ha_flag->ha_id;
00305     static uint32_t times = 0;
00306     char * where = "Context is NULL";
00307     char * what = "Reconnection Retry";
00308
00309     // bug-out if no context variables passed to callback
00310     if (context == NULL) {
00311         id_num = -1;
00312         goto bugout;
00313     }
00314
00315     switch (ha_flag->cid) {
```

```
00316     case ID_C1:
00317         where = TOPIC_SS;
00318         break;
00319     case ID_C2:
00320         where = TOPIC_SD;
00321         break;
00322     case ID_C3:
00323         where = TOPIC_HA;
00324         break;
00325     }
00326
00327
00328     if (times++ > MQTT_RECONN) {
00329         goto bugout;
00330     } else {
00331         if (times > 1) {
00332             fprintf(fout, "%s Connection lost, retrying %d \n", log_time(false), times);
00333             fprintf(fout, "%s Cause: %s, h_id %d, c_id %d, %s \n", log_time(false), cause, id_num,
      ha_flag->cid, what);
00334             fprintf(fout, "%s MQTT DAEMON reconnection failure  LOG Version %s : MQTT Version %s\n",
      log_time(false), LOG_VERSION, MQTT_VERSION);
00335         }
00336         fflush(fout);
00337         times = 0;
00338         return;
00339     }
00340
00341 bugout:
00342     fprintf(fout, "%s Connection lost, exit ha_energy program\n", log_time(false));
00343     fprintf(fout, "%s Cause: %s, h_id %d, c_id %d, %s \n", log_time(false), cause, id_num,
      ha_flag->cid, where);
00344     fprintf(fout, "%s MQTT DAEMON context is null failure  LOG Version %s : MQTT Version %s\n",
      log_time(false), LOG_VERSION, MQTT_VERSION);
00345     fflush(fout);
00346     exit(EXIT_FAILURE);
00347 }
```

4.1.2.2 ha_ac_off()

```
void ha_ac_off (
             void )
01015 {
01016     mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
01017     E.ac_sw_status = false;
01018 }
```

4.1.2.3 ha_ac_on()

```
void ha_ac_on (
             void )
01021 {
01022     mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
01023     E.ac_sw_status = true;
01024 }
```

4.1.2.4 ha_dc_off()

```
void ha_dc_off (
             void )
01030 {
01031     mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
01032     E.gti_sw_status = false;
01033 }
```

4.1.2.5 ha_dc_on()

```
void ha_dc_on (
            void )
```

```
01036 {
01037     mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
01038     E.gti_sw_status = true;
01039 }
```

4.1.2.6 log_time()

```
char * log_time (
            bool log)
```

```
01100 {
01101     static char time_log[RBUF_SIZ] = {0};
01102     static uint32_t len = 0, sync_time = TIME_SYNC_SEC - 1;
01103     time_t rawtime_log;
01104
01105     tzset();
01106     timezone = 0;
01107     daylight = 0;
01108     time(&rawtime_log);
01109     if (sync_time++ > TIME_SYNC_SEC) {
01110         sync_time = 0;
01111         snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
      time commands
01112         mqtt_gti_time(E.client_p, TOPIC_P, time_log);
01113     }
01114
01115     sprintf(time_log, "%s", ctime(&rawtime_log));
01116     len = strlen(time_log);
01117     time_log[len - 1] = 0; // munge out the return character
01118     if (log) {
01119         fprintf(fout, "%s ", time_log);
01120         fflush(fout);
01121     }
01122
01123     return time_log;
01124 }
```

4.1.2.7 log_timer()

```
bool log_timer (
            void )
```

```
01159 {
01160     bool itstime = false;
01161
01162     if (E.log_spam < LOW_LOG_SPAM) {
01163         E.log_time_reset = 0;
01164         itstime = true;
01165     }
01166     if (E.log_time_reset > RESET_LOG_SPAM) {
01167         E.log_spam = 0;
01168         itstime = true;
01169     }
01170     return itstime;
01171 }
```

4.1.2.8 main()

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

```
00356 {
00357     struct itimerval new_timer = {
00358         .it_value.tv_sec = CMD_SEC,
00359         .it_value.tv_usec = 0,
```

```
00360         .it_interval.tv_sec = CMD_SEC,
00361         .it_interval.tv_usec = 0,
00362     };
00363     struct itimerval old_timer;
00364     time_t rawtime;
00365     MQTTClient_connectOptions conn_opts_p = MQTTClient_connectOptions_initializer,
00366         conn_opts_sd = MQTTClient_connectOptions_initializer,
00367         conn_opts_ha = MQTTClient_connectOptions_initializer;
00368     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00369     MQTTClient_deliveryToken token;
00370     char hname[256], *hname_ptr = hname;
00371     size_t hname_len = 12;
00372 
00373     gethostname(hname, hname_len);
00374     hname[12] = 0;
00375     printf("\r\n  LOG Version %s : MQTT Version %s : Host Name %s\r\n", LOG_VERSION, MQTT_VERSION,
      hname);
00376     showIP();
00377     skeleton_daemon();
00378 
00379     while (true) {
00380         switch (E.mode.E) {
00381         case E_INIT:
00382 
00383 #ifdef LOG_TO_FILE
00384             fout = fopen(LOG_TO_FILE, "a");
00385             if (fout == NULL) {
00386                 fout = fopen(LOG_TO_FILE_ALT, "a");
00387                 if (fout == NULL) {
00388                     fout = stdout;
00389                     printf("\r\n%s Unable to open LOG file %s \r\n", log_time(false),
      LOG_TO_FILE_ALT);
00390                 }
00391             }
00392 #else
00393             fout = stdout;
00394 #endif
00395             fprintf(fout, "\r\n%s LOG Version %s : MQTT Version %s\r\n", log_time(false), LOG_VERSION,
      MQTT_VERSION);
00396             fflush(fout);
00397 
00398             if (!bsoc_init()) {
00399                 fprintf(fout, "\r\n%s bsoc_init failure \r\n", log_time(false));
00400                 fflush(fout);
00401                 exit(EXIT_FAILURE);
00402             }
00403             /*
00404              * set the timer for MQTT publishing sample speed
00405              * CMD_SEC         10
00406              */
00407             setitimer(ITIMER_REAL, &new_timer, &old_timer);
00408             signal(SIGALRM, timer_callback);
00409 
00410             if (strncmp(hname, TNAME, 6) == 0) {
00411                 MQTTClient_create(&E.client_p, LADDRESS, CLIENTID1,
00412                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00413                 conn_opts_p.keepAliveInterval = KAI;
00414                 conn_opts_p.cleansession = 1;
00415                 hname_ptr = LADDRESS;
00416             } else {
00417                 MQTTClient_create(&E.client_p, ADDRESS, CLIENTID1,
00418                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00419                 conn_opts_p.keepAliveInterval = KAI;
00420                 conn_opts_p.cleansession = 1;
00421                 hname_ptr = ADDRESS;
00422             }
00423 
00424             fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID1);
00425             fflush(fout);
00426             ha_flag_vars_ss.cid = ID_C1;
00427             MQTTClient_setCallbacks(E.client_p, &ha_flag_vars_ss, connlost, msgarrvd, delivered);
00428             if ((E.rc = MQTTClient_connect(E.client_p, &conn_opts_p)) != MQTTCLIENT_SUCCESS) {
00429                 fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
      log_time(false), E.rc, hname_ptr, CLIENTID1);
00430                 fflush(fout);
00431                 pthread_mutex_destroy(&E.ha_lock);
00432                 exit(EXIT_FAILURE);
00433             }
00434 
00435             if (strncmp(hname, TNAME, 6) == 0) {
00436                 MQTTClient_create(&E.client_sd, LADDRESS, CLIENTID2,
00437                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00438                 conn_opts_sd.keepAliveInterval = KAI;
00439                 conn_opts_sd.cleansession = 1;
00440                 hname_ptr = LADDRESS;
00441             } else {
00442                 MQTTClient_create(&E.client_sd, ADDRESS, CLIENTID2,
```

```
00443                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00444                 conn_opts_sd.keepAliveInterval = KAI;
00445                 conn_opts_sd.cleansession = 1;
00446                 hname_ptr = ADDRESS;
00447             }
00448 
00449             fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID2);
00450             fflush(fout);
00451             ha_flag_vars_sd.cid = ID_C2;
00452             MQTTClient_setCallbacks(E.client_sd, &ha_flag_vars_sd, connlost, msgarrvd, delivered);
00453             if ((E.rc = MQTTClient_connect(E.client_sd, &conn_opts_sd)) != MQTTCLIENT_SUCCESS) {
00454                 fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
      log_time(false), E.rc, hname_ptr, CLIENTID2);
00455                 fflush(fout);
00456                 pthread_mutex_destroy(&E.ha_lock);
00457                 exit(EXIT_FAILURE);
00458             }
00459 
00460             /*
00461              * Home Assistant MQTT receive messages
00462              */
00463             if (strncmp(hname, TNAME, 6) == 0) {
00464                 MQTTClient_create(&E.client_ha, LADDRESS, CLIENTID3,
00465                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00466                 conn_opts_ha.keepAliveInterval = KAI;
00467                 conn_opts_ha.cleansession = 1;
00468                 hname_ptr = LADDRESS;
00469             } else {
00470                 MQTTClient_create(&E.client_ha, ADDRESS, CLIENTID3,
00471                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00472                 conn_opts_ha.keepAliveInterval = KAI;
00473                 conn_opts_ha.cleansession = 1;
00474                 hname_ptr = ADDRESS;
00475             }
00476 
00477             fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID3);
00478             fflush(fout);
00479             ha_flag_vars_ha.cid = ID_C3;
00480             MQTTClient_setCallbacks(E.client_ha, &ha_flag_vars_ha, connlost, msgarrvd, delivered);
00481             if ((E.rc = MQTTClient_connect(E.client_ha, &conn_opts_ha)) != MQTTCLIENT_SUCCESS) {
00482                 fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
      log_time(false), E.rc, hname_ptr, CLIENTID3);
00483                 fflush(fout);
00484                 pthread_mutex_destroy(&E.ha_lock);
00485                 exit(EXIT_FAILURE);
00486             }
00487 
00488             /*
00489              * on topic received data will trigger the msgarrvd function
00490              */
00491             MQTTClient_subscribe(E.client_p, TOPIC_SS, QOS); // FM80 Q84
00492             MQTTClient_subscribe(E.client_sd, TOPIC_SD, QOS); // DUMPLOAD K42
00493             MQTTClient_subscribe(E.client_ha, TOPIC_HA, QOS); // Home Assistant Linux AMD64  and ARM64
00494 
00495             pubmsg.payload = "online";
00496             pubmsg.payloadlen = strlen("online");
00497             pubmsg.qos = QOS;
00498             pubmsg.retained = 0;
00499             ha_flag_vars_ss.deliveredtoken = 0;
00500             // notify HA we are running and controlling AC power plugs
00501             MQTTClient_publishMessage(E.client_p, TOPIC_PACA, &pubmsg, &token);
00502             MQTTClient_publishMessage(E.client_p, TOPIC_PDCA, &pubmsg, &token);
00503 
00504             // sync HA power switches
00505             mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
00506             mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
00507             mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00508             mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00509             mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
00510             mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
00511 
00512             E.ac_sw_on = true; // can be switched on once
00513             E.gti_sw_on = true; // can be switched on once
00514 
00515             /*
00516              * use libcurl to read AC power meter HTTP data
00517              * iammeter connected for split single phase monitoring and one leg GTI power exporting
00518              */
00519             iammeter_read1(IAMM1);
00520             iammeter_read2(IAMM2);
00521 
00522             /*
00523              * start the main energy monitoring loop
00524              */
00525             fprintf(fout, "\r\n%s Solar Energy AC power controller\r\n", log_time(false));
00526 
00527 #ifdef FAKE_VPV
```

```
00528             fprintf(fout, "\r\n Faking dumpload PV voltage\r\n");
00529 #endif
00530             ha_flag_vars_ss.energy_mode = NORM_MODE;
00531             E.mode.E = E_WAIT;
00532             break;
00533         case E_WAIT:
00534             if (ha_flag_vars_ss.runner || E.speed_go++ > 1500000) {
00535                 E.speed_go = 0;
00536                 ha_flag_vars_ss.runner = false;
00537                 E.mode.E = E_RUN;
00538             }
00539
00540             usleep(100);
00541             /*
00542              * main state-machine update sequence
00543              */
00544             bsoc_data_collect();
00545             if (!sanity_check()) {
00546                 fprintf(fout, "\r\n%s Sanity Check error %d %s \r\n", log_time(false), E.sane,
      mqtt_name[E.sane]);
00547                 fflush(fout);
00548             }
00549
00550             /*
00551              * stop and restart the energy control processing
00552              * from inside the program or from a remote Home Assistant command
00553              */
00554             if (solar_shutdown()) {
00555                 if (!E.startup) {
00556                     fprintf(fout, "%s SHUTDOWN Solar Energy Control ---> \r\n", log_time(false));
00557                 }
00558                 fflush(fout);
00559                 ramp_down_gti(E.client_p, true);
00560                 usleep(100000); // wait
00561                 ramp_down_ac(E.client_p, true);
00562                 usleep(100000); // wait
00563                 ramp_down_gti(E.client_p, true);
00564                 usleep(100000); // wait
00565                 ramp_down_ac(E.client_p, true);
00566                 usleep(100000); // wait
00567                 if (!E.startup) {
00568                     fprintf(fout, "%s Completed SHUTDOWN, Press again to RESTART.\r\n",
      log_time(false));
00569                     fflush(fout);
00570                 }
00571                 fflush(fout);
00572
00573                 uint8_t iam_delay = 0;
00574                 while (solar_shutdown()) {
00575                     mqtt_ha_shutdown(E.client_p, TOPIC_SHUTDOWN);
00576                     usleep(USEC_SEC); // wait
00577                     if ((int32_t) E.mvar[V_HACSW]) {
00578                         ha_ac_off();
00579                     }
00580                     if ((int32_t) E.mvar[V_HDCSW]) {
00581                         ha_dc_off();
00582                     }
00583                     if ((iam_delay++ > IAM_DELAY) && E.link.shutdown) {
00584                         E.fm80 = true;
00585                         E.dumpload = true;
00586                         E.iammeter = true;
00587                         E.homeassistant = true;
00588                     }
00589                 }
00590                 E.link.shutdown = 0;
00591                 fprintf(fout, "%s RESTART Solar Energy Control\r\n", log_time(false));
00592                 fflush(fout);
00593                 bsoc_set_mode(E.mode.pv_bias, true, true);
00594                 E.dl_excess = true;
00595                 mqtt_gti_power(E.client_p, TOPIC_P, DL_POWER_ZERO, 1); // zero power at startup
00596                 E.dl_excess = false;
00597 #ifdef AUTO_CHARGE
00598                 mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00599 #endif
00600                 usleep(100000); // wait
00601                 E.gti_sw_status = true;
00602                 ResetPI(&E.mode.pid);
00603                 ha_flag_vars_ss.runner = true;
00604                 E.fm80 = true;
00605                 E.dumpload = true;
00606                 E.iammeter = true;
00607                 E.homeassistant = true;
00608                 E.mode.in_pid_control = false; // shutdown auto energy control
00609                 E.mode.R = R_INIT;
00610             }
00611             if (ha_flag_vars_ss.receivedtoken) {
00612                 ha_flag_vars_ss.receivedtoken = false;
```

```
00613             }
00614             if (ha_flag_vars_sd.receivedtoken) {
00615                 ha_flag_vars_sd.receivedtoken = false;
00616             }
00617             break;
00618         case E_RUN:
00619             usleep(100);
00620             switch (E.mode.R) {
00621             case R_INIT:
00622                 E.once_ac = true;
00623                 E.once_gti = true;
00624                 E.ac_sw_on = true;
00625                 E.gti_sw_on = true;
00626                 E.mode.R = R_RUN;
00627                 E.mode.no_float = true;
00628                 break;
00629             case R_FLOAT:
00630                 if (E.mode.no_float) {
00631                     E.once_ac = true;
00632                     E.once_gti = true;
00633                     E.ac_sw_on = true;
00634                     E.gti_sw_on = true;
00635                     E.gti_sw_status = false;
00636                     E.ac_sw_status = false;
00637                     E.mode.no_float = false;
00638                 }
00639                 if (!E.gti_sw_status) {
00640                     if (gti_test() > MIN_BAT_KW_GTI_HI) {
00641                         mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00642                         E.gti_sw_status = true;
00643                         fprintf(fout, "%s R_FLOAT DC switch true \r\n", log_time(false));
00644                     }
00645                 }
00646                 usleep(100000); // wait
00647                 if (!E.ac_sw_status) {
00648                     if (ac_test() > MIN_BAT_KW_AC_HI) {
00649                         mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00650                         E.ac_sw_status = true;
00651                         fprintf(fout, "%s R_FLOAT AC switch true \r\n", log_time(false));
00652                     }
00653                 }
00654                 E.mode.pv_bias = PV_BIAS;
00655                 fm80_float(true);
00656                 break;
00657             case R_RUN:
00658             default:
00659                 E.mode.R = R_RUN;
00660                 E.mode.no_float = true;
00661                 break;
00662             }
00663             /*
00664              * main state-machine update sequence and control logic
00665              */
00666             /*
00667              * check for idle/data errors flags from sensors and HA
00668              */
00669             if (!E.mode.data_error) {
00670                 bsoc_set_mode(E.mode.pv_bias, true, false);
00671                 if (E.gti_delay++ >= GTI_DELAY) {
00672                     char gti_str[SBUF_SIZ];
00673                     int32_t error_drive;
00674 
00675                     /*
00676                      * reset the control mode from simple switched power to PID control
00677                      */
00678                     if (!E.mode.in_pid_control) {
00679                         mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00680                         E.gti_sw_status = true;
00681                         usleep(100000); // wait
00682                         mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00683                         E.ac_sw_status = true;
00684                         E.mode.pv_bias = PV_BIAS;
00685                         fprintf(fout, "%s in_pid_mode AC/DC switch true \r\n", log_time(false));
00686                         fm80_float(true);
00687                     } else {
00688                         if (!fm80_float(true)) {
00689                             E.mode.pv_bias = (int32_t) E.mode.error - PV_BIAS;
00690                         }
00691                     }
00692                     /*
00693                      * use PID style set-point error correction
00694                      */
00695                     E.mode.in_pid_control = true;
00696                     E.gti_delay = 0;
00697                     /*
00698                      * adjust power balance if battery charging energy is low
00699                      */
```

```
00700                     if (E.mvar[V_DPBAT] > PV_DL_BIAS_RATE) {
00701                         error_drive = (int32_t) E.mode.error - E.mode.pv_bias; // PI feedback control
      signal
00702                     } else {
00703                         error_drive = (int32_t) E.mode.error - PV_BIAS_RATE;
00704                     }
00705                     /*
00706                      * when main battery is in float, crank-up the power draw from the solar panels
00707                      */
00708                     if (fm80_float(true)) {
00709                         error_drive = (int32_t) (E.mode.error + PV_BIAS);
00710                     }
00711                     /*
00712                      * don't drive to zero power
00713                      */
00714                     if (error_drive < 0) {
00715                         error_drive = PV_BIAS_LOW; // control wide power swings
00716                         if (!fm80_sleep()) { // check for using sleep bias
00717                             if ((E.mvar[V_FBEKW] > MIN_BAT_KW_BSOC_SLP) && (E.mvar[V_PWA] >
      PWA_SLEEP)) {
00718                                 error_drive = PV_BIAS_SLEEP; // use higher power when we still have
      sun for better inverter efficiency
00719                             }
00720                         }
00721                     }
00722
00723                     /*
00724                      * reduce charging/diversion power to safe PS limits
00725                      */
00726                     if (E.mode.dl_mqtt_max > PV_DL_MPTT_MAX) {
00727                         if (!E.dl_excess) {
00728                             error_drive = PV_DL_MPTT_IDLE;
00729                         } else {
00730                             if (E.mode.dl_mqtt_max > PV_DL_MPTT_EXCESS) {
00731                                 error_drive = PV_DL_MPTT_IDLE;
00732                             }
00733                         }
00734                     } else {
00735                         if (E.dl_excess) {
00736                             error_drive = PV_DL_EXCESS + E.dl_excess_adj;
00737                         }
00738                     }
00739
00740                     /*
00741                      * shutdown GTI power at low DL battery Ah or Voltage
00742                      */
00743                     if ((E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) || (E.mvar[V_DVBAT] < PV_DL_B_V_LOW) ||
      (get_bat_runtime() < BAT_RUNTIME_GTI)) {
00744                         error_drive = PV_BIAS_ZERO;
00745                     }
00746
00747                     if (get_bat_runtime() < BAT_RUNTIME_LOW) {
00748                         error_drive = PV_BIAS_ZERO;
00749                         ha_ac_off();
00750                         ha_ac_off();
00751                         ha_ac_off();
00752                         ha_ac_off();
00753                         ha_dc_off();
00754                         ha_dc_off();
00755                         ha_dc_off();
00756                         ha_dc_off();
00757                         fprintf(fout, "%s Main Battery Runtime too low, shutting down power drains,
      %6f Hrs\r\n", log_time(false), get_bat_runtime());
00758                         ramp_down_ac(E.client_p, true);
00759                         ramp_down_gti(E.client_p, true);
00760                         mqtt_ha_shutdown(E.client_p, TOPIC_SHUTDOWN);
00761                         fflush(fout);
00762                     }
00763                     snprintf(gti_str, SBUF_SIZ - 1, "V%04dX", error_drive); // format for dumpload
      controller gti power commands
00764                     mqtt_gti_power(E.client_p, TOPIC_P, gti_str, 2);
00765
00766                 }
00767
00768 #ifndef  FAKE_VPV
00769                 if (fm80_float(true) || ((ac1_filter(E.mvar[V_BEN]) > BAL_MAX_ENERGY_AC) && (ac_test()
      > MIN_BAT_KW_AC_HI))) {
00770                     ramp_up_ac(E.client_p, E.ac_sw_on); // use once control
00771 #ifdef PSW_DEBUG
00772                     fprintf(fout, "%s MIN_BAT_KW_AC_HI AC switch %d \r\n", log_time(false),
      E.ac_sw_on);
00773 #endif
00774                     E.ac_sw_on = false; // once flag
00775                 }
00776 #endif
00777                 if (((ac2_filter(E.mvar[V_BEN]) < BAL_MIN_ENERGY_AC) || ((ac_test() <
      (MIN_BAT_KW_AC_LO + E.ac_low_adj))))) {
```

```
00778                     if (!fm80_float(true)) {
00779                         ramp_down_ac(E.client_p, E.ac_sw_on);
00780                         if (log_timer()) {
00781                             fprintf(fout, "%s RAMP DOWN AC, MIN_BAT_KW_AC_LO AC switch %d \r\n",
    log_time(false), E.ac_sw_on);
00782                         }
00783                     }
00784                     E.ac_sw_on = true;
00785                 }
00786 
00787 
00788                 /*
00789                  * Dump Load Excess testing
00790                  * send excess power into the home power grid taking care not to export energy to the
    utility grid
00791                  */
00792                 if (((dc1_filter(E.mvar[V_BEN]) > BAL_MAX_ENERGY_GTI) && (gti_test() >
    MIN_BAT_KW_GTI_HI)) || E.dl_excess) {
00793 #ifndef  FAKE_VPV
00794 #ifdef B_DLE_DEBUG
00795                     if (E.dl_excess) {
00796                         fprintf(fout, "%s DL excess ramp_up_gti, DC switch %d\r\n", log_time(false),
    E.gti_sw_on);
00797                     }
00798 #endif
00799                     ramp_up_gti(E.client_p, E.gti_sw_on, E.dl_excess);
00800                     if (log_timer()) {
00801                         fprintf(fout, "%s RAMP DOWN DC, MIN_BAT_KW_GTI_HI DC switch %d \r\n",
    log_time(false), E.gti_sw_on);
00802                     }
00803                     E.gti_sw_on = false; // once flag
00804 #endif
00805                 } else {
00806                     if ((dc2_filter(E.mvar[V_BEN]) < BAL_MIN_ENERGY_GTI) || (gti_test() <
    (MIN_BAT_KW_GTI_LO + E.gti_low_adj))) {
00807                         if (!E.dl_excess) {
00808                             if (log_timer()) {
00809                                 ramp_down_gti(E.client_p, true);
00810 #ifdef PSW_DEBUG
00811                                 fprintf(fout, "%s MIN_BAT_KW_GTI_LO DC switch %d \r\n",
    log_time(false), E.gti_sw_on);
00812 #endif
00813                             }
00814                             E.gti_sw_on = true;
00815                         }
00816                     }
00817                 }
00818             };
00819 
00820 #ifdef B_ADJ_DEBUG
00821             fprintf(fout, "\r\n LO ADJ: AC %8.2fWh, GTI %8.2fWh\r\n", MIN_BAT_KW_AC_LO + E.ac_low_adj,
    MIN_BAT_KW_GTI_LO + E.gti_low_adj);
00822 #endif
00823 #ifdef B_DLE_DEBUG
00824             if (E.dl_excess) {
00825                 fprintf(fout, "%s DL excess vars from ha_energy %d %d : Flag %d\r\n", log_time(false),
    E.mode.con4, E.mode.con5, E.dl_excess);
00826             }
00827 #endif
00828 
00829             time(&rawtime);
00830 
00831             if (E.im_delay++ >= IM_DELAY) {
00832                 E.im_delay = 0;
00833                 iammeter_read1(IAMM1);
00834                 iammeter_read2(IAMM2);
00835             }
00836             if (E.im_display++ >= IM_DISPLAY) {
00837                 char buffer[SYSLOG_SIZ];
00838                 uint32_t len;
00839 
00840                 E.im_display = 0;
00841                 mqtt_ha_pid(E.client_p, TOPIC_PPID);
00842                 if (!(E.fm80 && E.dumpload && E.iammeter)) {
00843                     if (!E.iammeter) {
00844                         E.link.iammeter_error++;
00845                     } else {
00846                         E.link.mqtt_error++;
00847                     }
00848                     E.link.shutdown++;
00849                     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,
00850                         E.link.mqtt_count, E.link.mqtt_error, E.link.iammeter_count,
    E.link.iammeter_error);
00851                     fflush(fout);
00852                     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,
00853                         E.link.mqtt_count, E.link.mqtt_error, E.link.iammeter_count,
      E.link.iammeter_error);
00854                     syslog(LOG_NOTICE, buffer);
00855                     mqtt_ha_shutdown(E.client_p, TOPIC_SHUTDOWN);
00856                     E.mode.data_error = true;
00857                 } else {
00858                     E.mode.data_error = false;
00859                     E.link.shutdown = 0;
00860                 }
00861                 snprintf(buffer, RBUF_SIZ - 1, "%s", ctime(&rawtime));
00862                 len = strlen(buffer);
00863                 buffer[len - 1] = 0; // munge out the return character
00864                 fprintf(fout, "%s ", buffer);
00865                 fflush(fout);
00866                 E.fm80 = false;
00867                 E.dumpload = false;
00868                 E.homeassistant = false;
00869                 E.iammeter = false;
00870                 sync_ha();
00871                 print_im_vars();
00872                 print_mvar_vars();
00873                 fprintf(fout, "%s\r", ctime(&rawtime));
00874             }
00875             E.mode.E = E_WAIT;
00876             fflush(fout);
00877             if (E.mode.con6) {
00878                 E.mode.R = R_IDLE;
00879             }
00880             if (E.mode.con7) {
00881                 E.mode.E = E_STOP;
00882             }
00883             break;
00884         case E_STOP:
00885         default:
00886             fflush(fout);
00887             fprintf(fout, "\r\n%s HA Energy stopped and exited.\r\n", log_time(false));
00888             fflush(fout);
00889             return 0;
00890             break;
00891         }
00892     }
00893 }
```

4.1.2.9 ramp_down_ac()

```
void ramp_down_ac (
             MQTTClient client_p,
             bool sw_off)
```

```
01005 {
01006     if (sw_off) {
01007         mqtt_ha_switch(client_p, TOPIC_PACC, false);
01008         E.ac_sw_status = false;
01009         usleep(200000);
01010     }
01011     E.once_ac = true;
01012 }
```

4.1.2.10 ramp_down_gti()

```
void ramp_down_gti (
             MQTTClient client_p,
             bool sw_off)
```

```
00966 {
00967     static uint32_t times = 0;
00968     if (sw_off) {
00969         mqtt_ha_switch(client_p, TOPIC_PDCC, false);
00970         E.once_gti_zero = true;
00971         times = 0;
00972         E.gti_sw_status = false;
00973     }
00974     E.once_gti = true;
00975
00976     if (E.once_gti_zero) {
00977         mqtt_gti_power(client_p, TOPIC_P, DL_POWER_ZERO, 7); // zero power
```

```
00978         if (times++ < LOW_LOG_SPAM) {
00979             E.once_gti_zero = false;
00980         }
00981     }
00982 }
```

4.1.2.11 ramp_up_ac()

```
void ramp_up_ac (
             MQTTClient client_p,
             bool start)
```

```
00988 {
00989
00990     if (start) {
00991         E.once_ac = true;
00992     }
00993
00994     if (E.once_ac) {
00995         if (get_bat_runtime() > BAT_RUNTIME_GTI) {
00996             E.once_ac = false;
00997             mqtt_ha_switch(client_p, TOPIC_PACC, true);
00998             E.ac_sw_status = true;
00999             usleep(200000); // wait for voltage to ramp
01000         }
01001     }
01002 }
```

4.1.2.12 ramp_up_gti()

```
void ramp_up_gti (
             MQTTClient client_p,
             bool start,
             bool excess)
```

```
00899 {
00900     static uint32_t sequence = 0;
00901
00902     if (start) {
00903         E.once_gti = true;
00904     }
00905
00906     if (E.once_gti) {
00907         E.once_gti = false;
00908         sequence = 0;
00909         if (!excess) {
00910             if (get_bat_runtime() > BAT_RUNTIME_GTI) {
00911                 mqtt_ha_switch(client_p, TOPIC_PDCC, true);
00912                 E.gti_sw_status = true;
00913                 usleep(500000); // wait for PS voltage to ramp
00914             }
00915         } else {
00916             sequence = 1;
00917         }
00918     }
00919
00920     switch (sequence) {
00921     case 4:
00922         E.once_gti_zero = true;
00923         break;
00924     case 3:
00925     case 2:
00926     case 1:
00927         E.once_gti_zero = true;
00928         if (bat_current_stable() || E.dl_excess) { // check battery current std dev, stop
      'motorboating'
00929             sequence++;
00930             if (get_bat_runtime() > BAT_RUNTIME_GTI) {
00931                 if (!mqtt_gti_power(client_p, TOPIC_P, "+#", 3)) {
00932                     sequence = 0;
00933                 }; // +100W power
00934             } else {
00935                 sequence = 0;
00936             }
00937         } else {
00938             usleep(500000); // wait a bit more for power to be stable
```

```
00939             sequence = 1; // do power ramps when ready
00940             if (!mqtt_gti_power(client_p, TOPIC_P, "-#", 4)) {
00941                 sequence = 0;
00942             }; // - 100W power
00943         }
00944         break;
00945     case 0:
00946         sequence++;
00947         if (E.once_gti_zero) {
00948             mqtt_gti_power(client_p, TOPIC_P, DL_POWER_ZERO, 5); // zero power
00949             E.once_gti_zero = false;
00950         }
00951         break;
00952     default:
00953         if (E.once_gti_zero) {
00954             mqtt_gti_power(client_p, TOPIC_P, DL_POWER_ZERO, 6); // zero power
00955             E.once_gti_zero = false;
00956         }
00957         sequence = 0;
00958         break;
00959     }
00960 }
```

4.1.2.13 sanity_check()

```
bool sanity_check (
             void )
```

```
00258 {
00259     if (E.mvar[V_PWA] > PWA_SANE) {
00260         E.sane = S_PWA;
00261         return false;
00262     }
00263     if (E.mvar[V_PAMPS] > PAMPS_SANE) {
00264         E.sane = S_PAMPS;
00265         return false;
00266     }
00267     if (E.mvar[V_PVOLTS] > PVOLTS_SANE) {
00268         E.sane = S_PVOLTS;
00269         return false;
00270     }
00271     if (E.mvar[V_FBAMPS] > BAMPS_SANE) {
00272         E.sane = S_FBAMPS;
00273         return false;
00274     }
00275     return true;
00276 }
```

4.1.2.14 showIP()

```
void showIP (
             void )
```

```
00164 {
00165     struct ifaddrs *ifaddr, *ifa;
00166     int s;
00167     char host[NI_MAXHOST];
00168
00169     if (getifaddrs(&ifaddr) == -1) {
00170         perror("getifaddrs");
00171         exit(EXIT_FAILURE);
00172     }
00173
00174
00175     for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
00176         if (ifa->ifa_addr == NULL)
00177             continue;
00178
00179         s = getnameinfo(ifa->ifa_addr, sizeof(struct sockaddr_in), host, NI_MAXHOST, NULL, 0,
      NI_NUMERICHOST);
00180
00181         if (ifa->ifa_addr->sa_family == AF_INET) {
00182             if (s != 0) {
00183                 exit(EXIT_FAILURE);
00184             }
00185             printf("\tInterface : <%s>\n", ifa->ifa_name);
00186             printf("\t  Address : <%s>\n", host);
00187         }
00188     }
00189
00190     freeifaddrs(ifaddr);
00191 }
```

4.1.2.15 skeleton_daemon()

```
void skeleton_daemon ()  [static]
```

```
00198 {
00199     pid_t pid;
00200
00201     /* Fork off the parent process */
00202     pid = fork();
00203
00204     /* An error occurred */
00205     if (pid < 0) {
00206         printf("\r\n%s DAEMON failure  LOG Version %s : MQTT Version %s\r\n", log_time(false),
      LOG_VERSION, MQTT_VERSION);
00207         exit(EXIT_FAILURE);
00208     }
00209
00210     /* Success: Let the parent terminate */
00211     if (pid > 0) {
00212         exit(EXIT_SUCCESS);
00213     }
00214
00215     /* On success: The child process becomes session leader */
00216     if (setsid() < 0) {
00217         exit(EXIT_FAILURE);
00218     }
00219
00220     /* Catch, ignore and handle signals */
00221     /*TODO: Implement a working signal handler */
00222     //    signal(SIGCHLD, SIG_IGN);
00223     //    signal(SIGHUP, SIG_IGN);
00224
00225     /* Fork off for the second time*/
00226     pid = fork();
00227
00228     /* An error occurred */
00229     if (pid < 0) {
00230         exit(EXIT_FAILURE);
00231     }
00232
00233     /* Success: Let the parent terminate */
00234     if (pid > 0) {
00235         exit(EXIT_SUCCESS);
00236     }
00237
00238     /* Set new file permissions */
00239     umask(0);
00240
00241     /* Change the working directory to the root directory */
00242     /* or another appropriated directory */
00243     chdir("/");
00244
00245     /* Close all open file descriptors */
00246     int x;
00247     for (x = sysconf(_SC_OPEN_MAX); x >= 0; x--) {
00248         close(x);
00249     }
00250
00251 }
```

4.1.2.16 solar_shutdown()

```
bool solar_shutdown (
            void )  [static]
```

```
01045 {
01046     static bool ret = false;
01047
01048     if (E.startup) {
01049         ret = true;
01050         E.startup = false;
01051         return ret;
01052     } else {
01053         ret = false;
01054
01055         /*
01056          * FIXME
01057          *
01058          */
01059     }
01060
```

```
01061     if (E.solar_shutdown) {
01062         ret = true;
01063     } else {
01064         ret = false;
01065     }
01066
01067     if ((E.mvar[V_FBEKW] < BAT_CRITICAL) && !E.startup) { // special case for low battery
01068         if (!E.mode.bat_crit) {
01069             ret = true;
01070 #ifdef CRITIAL_SHUTDOWN_LOG
01071             fprintf(fout, "%s Solar BATTERY CRITICAL shutdown comms check \r\n", log_time(false));
01072             fflush(fout);
01073 #endif
01074             E.mode.bat_crit = true;
01075             return ret;
01076         }
01077     } else {
01078         E.mode.bat_crit = false;
01079     }
01080
01081     if (E.link.shutdown >= MAX_ERROR) {
01082         ret = true;
01083         if (E.fm80 && E.dumpload && E.iammeter) {
01084             ret = false;
01085             E.link.shutdown = 0;
01086         }
01087
01088 #ifdef DEBUG_SHUTDOWN
01089         fprintf(fout, "%s Solar shutdown comms check ret = %d \r\n", log_time(false), ret);
01090         fflush(fout);
01091 #endif
01092     }
01093     return ret;
01094 }
```

4.1.2.17 sync_ha()

```
bool sync_ha (
            void )
```

```
01130 {
01131     bool sync = false;
01132     if (E.gti_sw_status != (bool) ((int32_t) E.mvar[V_HDCSW])) {
01133         fprintf(fout, "DC_MM %d %d ", (bool) E.gti_sw_status, (bool) ((int32_t) E.mvar[V_HDCSW]));
01134         mqtt_ha_switch(E.client_p, TOPIC_PDCC, !E.gti_sw_status);
01135         E.dc_mismatch = true;
01136         fflush(fout);
01137         sync = true;
01138     } else {
01139         E.dc_mismatch = false;
01140     }
01141
01142     E.ac_sw_status = (bool) ((int32_t) E.mvar[V_HACSW]); // TEMP FIX for MISmatch errors
01143     if (E.ac_sw_status != (bool) ((int32_t) E.mvar[V_HACSW])) {
01144         fprintf(fout, "AC_MM %d %d ", (bool) E.ac_sw_status, (bool) ((int32_t) E.mvar[V_HACSW]));
01145         mqtt_ha_switch(E.client_p, TOPIC_PACC, !E.ac_sw_status);
01146         E.ac_mismatch = true;
01147         fflush(fout);
01148         sync = true;
01149     } else {
01150         E.ac_mismatch = false;
01151     }
01152     return sync;
01153 }
```

4.1.2.18 timer_callback()

```
void timer_callback (
            int32_t signum)
```

```
00287 {
00288     signal(signum, timer_callback);
00289     ha_flag_vars_ss.runner = true;
00290     E.ten_sec_clock++;
00291     E.log_spam++;
00292     E.log_time_reset++;
00293     if (E.log_spam > MAX_LOG_SPAM) {
00294         E.log_spam = 0;
00295     }
00296 }
```

4.1.3 Variable Documentation

4.1.3.1 E

```
struct energy_type E
00111                           {
00112     .once_gti = true,
00113     .once_ac = true,
00114     .once_gti_zero = true,
00115     .iammeter = false,
00116     .fm80 = false,
00117     .dumpload = false,
00118     .homeassistant = false,
00119     .ac_low_adj = 0.0f,
00120     .gti_low_adj = 0.0f,
00121     .ac_sw_on = true,
00122     .gti_sw_on = true,
00123     .im_delay = 0,
00124     .gti_delay = 0,
00125     .im_display = 0,
00126     .rc = 0,
00127     .speed_go = 0,
00128     .mode.pid.iMax = PV_IMAX,
00129     .mode.pid.iMin = 0.0f,
00130     .mode.pid.pGain = PV_PGAIN,
00131     .mode.pid.iGain = PV_IGAIN,
00132     .mode.mode_tmr = 0,
00133     .mode.mode = true,
00134     .mode.in_pid_control = false,
00135     .mode.dl_mqtt_max = PV_DL_MPTT_MAX,
00136     .mode.E = E_INIT,
00137     .mode.R = R_INIT,
00138     .mode.no_float = true,
00139     .mode.data_error = false,
00140     .ac_sw_status = false,
00141     .gti_sw_status = false,
00142     .solar_mode = false,
00143     .solar_shutdown = false,
00144     .mode.pv_bias = PV_BIAS_LOW,
00145     .sane = S_DLAST,
00146     .startup = true,
00147     .ac_mismatch = false,
00148     .dc_mismatch = false,
00149     .mode_mismatch = false,
00150     .link.shutdown = 0,
00151     .mode.bat_crit = false,
00152     .dl_excess = false,
00153     .dl_excess_adj = 0.0f,
00154 };
```

4.1.3.2 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,
}
00096                                       {
00097     .runner = false,
00098     .receivedtoken = false,
00099     .deliveredtoken = false,
00100     .rec_ok = false,
00101     .ha_id = HA_ID,
00102     .var_update = 0,
00103 };
```

4.1.3.3 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,
}
```

```
00065                                       {
00066     .runner = false,
00067     .receivedtoken = false,
00068     .deliveredtoken = false,
00069     .rec_ok = false,
00070     .ha_id = P8055_ID,
00071     .var_update = 0,
00072 };
```

4.1.3.4 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,
}
```

```
00086                                       {
00087     .runner = false,
00088     .receivedtoken = false,
00089     .deliveredtoken = false,
00090     .rec_ok = false,
00091     .ha_id = DUMPLOAD_ID,
00092     .var_update = 0,
00093 };
```

4.1.3.5 ha_flag_vars_ss

```
struct ha_flag_type ha_flag_vars_ss
```

Initial value:

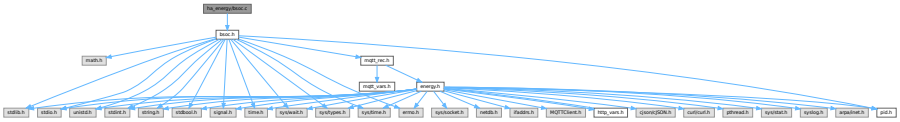
```
= {
    .runner = false,
    .receivedtoken = false,
    .deliveredtoken = false,
    .rec_ok = false,
    .ha_id = FM80_ID,
    .var_update = 0,
    .energy_mode = NORM_MODE,
}
```

```
00075                                       {
00076     .runner = false,
00077     .receivedtoken = false,
00078     .deliveredtoken = false,
00079     .rec_ok = false,
00080     .ha_id = FM80_ID,
00081     .var_update = 0,
00082     .energy_mode = NORM_MODE,
00083 };
```

4.2 ha_energy/bsoc.c File Reference

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

Include dependency graph for bsoc.c:



Data Structures

- struct local_type

Functions

- static double error_filter (const double)
- bool bsoc_init (void)
- void bsoc_set_std_dev (const double value, const uint32_t i)
- bool bsoc_data_collect (void)
- double bsoc_ac (void)
- double bsoc_gti (void)
- double gti_test (void)
- double ac_test (void)
- double get_bat_runtime (void)
- double get_batc_dev (void)
- double calculateStandardDeviation (const uint32_t N, const double data[ ])
- bool bat_current_stable (void)
- bool bsoc_set_mode (const double target, const bool mode, const bool init)
- double ac0_filter (const double raw)
- double ac1_filter (const double raw)
- double ac2_filter (const double raw)
- double dc0_filter (const double raw)
- double dc1_filter (const double raw)
- double dc2_filter (const double raw)
- double drive0_filter (const double raw)
- double drive1_filter (const double raw)

Variables

- const char ∗ mqtt_name [V_DLAST]
- static struct local_type L

4.2.1 Detailed Description

setup basic start state for BSOC functions

4.2.2 Function Documentation

4.2.2.1 ac0_filter()

```
double ac0_filter (
            const double raw)
```

```
00400 {
00401     static double accum = 0.0f;
00402     static double coef = COEFF;
00403     accum = accum - accum / coef + raw;
00404     return accum / coef;
00405 }
```

4.2.2.2 ac1_filter()

```
double ac1_filter (
            const double raw)
```

```
00408 {
00409     static double accum = 0.0f;
00410     static double coef = COEF;
00411     accum = accum - accum / coef + raw;
00412     return accum / coef;
00413 }
```

4.2.2.3 ac2_filter()

```
double ac2_filter (
            const double raw)
```

```
00416 {
00417     static double accum = 0.0f;
00418     static double coef = COEF;
00419     accum = accum - accum / coef + raw;
00420     return accum / coef;
00421 }
```

4.2.2.4 ac_test()

```
double ac_test (
            void )
```

```
00206 {
00207     return ac0_filter(L.ac_weight);
00208 }
```

4.2.2.5 bat_current_stable()

```
bool bat_current_stable (
            void )
```

```
00260 {
00261     static double gap = 0.0f;
00262
00263     if (L.batc_std_dev <= (MAX_BATC_DEV + gap)) {
00264         gap = MAX_BATC_DEV;
00265         if (L.bat_c_std_dev[0] < BAT_C_DRAW) {
00266             return true;
00267         } else {
00268             gap = 0.0f;
00269             return false;
00270         }
00271     } else {
00272         gap = 0.0f;
00273         return false;
00274     }
00275 }
```

4.2.2.6 bsoc_ac()

```
double bsoc_ac (
            void )
```

```
00139 {
00140
00141     return ac0_filter(L.ac_weight);
00142 };
```

4.2.2.7 bsoc_data_collect()

```
bool bsoc_data_collect (
            void )
```

```
00087 {
00088     bool ret = false;
00089     static uint32_t i = 0;
00090     // lockout threaded updates
00091     pthread_mutex_lock(&E.ha_lock); // lockout MQTT var updates
00092
00093     L.ac_weight = E.mvar[V_FBEKW];
00094     L.gti_weight = E.mvar[V_FBEKW];
00095 #ifdef FAKE_VPV // no DUMPLOAD AC charger
00096     if (E.gti_sw_on) {
00097         pv_voltage = PV_V_NOM;
00098     } else {
00099         pv_voltage = PV_V_FAKE;
00100     }
00101     E.mvar[V_DVPV] = pv_voltage;
00102 #else
00103     L.pv_voltage = E.mvar[V_DVPV];
00104 #endif
00105     L.bat_voltage = E.mvar[V_DVBAT];
00106     L.bat_current = E.mvar[V_DCMPPT];
00107     L.bat_runtime = E.mvar[V_FRUNT];
00108     E.ac_low_adj = E.mvar[V_FSO]* -0.5f;
00109     E.gti_low_adj = E.mvar[V_FACE] * -0.5f;
00110     E.mode.dl_mqtt_max = E.mvar[V_DPMPPT];
00111     E.bat_runtime = E.mvar[V_FRUNT];
00112
00113     pthread_mutex_unlock(&E.ha_lock); // resume remote MQTT var updates
00114
00115     if (E.ac_low_adj < -2000.0f) {
00116         E.ac_low_adj = -2000.0f;
00117     }
00118     if (E.gti_low_adj < -2000.0f) {
00119         E.gti_low_adj = -2000.0f;
00120     }
00121
00122     L.bat_c_std_dev[i++] = L.bat_current;
00123     if (i >= DEV_SIZE) {
00124         i = 0;
00125     }
00126
00127     calculateStandardDeviation(DEV_SIZE, L.bat_c_std_dev);
00128
00129 #ifdef BSOC_DEBUG
00130     fprintf(fout, "\r\nmqtt var bsoc update\r\n");
00131 #endif
00132     return ret;
00133 }
```

4.2.2.8 bsoc_gti()

```
double bsoc_gti (
            void )
```

```
00149 {
00150     static bool once = true;
00151 #ifdef BSOC_DEBUG
00152     fprintf(fout, "pvp %f, gweight %f, aweight %f, batv %f, batc %f\r\n", pv_voltage, gti_weight,
      ac_weight, bat_voltage, bat_current);
00153 #endif
00154     // check for 48VDC AC charger powered from the Solar battery bank AC inverter unless E.dl_excess
      is TRUE
```

```
00155     if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
00156         L.gti_weight = 0.0f; // reduce power to zero
00157     } else {
00158         if (E.dl_excess) {
00159             if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
00160                 L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
00161                 once = true;
00162             } else {
00163                 L.gti_weight = 0.0f; // reduce power to zero
00164                 if (once) {
00165                     fprintf(fout, "%s Dump Load Battery Ah below %f \n", log_time(false),
      PV_DL_B_AH_MIN);
00166                     once = false;
00167                 }
00168             }
00169         }
00170     }
00171
00172
00173     return dc0_filter(L.gti_weight);
00174 };
```

4.2.2.9 bsoc_init()

```
bool bsoc_init (
              void )
```

```
00062 {
00063     L.ac_weight = 0.0f;
00064     L.gti_weight = 0.0f;
00065     // use MUTEX locks for message passing between remote programs
00066     if (pthread_mutex_init(&E.ha_lock, NULL) != 0) {
00067         fprintf(fout, "%s mutex init has failed\n", log_time(false));
00068         return false;
00069     }
00070     return true;
00071 };
```

4.2.2.10 bsoc_set_mode()

```
bool bsoc_set_mode (
              const double target,
              const bool mode,
              const bool init)
```

```
00282 {
00283     static bool bsoc_mode = false;
00284     static bool bsoc_high = false, ha_ac_mode = true;
00285     static double accum = 0.0f, vpwa = 0.0f;
00286
00287     if (init) {
00288         bsoc_mode = false;
00289         bsoc_high = false;
00290         ha_ac_mode = true;
00291         accum = 0.0f;
00292         vpwa = 0.0f;
00293         return true;
00294     }
00295     /*
00296      * running avg filter
00297      */
00298     accum = accum - accum / COEFN + E.mvar[V_PWA];
00299     vpwa = accum / COEFN;
00300
00301     if ((vpwa >= PV_FULL_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
00302         if (!bsoc_mode) {
00303             ResetPI(&E.mode.pid);
00304         }
00305         bsoc_mode = true;
00306         bsoc_high = true;
00307         if (!ha_ac_mode) {
00308             ha_ac_on();
00309             ha_ac_mode = true;
00310         }
00311
00312     } else {
00313         if (bsoc_high) { // turn off at min limit power
```

```
00314             if ((vpwa >= PV_MIN_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
00315                 bsoc_mode = true;
00316                 if (ha_ac_mode) {
00317                     ha_ac_off();
00318                     ha_ac_mode = false;
00319                 }
00320             } else {
00321                 bsoc_high = false;
00322                 ha_ac_mode = false;
00323             }
00324         }
00325     }
00326
00327     E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) + E.mvar[V_DPPV]; // use as a temp variable
00328     E.mode.total_system = (E.mvar[V_FLO] - E.mode.gti_dumpload) + E.mvar[V_DPPV] +(E.print_vars[L3_P]*
      -1.0f);
00329     E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) - E.mvar[V_DPPV]; // use this value
00330
00331     /*
00332      * look at system energy balance for power control drive
00333      */
00334     if (mode) { // add GTI power from dumpload
00335         E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + E.mode.gti_dumpload +
      PBAL_OFFSET);
00336     } else {
00337         E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + PBAL_OFFSET);
00338     }
00339
00340     if (E.mode.error > 0.0f) {
00341         L.coef = COEF;
00342     } else {
00343         L.coef = COEFN;
00344     }
00345     E.mode.target = target;
00346     E.mode.error = round(error_filter(E.mode.error));
00347     /*
00348      * check for idle flag from HA
00349      */
00350     if (E.mode.con6) {
00351         ha_ac_mode = true;
00352         bsoc_mode = false;
00353         fprintf(fout, "%s idle flag from HA\n", log_time(false));
00354     }
00355
00356     /*
00357      * HA start excess button pressed
00358      */
00359     if (E.mode.con4) {
00360         E.dl_excess = true;
00361         E.mode.con4 = false;
00362         fprintf(fout, "%s HA start excess button pressed\n", log_time(false));
00363     }
00364
00365     /*
00366      * HA stop excess button pressed
00367      */
00368     if (E.mode.con5) {
00369         mqtt_gti_power(E.client_p, TOPIC_P, DL_POWER_ZERO, 9); // zero power at excess shutdown
00370         E.dl_excess = false;
00371         E.mode.con5 = false;
00372         fprintf(fout, "%s HA stop excess button pressed\n", log_time(false));
00373     }
00374
00375     /*
00376      * DL buffer battery low set-point excess load shutdown
00377      */
00378     if (E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) {
00379         mqtt_gti_power(E.client_p, TOPIC_P, DL_POWER_ZERO, 10); // zero power at excess shutdown
00380         E.dl_excess = false;
00381         E.mode.con4 = false;
00382         E.mode.con5 = false;
00383     }
00384
00385     fflush(fout);
00386     return bsoc_mode;
00387 }
```

4.2.2.11 bsoc_set_std_dev()

```
void bsoc_set_std_dev (
            const double value,
            const uint32_t i)
```

```
00077 {
00078     L.bat_c_std_dev[i] = value;
00079 }
```

4.2.2.12 calculateStandardDeviation()

```
double calculateStandardDeviation (
              const uint32_t N,
              const double data[])
```

```
00225 {
00226     // variable to store sum of the given data
00227     double sum = 0;
00228
00229     for (int i = 0; i < N; i++) {
00230         sum += data[i];
00231     }
00232
00233     // calculating mean
00234     double mean = sum / N;
00235
00236     // temporary variable to store the summation of square
00237     // of difference between individual data items and mean
00238     double values = 0;
00239
00240     for (int i = 0; i < N; i++) {
00241         values += pow(data[i] - mean, 2);
00242     }
00243
00244     // variance is the square of standard deviation
00245     double variance = values / N;
00246
00247     // calculating standard deviation by finding square root
00248     // of variance
00249     double standardDeviation = sqrt(variance);
00250     L.batc_std_dev = standardDeviation;
00251
00252 #ifdef BSOC_DEBUG
00253     // printing standard deviation
00254     fprintf(fout, "STD DEV of Current %.2f\r\n", standardDeviation);
00255 #endif
00256     return standardDeviation;
00257 }
```

4.2.2.13 dc0_filter()

```
double dc0_filter (
              const double raw)
```

```
00424 {
00425     static double accum = 0.0f;
00426     static double coef = COEFF;
00427     accum = accum - accum / coef + raw;
00428     return accum / coef;
00429 }
```

4.2.2.14 dc1_filter()

```
double dc1_filter (
              const double raw)
```

```
00432 {
00433     static double accum = 0.0f;
00434     static double coef = COEF;
00435     accum = accum - accum / coef + raw;
00436     return accum / coef;
00437 }
```

4.2.2.15 dc2_filter()

```
double dc2_filter (
            const double raw)
```

```
00440 {
00441     static double accum = 0.0f;
00442     static double coef = COEF;
00443     accum = accum - accum / coef + raw;
00444     return accum / coef;
00445 }
```

4.2.2.16 drive0_filter()

```
double drive0_filter (
            const double raw)
```

```
00448 {
00449     static double accum = 0.0f;
00450     static double coef = COEF;
00451     accum = accum - accum / coef + raw;
00452     return accum / coef;
00453 }
```

4.2.2.17 drive1_filter()

```
double drive1_filter (
            const double raw)
```

```
00456 {
00457     static double accum = 0.0f;
00458     static double coef = COEFF;
00459     accum = accum - accum / coef + raw;
00460     return accum / coef;
00461 }
```

4.2.2.18 error_filter()

```
double error_filter (
            const double raw)  [static]
```

```
00393 {
00394     static double accum = 0.0f;
00395     accum = accum - accum / L.coef + raw;
00396     return accum / L.coef;
00397 }
```

4.2.2.19 get_bat_runtime()

```
double get_bat_runtime (
            void )
```

```
00211 {
00212     return L.bat_runtime;
00213 }
```

4.2.2.20 get_batc_dev()

```
double get_batc_dev (
            void )
```

```
00216 {
00217     return L.batc_std_dev;
00218 }
```

4.2.2.21 gti_test()

```
double gti_test (
              void )
```

```
00180 {
00181     static bool once = true;
00182     // check for 48VDC AC charger powered from the Solar battery bank AC inverter
00183     if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
00184         L.gti_weight = 0.0f; // reduce power to zero
00185 #ifdef BSOC_DEBUG
00186         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);
00187 #endif
00188     } else {
00189         if (E.dl_excess) {
00190             if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
00191                 L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
00192                 once = true;
00193             } else {
00194                 L.gti_weight = 0.0f; // reduce power to zero
00195                 if (once) {
00196                     fprintf(fout, "%s Dump Load Battery Ah below %f \n", log_time(false),
      PV_DL_B_AH_MIN);
00197                     once = false;
00198                 }
00199             }
00200         }
00201     }
00202     return dc0_filter(L.gti_weight);
00203 }
```

4.2.3 Variable Documentation

4.2.3.1 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,
    .bat_runtime = 0.0f,
}
```

```
00045                               {
00046     .ac_weight = 0.0f,
00047     .bat_current = 0.0f,
00048     .bat_voltage = 0.0f,
00049     .batc_std_dev = 0.0f,
00050     .coef = COEF,
00051     .gti_weight = 0.0f,
00052     .pv_voltage = 0.0f,
00053     .bat_runtime = 0.0f,
00054 };
```

4.2.3.2 mqtt_name

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

```
00003                                   {
00004     "pccmode",
00005     "batenergykw",
00006     "runtime",
00007     "bamps",
00008     "bvolts",
00009     "load",
00010     "solar",
00011     "acenergy",
```

```
00012     "benergy",
00013     "pwatts",
00014     "pamps",
00015     "pvolts",
00016     "flast",
00017     "HAdcsw",
00018     "HAacsw",
00019     "HAshut",
00020     "HAmode",
00021     "HAcon0",
00022     "HAcon1",
00023     "HAcon2",
00024     "HAcon3",
00025     "HAcon4",
00026     "HAcon5",
00027     "HAcon6",
00028     "HAcon7",
00029     "DLv_pv",
00030     "DLp_pv",
00031     "DLp_bat",
00032     "DLv_bat",
00033     "DLc_mppt",
00034     "DLp_mppt",
00035     "DLah_bat",
00036     "DLccmode",
00037     "DLgti",
00038 };
```

4.3 bsoc.h

```
00001 /*
00002  * File:   bsoc.h
00003  * Author: root
00004  *
00005  * Created on February 10, 2024, 6:24 PM
00006  */
00007
00008 #ifndef BSOC_H
00009 #define BSOC_H
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014 #include <math.h>
00015     //#define BSOC_DEBUG
00016
00017 #define MIN_PV_VOLTS    5.0f
00018 #define MIN_BAT_VOLTS   23.0f
00019 #define MIN_BAT_KW      4100.0f
00020
00021 #define DEV_SIZE        10
00022 #define MAX_BATC_DEV    1.5f
00023 #define BAT_C_DRAW      3.0f
00024
00025 #define PBAL_OFFSET     -50.0f // postive bias for control point
00026 #define PV_FULL_PWR     300.0f
00027 #define PV_MIN_PWR      160.0f
00028 #define PV_V_NOM        60.0f
00029 #define PV_V_FAKE       0.336699f
00030
00031 #define COEF            8.0f
00032 #define COEFN           4.0f
00033 #define COEFF           2.0f
00034
00035 #include <stdlib.h>
00036 #include <stdio.h> /* for printf() */
00037 #include <unistd.h>
00038 #include <stdint.h>
00039 #include <string.h>
00040 #include <stdbool.h>
00041 #include <signal.h>
00042 #include <time.h>
00043 #include <sys/wait.h>
00044 #include <sys/types.h>
00045 #include <sys/time.h>
00046 #include <errno.h>
00047 #include <math.h>
00048 #include "pid.h"
00049 #include "mqtt_rec.h"
00050
00051     bool bsoc_init(void);
00052     bool bsoc_data_collect(void);
00053     double bsoc_ac(void);
```

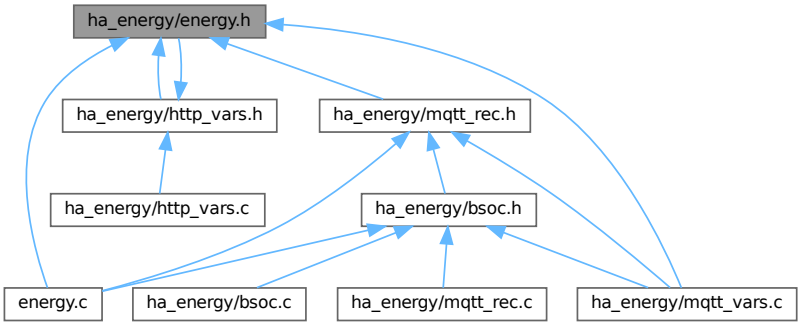
```
00054     double bsoc_gti(void);
00055     double gti_test(void);
00056     double ac_test(void);
00057     double get_batc_dev(void);
00058     bool bat_current_stable(void);
00059     void bsoc_set_std_dev(const double, const uint32_t);
00060     double get_bat_runtime(void);
00061 
00062     double calculateStandardDeviation(const uint32_t, const double *);
00063 
00064     bool bsoc_set_mode(const double, const bool, const bool);
00065 
00066     double ac0_filter(const double);
00067     double ac1_filter(const double);
00068     double ac2_filter(const double);
00069     double dc0_filter(const double);
00070     double dc1_filter(const double);
00071     double dc2_filter(const double);
00072     double drive0_filter(const double);
00073     double drive1_filter(const double);
00074 
00075 #ifdef __cplusplus
00076 }
00077 #endif
00078 
00079 #endif /* BSOC_H */
00080 
```

4.4 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 "http_vars.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** "V0.76"
- #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 **MQTT_RECONN** 3
- #define **KAI** 60
- #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** 0.68f
- #define **BAT_SOC_LOW_AC** 0.72f
- #define **BAT_CRITICAL** 746.0f
- #define **BAT_RUNTIME_LOW** 5.0f
- #define **BAT_RUNTIME_GTI** 6.0f
- #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** 120.0f
- #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 **DL_POWER_ZERO** "V0000X"
- #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 **BAMPS_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
- #define **L4_P** L3_P+IA_LAST

Enumerations

- enum **client_id** { **ID_C1** , **ID_C2** , **ID_C3** }
- 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_S** ,
  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_DCMPPT** , **V_DPMPPT** , **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_DCMPPT** , **S_DPMPPT** , **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)
- 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**

4.4.1 Enumeration Type Documentation

4.4.1.1 client_id

```
enum client_id
00199                   {
00200         ID_C1,
00201         ID_C2,
00202         ID_C3,
00203     };
```

4.4.1.2 energy_state

```
enum energy_state
00205                       {
00206         E_INIT,
00207         E_RUN,
00208         E_WAIT,
00209         E_IDLE,
00210         E_STOP,
00211         E_LAST,
00212     };
```

4.4.1.3 iammeter_id

```
enum iammeter_id
00231                     {
00232         IA_VOLTAGE,
00233         IA_CURRENT,
00234         IA_POWER,
00235         IA_IMPORT,
00236         IA_EXPORT,
00237         IA_FREQ,
00238         IA_PF,
00239         IA_LAST,
00240     };
```

4.4.1.4 iammeter_phase

```
enum iammeter_phase
00223                         {
00224         PHASE_A,
00225         PHASE_B,
00226         PHASE_C,
00227         PHASE_S,
00228         PHASE_LAST,
00229     };
```

4.4.1.5 mqtt_vars

```
enum mqtt_vars
00242                    {
00243         V_FCCM,
00244         V_FBEKW,
00245         V_FRUNT,
00246         V_FBAMPS,
00247         V_FBV,
00248         V_FLO,
00249         V_FSO,
00250         V_FACE,
00251         V_BEN,
00252         V_PWA,
00253         V_PAMPS,
00254         V_PVOLTS,
00255         V_FLAST,
00256         V_HDCSW,
00257         V_HACSW,
00258         V_HSHUT,
00259         V_HMODE,
00260         V_HCON0,
00261         V_HCON1,
00262         V_HCON2,
00263         V_HCON3,
00264         V_HCON4,
00265         V_HCON5,
00266         V_HCON6,
00267         V_HCON7,
00268         // add other data ranges here
00269         V_DVPV,
00270         V_DPPV,
00271         V_DPBAT,
00272         V_DVBAT,
00273         V_DCMPPT,
00274         V_DPMPPT,
00275         V_DAHBAT,
00276         V_DCCMODE,
00277         V_DGTI,
00278         V_DLAST,
00279     };
```

4.4.1.6 running_state

```
enum running_state
00214                        {
00215         R_INIT,
00216         R_FLOAT,
00217         R_SLEEP,
00218         R_RUN,
00219         R_IDLE,
00220         R_LAST,
00221     };
```

4.4.1.7 sane_vars

```
enum sane_vars
00281                   {
```

```
00282         S_FCCM,
00283         S_FBEKW,
00284         S_FRUNT,
00285         S_FBAMPS,
00286         S_FBV,
00287         S_FLO,
00288         S_FSO,
00289         S_FACE,
00290         S_BEN,
00291         S_PWA,
00292         S_PAMPS,
00293         S_PVOLTS,
00294         S_FLAST,
00295         S_HDCSW,
00296         S_HACSW,
00297         S_HSHUT,
00298         S_HMODE,
00299         // add other data ranges here
00300         S_DVPV,
00301         S_DPPV,
00302         S_DPBAT,
00303         S_DVBAT,
00304         S_DCMPPT,
00305         S_DPMPPT,
00306         S_DAHBAT,
00307         S_DCCMODE,
00308         S_DGTI,
00309         S_DLAST,
00310     };
```

4.4.2 Function Documentation

4.4.2.1 connlost()

```
void connlost (
            void * context,
            char * cause)
```

trouble in River-city

```
00302 {
00303     struct ha_flag_type *ha_flag = context;
00304     int32_t id_num = ha_flag->ha_id;
00305     static uint32_t times = 0;
00306     char * where = "Context is NULL";
00307     char * what = "Reconnection Retry";
00308 
00309     // bug-out if no context variables passed to callback
00310     if (context == NULL) {
00311         id_num = -1;
00312         goto bugout;
00313     }
00314 
00315     switch (ha_flag->cid) {
00316     case ID_C1:
00317         where = TOPIC_SS;
00318         break;
00319     case ID_C2:
00320         where = TOPIC_SD;
00321         break;
00322     case ID_C3:
00323         where = TOPIC_HA;
00324         break;
00325     }
00326 
00327 
00328     if (times++ > MQTT_RECONN) {
00329         goto bugout;
00330     } else {
00331         if (times > 1) {
00332             fprintf(fout, "%s Connection lost, retrying %d \n", log_time(false), times);
00333             fprintf(fout, "%s Cause: %s, h_id %d, c_id %d, %s \n", log_time(false), cause, id_num,
      ha_flag->cid, what);
00334             fprintf(fout, "%s MQTT DAEMON reconnection failure  LOG Version %s : MQTT Version %s\n",
      log_time(false), LOG_VERSION, MQTT_VERSION);
00335         }
00336         fflush(fout);
00337         times = 0;
00338         return;
```

```
00339     }
00340
00341 bugout:
00342     fprintf(fout, "%s Connection lost, exit ha_energy program\n", log_time(false));
00343     fprintf(fout, "%s Cause: %s, h_id %d, c_id %d, %s \n", log_time(false), cause, id_num,
      ha_flag->cid, where);
00344     fprintf(fout, "%s MQTT DAEMON context is null failure  LOG Version %s : MQTT Version %s\n",
      log_time(false), LOG_VERSION, MQTT_VERSION);
00345     fflush(fout);
00346     exit(EXIT_FAILURE);
00347 }
```

4.4.2.2 ha_ac_off()

```
void ha_ac_off (
              void )
01015 {
01016     mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
01017     E.ac_sw_status = false;
01018 }
```

4.4.2.3 ha_ac_on()

```
void ha_ac_on (
              void )
01021 {
01022     mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
01023     E.ac_sw_status = true;
01024 }
```

4.4.2.4 ha_dc_off()

```
void ha_dc_off (
              void )
01030 {
01031     mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
01032     E.gti_sw_status = false;
01033 }
```

4.4.2.5 ha_dc_on()

```
void ha_dc_on (
              void )
01036 {
01037     mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
01038     E.gti_sw_status = true;
01039 }
```

4.4.2.6 log_time()

```
char * log_time (
            bool log)
```

```
01100 {
01101     static char time_log[RBUF_SIZ] = {0};
01102     static uint32_t len = 0, sync_time = TIME_SYNC_SEC - 1;
01103     time_t rawtime_log;
01104
01105     tzset();
01106     timezone = 0;
01107     daylight = 0;
01108     time(&rawtime_log);
01109     if (sync_time++ > TIME_SYNC_SEC) {
01110         sync_time = 0;
01111         snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
      time commands
01112         mqtt_gti_time(E.client_p, TOPIC_P, time_log);
01113     }
01114
01115     sprintf(time_log, "%s", ctime(&rawtime_log));
01116     len = strlen(time_log);
01117     time_log[len - 1] = 0; // munge out the return character
01118     if (log) {
01119         fprintf(fout, "%s ", time_log);
01120         fflush(fout);
01121     }
01122
01123     return time_log;
01124 }
```

4.4.2.7 log_timer()

```
bool log_timer (
            void )
```

```
01159 {
01160     bool itstime = false;
01161
01162     if (E.log_spam < LOW_LOG_SPAM) {
01163         E.log_time_reset = 0;
01164         itstime = true;
01165     }
01166     if (E.log_time_reset > RESET_LOG_SPAM) {
01167         E.log_spam = 0;
01168         itstime = true;
01169     }
01170     return itstime;
01171 }
```

4.4.2.8 ramp_down_ac()

```
void ramp_down_ac (
            MQTTClient client_p,
            bool sw_off)
```

```
01005 {
01006     if (sw_off) {
01007         mqtt_ha_switch(client_p, TOPIC_PACC, false);
01008         E.ac_sw_status = false;
01009         usleep(200000);
01010     }
01011     E.once_ac = true;
01012 }
```

4.4.2.9 ramp_down_gti()

```
void ramp_down_gti (
            MQTTClient client_p,
            bool sw_off)
```

```
00966 {
00967     static uint32_t times = 0;
00968     if (sw_off) {
00969         mqtt_ha_switch(client_p, TOPIC_PDCC, false);
00970         E.once_gti_zero = true;
00971         times = 0;
00972         E.gti_sw_status = false;
00973     }
00974     E.once_gti = true;
00975
00976     if (E.once_gti_zero) {
00977         mqtt_gti_power(client_p, TOPIC_P, DL_POWER_ZERO, 7); // zero power
00978         if (times++ < LOW_LOG_SPAM) {
00979             E.once_gti_zero = false;
00980         }
00981     }
00982 }
```

4.4.2.10 ramp_up_ac()

```
void ramp_up_ac (
             MQTTClient client_p,
             bool start)
```

```
00988 {
00989
00990     if (start) {
00991         E.once_ac = true;
00992     }
00993
00994     if (E.once_ac) {
00995         if (get_bat_runtime() > BAT_RUNTIME_GTI) {
00996             E.once_ac = false;
00997             mqtt_ha_switch(client_p, TOPIC_PACC, true);
00998             E.ac_sw_status = true;
00999             usleep(200000); // wait for voltage to ramp
01000         }
01001     }
01002 }
```

4.4.2.11 ramp_up_gti()

```
void ramp_up_gti (
             MQTTClient client_p,
             bool start,
             bool excess)
```

```
00899 {
00900     static uint32_t sequence = 0;
00901
00902     if (start) {
00903         E.once_gti = true;
00904     }
00905
00906     if (E.once_gti) {
00907         E.once_gti = false;
00908         sequence = 0;
00909         if (!excess) {
00910             if (get_bat_runtime() > BAT_RUNTIME_GTI) {
00911                 mqtt_ha_switch(client_p, TOPIC_PDCC, true);
00912                 E.gti_sw_status = true;
00913                 usleep(500000); // wait for PS voltage to ramp
00914             }
00915         } else {
00916             sequence = 1;
00917         }
00918     }
00919
00920     switch (sequence) {
00921     case 4:
00922         E.once_gti_zero = true;
00923         break;
00924     case 3:
00925     case 2:
00926     case 1:
00927         E.once_gti_zero = true;
```

```
00928         if (bat_current_stable() || E.dl_excess) { // check battery current std dev, stop
      'motorboating'
00929             sequence++;
00930             if (get_bat_runtime() > BAT_RUNTIME_GTI) {
00931                 if (!mqtt_gti_power(client_p, TOPIC_P, "+#", 3)) {
00932                     sequence = 0;
00933                 }; // +100W power
00934             } else {
00935                 sequence = 0;
00936             }
00937         } else {
00938             usleep(500000); // wait a bit more for power to be stable
00939             sequence = 1; // do power ramps when ready
00940             if (!mqtt_gti_power(client_p, TOPIC_P, "-#", 4)) {
00941                 sequence = 0;
00942             }; // - 100W power
00943         }
00944         break;
00945     case 0:
00946         sequence++;
00947         if (E.once_gti_zero) {
00948             mqtt_gti_power(client_p, TOPIC_P, DL_POWER_ZERO, 5); // zero power
00949             E.once_gti_zero = false;
00950         }
00951         break;
00952     default:
00953         if (E.once_gti_zero) {
00954             mqtt_gti_power(client_p, TOPIC_P, DL_POWER_ZERO, 6); // zero power
00955             E.once_gti_zero = false;
00956         }
00957         sequence = 0;
00958         break;
00959     }
00960 }
```

4.4.2.12 sanity_check()

```
bool sanity_check (
             void )
```

```
00258 {
00259     if (E.mvar[V_PWA] > PWA_SANE) {
00260         E.sane = S_PWA;
00261         return false;
00262     }
00263     if (E.mvar[V_PAMPS] > PAMPS_SANE) {
00264         E.sane = S_PAMPS;
00265         return false;
00266     }
00267     if (E.mvar[V_PVOLTS] > PVOLTS_SANE) {
00268         E.sane = S_PVOLTS;
00269         return false;
00270     }
00271     if (E.mvar[V_FBAMPS] > BAMPS_SANE) {
00272         E.sane = S_FBAMPS;
00273         return false;
00274     }
00275     return true;
00276 }
```

4.4.2.13 sync_ha()

```
bool sync_ha (
             void )
```

```
01130 {
01131     bool sync = false;
01132     if (E.gti_sw_status != (bool) ((int32_t) E.mvar[V_HDCSW])) {
01133         fprintf(fout, "DC_MM %d %d ", (bool) E.gti_sw_status, (bool) ((int32_t) E.mvar[V_HDCSW]));
01134         mqtt_ha_switch(E.client_p, TOPIC_PDCC, !E.gti_sw_status);
01135         E.dc_mismatch = true;
01136         fflush(fout);
01137         sync = true;
01138     } else {
01139         E.dc_mismatch = false;
01140     }
01141
01142     E.ac_sw_status = (bool) ((int32_t) E.mvar[V_HACSW]); // TEMP FIX for MISmatch errors
```

```
01143     if (E.ac_sw_status != (bool) ((int32_t) E.mvar[V_HACSW])) {
01144         fprintf(fout, "AC_MM %d %d ", (bool) E.ac_sw_status, (bool) ((int32_t) E.mvar[V_HACSW]));
01145         mqtt_ha_switch(E.client_p, TOPIC_PACC, !E.ac_sw_status);
01146         E.ac_mismatch = true;
01147         fflush(fout);
01148         sync = true;
01149     } else {
01150         E.ac_mismatch = false;
01151     }
01152     return sync;
01153 }
```

4.4.2.14 timer_callback()

```
void timer_callback (
             int32_t signum)
```

```
00287 {
00288     signal(signum, timer_callback);
00289     ha_flag_vars_ss.runner = true;
00290     E.ten_sec_clock++;
00291     E.log_spam++;
00292     E.log_time_reset++;
00293     if (E.log_spam > MAX_LOG_SPAM) {
00294         E.log_spam = 0;
00295     }
00296 }
```

4.4.3 Variable Documentation

4.4.3.1 E

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

```
00111                       {
00112     .once_gti = true,
00113     .once_ac = true,
00114     .once_gti_zero = true,
00115     .iammeter = false,
00116     .fm80 = false,
00117     .dumpload = false,
00118     .homeassistant = false,
00119     .ac_low_adj = 0.0f,
00120     .gti_low_adj = 0.0f,
00121     .ac_sw_on = true,
00122     .gti_sw_on = true,
00123     .im_delay = 0,
00124     .gti_delay = 0,
00125     .im_display = 0,
00126     .rc = 0,
00127     .speed_go = 0,
00128     .mode.pid.iMax = PV_IMAX,
00129     .mode.pid.iMin = 0.0f,
00130     .mode.pid.pGain = PV_PGAIN,
00131     .mode.pid.iGain = PV_IGAIN,
00132     .mode.mode_tmr = 0,
00133     .mode.mode = true,
00134     .mode.in_pid_control = false,
00135     .mode.dl_mqtt_max = PV_DL_MPTT_MAX,
00136     .mode.E = E_INIT,
00137     .mode.R = R_INIT,
00138     .mode.no_float = true,
00139     .mode.data_error = false,
00140     .ac_sw_status = false,
00141     .gti_sw_status = false,
00142     .solar_mode = false,
00143     .solar_shutdown = false,
00144     .mode.pv_bias = PV_BIAS_LOW,
00145     .sane = S_DLAST,
00146     .startup = true,
00147     .ac_mismatch = false,
00148     .dc_mismatch = false,
00149     .mode_mismatch = false,
00150     .link.shutdown = 0,
00151     .mode.bat_crit = false,
00152     .dl_excess = false,
00153     .dl_excess_adj = 0.0f,
00154 };
```

4.4.3.2 ha_flag_vars_ss

```
struct ha_flag_type ha_flag_vars_ss  [extern]
00075                                    {
00076     .runner = false,
00077     .receivedtoken = false,
00078     .deliveredtoken = false,
00079     .rec_ok = false,
00080     .ha_id = FM80_ID,
00081     .var_update = 0,
00082     .energy_mode = NORM_MODE,
00083 };
```

4.5 energy.h

Go to the documentation of this file.

```
00001
00004
00005 #ifndef BMC_H
00006 #define BMC_H
00007
00008 #ifdef __cplusplus
00009 extern "C" {
00010 #endif
00011 #include <stdlib.h>
00012 #include <stdio.h> /* for printf() */
00013 #include <unistd.h>
00014 #include <stdint.h>
00015 #include <string.h>
00016 #include <stdbool.h>
00017 #include <signal.h>
00018 #include <time.h>
00019 #include <sys/wait.h>
00020 #include <sys/types.h>
00021 #include <sys/time.h>
00022 #include <errno.h>
00023 #include <cjson/cJSON.h>
00024 #include <curl/curl.h>
00025 #include <pthread.h>
00026 #include <sys/stat.h>
00027 #include <syslog.h>
00028 #include <arpa/inet.h>
00029 #include <sys/socket.h>
00030 #include <netdb.h>
00031 #include <ifaddrs.h>
00032 #include "MQTTClient.h"
00033 #include "pid.h"
00034 #include "http_vars.h"
00035
00036
00037 #define LOG_VERSION     "V0.76"
00038 #define MQTT_VERSION    "V3.11"
00039 #define TNAME  "maint9"
00040 #define LADDRESS        "tcp://127.0.0.1:1883"
00041 #ifdef __amd64
00042 #define ADDRESS         "tcp://10.1.1.172:1883"
00043 #else
00044 #define ADDRESS         "tcp://10.1.1.30:1883"
00045 #endif
00046 #define CLIENTID1       "Energy_Mqtt_HA1"
00047 #define CLIENTID2       "Energy_Mqtt_HA2"
00048 #define CLIENTID3       "Energy_Mqtt_HA3"
00049 #define TOPIC_P         "mateq84/data/gticmd"
00050 #define TOPIC_SPAM      "mateq84/data/spam"
00051 #define TOPIC_PACA      "home-assistant/gtiac/availability"
00052 #define TOPIC_PDCA      "home-assistant/gtidc/availability"
00053 #define TOPIC_PACC      "home-assistant/gtiac/contact"
00054 #define TOPIC_PDCC      "home-assistant/gtidc/contact"
00055 #define TOPIC_PPID      "home-assistant/solar/pid"
00056 #define TOPIC_SHUTDOWN  "home-assistant/solar/shutdown"
00057 #define TOPIC_SS        "mateq84/data/solar"
00058 #define TOPIC_SD        "mateq84/data/dumpload"
00059 #define TOPIC_HA        "home-assistant/status/switch"
00060 #define QOS             1
00061 #define TIMEOUT         10000L
00062 #define SPACING_USEC    500 * 1000
00063 #define USEC_SEC        1000000L
00064
```

```
00065 #define DAQ_STR 32
00066 #define DAQ_STR_M DAQ_STR-1
00067
00068 #define SBUF_SIZ        16  // short buffer string size
00069 #define RBUF_SIZ        82
00070 #define SYSLOG_SIZ      512
00071
00072 #define MQTT_TIMEOUT    900
00073 #define SW_QOS          1
00074
00075 #define MQTT_RECONN     3
00076 #define KAI             60
00077
00078 #define NO_CYLON
00079 #define CRITIAL_SHUTDOWN_LOG
00080
00081 #define UNIT_TEST       2
00082 #define NORM_MODE       0
00083 #define PID_MODE        1
00084 #define MAX_ERROR       5
00085 #define IAM_DELAY       120
00086
00087 #define CMD_SEC         10
00088 #define TIME_SYNC_SEC   30
00089
00090     /*
00091      * Battery SoC cycle limits parameters
00092      */
00093 #define BAT_M_KW            5120.0f
00094 #define BAT_SOC_TOP         0.98f
00095 #define BAT_SOC_HIGH        0.95f
00096 #define BAT_SOC_LOW         0.68f
00097 #define BAT_SOC_LOW_AC      0.72f
00098 #define BAT_CRITICAL        746.0f
00099 #define BAT_RUNTIME_LOW     5.0f
00100 #define BAT_RUNTIME_GTI     6.0f
00101 #define MIN_BAT_KW_BSOC_SLP 4000.0f
00102 #define MIN_BAT_KW_BSOC_HI  4550.0f
00103
00104 #define MIN_BAT_KW_GTI_HI   BAT_M_KW*BAT_SOC_TOP
00105 #define MIN_BAT_KW_GTI_LO   BAT_M_KW*BAT_SOC_LOW
00106
00107 #define MIN_BAT_KW_AC_HI    BAT_M_KW*BAT_SOC_HIGH
00108 #define MIN_BAT_KW_AC_LO    BAT_M_KW*BAT_SOC_LOW_AC
00109
00110     /*
00111      * PV panel cycle limits parameters
00112      */
00113 #define PV_PGAIN            0.85f
00114 #define PV_IGAIN            0.12f
00115 #define PV_IMAX             1400.0f
00116 #define PV_BIAS             288.0f
00117 #define PV_BIAS_ZERO          0.0f
00118 #define PV_BIAS_LOW         222.0f
00119 #define PV_BIAS_FLOAT       399.0f
00120 #define PV_BIAS_SLEEP       480.0f
00121 #define PV_BIAS_RATE        320.0f
00122 #define PV_DL_MPTT_MAX     1200.0f
00123 #define PV_DL_MPTT_EXCESS  1300.0f
00124 #define PV_DL_MPTT_IDLE     57.0f
00125 #define PV_DL_BIAS_RATE     75.0f
00126 #define PV_DL_EXCESS       500.0f
00127 #define PV_DL_B_AH_LOW     100.0f
00128 #define PV_DL_B_AH_MIN     120.0f // DL battery should be at least 175Ah
00129 #define PV_DL_B_V_LOW       23.8f // Battery low-voltqage cutoff
00130 #define PWA_SLEEP          200.0f
00131 #define DL_AC_DC_EFF        1.24f
00132
00133     /*
00134      * Energy control loop parameters
00135      */
00136 #define BAL_MIN_ENERGY_AC   -200.0f
00137 #define BAL_MAX_ENERGY_AC   200.0f
00138 #define BAL_MIN_ENERGY_GTI  -1400.0f
00139 #define BAL_MAX_ENERGY_GTI  200.0f
00140
00141 #define DL_POWER_ZERO  "V0000X"
00142
00143 #define LOG_TO_FILE         "/store/logs/energy.log"
00144 #define LOG_TO_FILE_ALT     "/tmp/energy.log"
00145
00146 #define MAX_LOG_SPAM  60
00147 #define LOW_LOG_SPAM  2
00148 #define RESET_LOG_SPAM  120
00149
00150     //#define IM_DEBUG                    // WEM3080T LOGGING
00151     //#define B_ADJ_DEBUG                 // debug printing
```

```
00152     //#define FAKE_VPV                      // NO AC CHARGER for DUMPLOAD, batteries are
      cross-connected to a parallel bank
00153     //#define PSW_DEBUG
00154     //#define DEBUG_SHUTDOWN
00155
00156     //#define AUTO_CHARGE                   // turn on dumpload charger during restarts
00157     //#define B_DLE_DEBUG    // Dump Load debugging
00158     //#define BSOC_DEGUB
00159     //#define DEBUG_HA_CMD
00160
00161 #define IM_DELAY            1   // tens of second updates
00162 #define IM_DISPLAY          1
00163 #define GTI_DELAY           1
00164
00165     /*
00166      * sane limits for system data elements
00167      */
00168 #define PWA_SANE            1700.0f
00169 #define PAMPS_SANE          16.0f
00170 #define PVOLTS_SANE         150.0f
00171 #define BAMPS_SANE          70.0f
00172
00173     /*
00174         Three Phase WiFi Energy Meter (WEM3080T)
00175     name    Unit    Description
00176     wem3080t_voltage_a  V   A phase voltage
00177     wem3080t_current_a  A   A phase current
00178     wem3080t_power_a    W   A phase active power
00179     wem3080t_importenergy_a kWh A phase import energy
00180     wem3080t_exportgrid_a   kWh A phase export energy
00181     wem3080t_frequency_a    kWh A phase frequency
00182     wem3080t_pf_a   kWh A phase power factor
00183     wem3080t_voltage_b  V   B phase voltage
00184     wem3080t_current_b  A   B phase current
00185     wem3080t_power_b    W   B phase active power
00186     wem3080t_importenergy_b kWh B phase import energy
00187     wem3080t_exportgrid_b   kWh B phase export energy
00188     wem3080t_frequency_b    kWh B phase frequency
00189     wem3080t_pf_b   kWh B phase power factor
00190     wem3080t_voltage_c  V   C phase voltage
00191     wem3080t_current_c  A   C phase current
00192     wem3080t_power_c    W   C phase active power
00193     wem3080t_importenergy_c kWh C phase import energy
00194     wem3080t_exportgrid_c   kWh C phase export energy
00195     wem3080t_frequency_c    kWh C phase frequency
00196     wem3080t_pf_c   kWh C phase power factor
00197      */
00198
00199     enum client_id {
00200         ID_C1,
00201         ID_C2,
00202         ID_C3,
00203     };
00204
00205     enum energy_state {
00206         E_INIT,
00207         E_RUN,
00208         E_WAIT,
00209         E_IDLE,
00210         E_STOP,
00211         E_LAST,
00212     };
00213
00214     enum running_state {
00215         R_INIT,
00216         R_FLOAT,
00217         R_SLEEP,
00218         R_RUN,
00219         R_IDLE,
00220         R_LAST,
00221     };
00222
00223     enum iammeter_phase {
00224         PHASE_A,
00225         PHASE_B,
00226         PHASE_C,
00227         PHASE_S,
00228         PHASE_LAST,
00229     };
00230
00231     enum iammeter_id {
00232         IA_VOLTAGE,
00233         IA_CURRENT,
00234         IA_POWER,
00235         IA_IMPORT,
00236         IA_EXPORT,
00237         IA_FREQ,
```

```
00238         IA_PF,
00239         IA_LAST,
00240     };
00241
00242     enum mqtt_vars {
00243         V_FCCM,
00244         V_FBEKW,
00245         V_FRUNT,
00246         V_FBAMPS,
00247         V_FBV,
00248         V_FLO,
00249         V_FSO,
00250         V_FACE,
00251         V_BEN,
00252         V_PWA,
00253         V_PAMPS,
00254         V_PVOLTS,
00255         V_FLAST,
00256         V_HDCSW,
00257         V_HACSW,
00258         V_HSHUT,
00259         V_HMODE,
00260         V_HCON0,
00261         V_HCON1,
00262         V_HCON2,
00263         V_HCON3,
00264         V_HCON4,
00265         V_HCON5,
00266         V_HCON6,
00267         V_HCON7,
00268         // add other data ranges here
00269         V_DVPV,
00270         V_DPPV,
00271         V_DPBAT,
00272         V_DVBAT,
00273         V_DCMPPT,
00274         V_DPMPPT,
00275         V_DAHBAT,
00276         V_DCCMODE,
00277         V_DGTI,
00278         V_DLAST,
00279     };
00280
00281     enum sane_vars {
00282         S_FCCM,
00283         S_FBEKW,
00284         S_FRUNT,
00285         S_FBAMPS,
00286         S_FBV,
00287         S_FLO,
00288         S_FSO,
00289         S_FACE,
00290         S_BEN,
00291         S_PWA,
00292         S_PAMPS,
00293         S_PVOLTS,
00294         S_FLAST,
00295         S_HDCSW,
00296         S_HACSW,
00297         S_HSHUT,
00298         S_HMODE,
00299         // add other data ranges here
00300         S_DVPV,
00301         S_DPPV,
00302         S_DPBAT,
00303         S_DVBAT,
00304         S_DCMPPT,
00305         S_DPMPPT,
00306         S_DAHBAT,
00307         S_DCCMODE,
00308         S_DGTI,
00309         S_DLAST,
00310     };
00311
00312 #define MAX_IM_VAR  IA_LAST*PHASE_LAST
00313
00314 #define L1_P    IA_POWER
00315 #define L2_P    L1_P+IA_LAST
00316 #define L3_P    L2_P+IA_LAST
00317 #define L4_P    L3_P+IA_LAST
00318
00319     struct link_type {
00320         volatile uint32_t iammeter_error, iammeter_count;
00321         volatile uint32_t mqtt_error, mqtt_count;
00322         volatile uint32_t shutdown;
00323     };
00324
```

```
00325     struct mode_type {
00326         volatile double error, target, total_system, gti_dumpload, pv_bias, dl_mqtt_max, off_grid,
      sequence;
00327         volatile bool mode, in_pid_control, con0, con1, con2, con3, con4, con5, con6, con7, no_float,
      data_error, bat_crit;
00328         volatile uint32_t mode_tmr;
00329         volatile struct SPid pid;
00330         volatile enum energy_state E;
00331         volatile enum running_state R;
00332     };
00333 
00334     struct energy_type {
00335         volatile double print_vars[MAX_IM_VAR];
00336         volatile double im_vars[IA_LAST][PHASE_LAST];
00337         volatile double mvar[V_DLAST + 1];
00338         volatile bool once_gti, once_ac, iammeter, fm80, dumpload, homeassistant, once_gti_zero;
00339         volatile double gti_low_adj, ac_low_adj, dl_excess_adj, bat_runtime;
00340         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;
00341         volatile uint32_t speed_go, im_delay, im_display, gti_delay;
00342         volatile int32_t rc, sane;
00343         volatile uint32_t ten_sec_clock, log_spam, log_time_reset;
00344         pthread_mutex_t ha_lock;
00345         struct mode_type mode;
00346         struct link_type link;
00347         MQTTClient client_p, client_sd, client_ha;
00348     };
00349 
00350     extern struct energy_type E;
00351     extern struct ha_flag_type ha_flag_vars_ss;
00352     extern FILE* fout;
00353 
00354     void timer_callback(int32_t);
00355     void connlost(void *, char *);
00356 
00357     void ramp_up_gti(MQTTClient, bool, bool);
00358     void ramp_up_ac(MQTTClient, bool);
00359     void ramp_down_gti(MQTTClient, bool);
00360     void ramp_down_ac(MQTTClient, bool);
00361     void ha_ac_off(void);
00362     void ha_ac_on(void);
00363     void ha_dc_off(void);
00364     void ha_dc_on(void);
00365 
00366     bool sanity_check(void);
00367     char * log_time(bool);
00368     bool sync_ha(void);
00369     bool log_timer(void);
00370 
00371 #ifdef __cplusplus
00372 }
00373 #endif
00374 
00375 #endif /* BMC_H */
00376 
```

4.6 ha_energy/http_vars.c File Reference

```
#include "http_vars.h"
#include <time.h>
```

Include dependency graph for http_vars.c:

Functions

- static void iammeter_get_data (const double, const uint32_t, const uint32_t)
- bool mqtt_gti_time (MQTTClient, const char *, char *)

- size_t iammeter_write_callback1 (char *buffer, size_t size, size_t nitems, void *stream)
- size_t iammeter_write_callback2 (char *buffer, size_t size, size_t nitems, void *stream)
- void iammeter_read1 (const char *meter)
- void iammeter_read2 (const char *meter)
- void print_im_vars (void)

Variables

- static CURL * **curl**
- static CURLcode **res**

4.6.1 Detailed Description

read and format data returned from libcurl http WRITEDATA function call

4.6.2 Function Documentation

4.6.2.1 iammeter_get_data()

```
void iammeter_get_data (
            const double valuedouble,
            const uint32_t i,
            const uint32_t j)  [static]
```

```
00203 {
00204     E.im_vars[i][j] = valuedouble;
00205 }
```

4.6.2.2 iammeter_read1()

```
void iammeter_read1 (
            const char * meter)
```

```
00132 {
00133
00134     curl = curl_easy_init();
00135     if (curl) {
00136         E.link.iammeter_count++;
00137         curl_easy_setopt(curl, CURLOPT_URL, meter);
00138         curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, iammeter_write_callback1);
00139         curl_easy_setopt(curl, CURLOPT_WRITEDATA, E.print_vars); // external data array for iammeter
      values
00140         curl_easy_setopt(curl, CURLOPT_TCP_KEEPALIVE, 1L);
00141         /* set keep-alive idle time to 120 seconds */
00142         curl_easy_setopt(curl, CURLOPT_TCP_KEEPIDLE, 60L);
00143         /* interval time between keep-alive probes: 60 seconds */
00144         curl_easy_setopt(curl, CURLOPT_TCP_KEEPINTVL, 30L);
00145
00146         res = curl_easy_perform(curl);
00147         /* Check for errors */
00148         if (res != CURLE_OK) {
00149             if (res == CURLE_GOT_NOTHING) {
00150                 E.iammeter = false;
00151             } else {
00152                 fprintf(fout, "%s curl_easy_perform() failed in iammeter_read1: %s %s\n",
      log_time(false),
00153                     curl_easy_strerror(res), meter);
00154                 fflush(fout);
00155                 E.iammeter = false;
00156                 E.link.iammeter_error++;
00157             }
00158         } else {
00159             E.iammeter = true;
00160         }
00161         curl_easy_cleanup(curl);
00162     }
00163 }
```

4.6.2.3 iammeter_read2()

```
void iammeter_read2 (
            const char * meter)
```

```
00166 {
00167
00168     curl = curl_easy_init();
00169     if (curl) {
00170         E.link.iammeter_count++;
00171         curl_easy_setopt(curl, CURLOPT_URL, meter);
00172         curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, iammeter_write_callback2);
00173         curl_easy_setopt(curl, CURLOPT_WRITEDATA, E.print_vars); // external data array for iammeter
      values
00174         curl_easy_setopt(curl, CURLOPT_TCP_KEEPALIVE, 1L);
00175         /* set keep-alive idle time to 120 seconds */
00176         curl_easy_setopt(curl, CURLOPT_TCP_KEEPIDLE, 60L);
00177         /* interval time between keep-alive probes: 60 seconds */
00178         curl_easy_setopt(curl, CURLOPT_TCP_KEEPINTVL, 30L);
00179
00180         res = curl_easy_perform(curl);
00181         /* Check for errors */
00182         if (res != CURLE_OK) {
00183             if (res == CURLE_GOT_NOTHING) {
00184                 E.iammeter = false;
00185             } else {
00186                 fprintf(fout, "%s curl_easy_perform() failed in iammeter_read1: %s %s\n",
      log_time(false),
00187                     curl_easy_strerror(res), meter);
00188                 fflush(fout);
00189                 E.iammeter = false;
00190                 E.link.iammeter_error++;
00191             }
00192         } else {
00193             E.iammeter = true;
00194         }
00195         curl_easy_cleanup(curl);
00196     }
00197 }
```

4.6.2.4 iammeter_write_callback1()

```
size_t iammeter_write_callback1 (
            char * buffer,
            size_t size,
            size_t nitems,
            void * stream)
```

```
00014 {
00015     cJSON *json = cJSON_ParseWithLength(buffer, strlen(buffer));
00016     struct energy_type * e = stream;
00017     uint32_t next_var = 0;
00018
00019     E.link.iammeter_count++;
00020
00021     if (json == NULL) {
00022         const char *error_ptr = cJSON_GetErrorPtr();
00023         E.link.iammeter_error++;
00024         if (error_ptr != NULL) {
00025             fprintf(fout, "%s Error in iammeter_write_callback1 %u: %s\n", log_time(false),
      E.link.iammeter_error, error_ptr);
00026             fflush(fout);
00027         }
00028         goto iammeter_exit;
00029     }
00030 #ifdef IM_DEBUG
00031     fprintf(fout, "\n iammeter_read_callback %s \n", buffer);
00032 #endif
00033
00034     cJSON *data_result = cJSON_GetObjectItemCaseSensitive(json, "Datas");
00035
00036     if (!data_result) {
00037         size = 0;
00038         nitems = 0;
00039         goto iammeter_exit;
00040     }
00041
00042     cJSON *jname;
00043     uint32_t phase = PHASE_A; // get data from whole house monitor
```

```
00044
00045     cJSON_ArrayForEach(jname, data_result)
00046     {
00047         cJSON *ianame;
00048 #ifdef IM_DEBUG
00049         fprintf(fout, "\n iammeter variables ");
00050 #endif
00051
00052         cJSON_ArrayForEach(ianame, jname)
00053         {
00054             uint32_t phase_var = IA_VOLTAGE;
00055             iammeter_get_data(ianame->valuedouble, phase_var, phase);
00056             e->print_vars[next_var++] = ianame->valuedouble;
00057 #ifdef IM_DEBUG
00058             fprintf(fout, "%8.2f ", im_vars[phase_var][phase]);
00059 #endif
00060             phase_var++;
00061         }
00062         phase++;
00063     }
00064 #ifdef IM_DEBUG
00065     fprintf(fout, "\n");
00066 #endif
00067
00068 iammeter_exit:
00069     cJSON_Delete(json);
00070     return size * nitems;
00071 }
```

4.6.2.5 iammeter_write_callback2()

```
size_t iammeter_write_callback2 (
            char * buffer,
            size_t size,
            size_t nitems,
            void * stream)
```

```
00074 {
00075     cJSON *json = cJSON_ParseWithLength(buffer, strlen(buffer));
00076     struct energy_type * e = stream;
00077     uint32_t next_var = PHASE_S*IA_LAST;
00078
00079     E.link.iammeter_count++;
00080
00081     if (json == NULL) {
00082         const char *error_ptr = cJSON_GetErrorPtr();
00083         E.link.iammeter_error++;
00084         if (error_ptr != NULL) {
00085             fprintf(fout, "%s Error in iammeter_write_callback2 %u: %s\n", log_time(false),
      E.link.iammeter_error, error_ptr);
00086             fflush(fout);
00087         }
00088         goto iammeter_exit;
00089     }
00090 #ifdef IM_DEBUG2
00091     fprintf(fout, "\n iammeter_read_callback %s, next_var %d, L4_P %d \n", buffer, next_var, L4_P);
00092 #endif
00093
00094     cJSON *data_result = cJSON_GetObjectItemCaseSensitive(json, "Data");
00095
00096     if (!data_result) {
00097         size = 0;
00098         nitems = 0;
00099         goto iammeter_exit;
00100     }
00101
00102     cJSON *jname;
00103     uint32_t phase = PHASE_S; // get data from server monitor
00104
00105 #ifdef IM_DEBUG2
00106     fprintf(fout, " iammeter variables ");
00107 #endif
00108
00109     cJSON_ArrayForEach(jname, data_result) // single phase of data
00110     {
00111         uint32_t phase_var = IA_VOLTAGE;
00112         iammeter_get_data(jname->valuedouble, phase_var, phase);
00113         e->print_vars[next_var++] = jname->valuedouble;
00114 #ifdef IM_DEBUG2
00115         fprintf(fout, " %8.2f ", jname->valuedouble);
00116 #endif
```

```
00117         phase_var++;
00118     }
00119 #ifdef IM_DEBUG2
00120     fprintf(fout, "\n");
00121 #endif
00122
00123 iammeter_exit:
00124     cJSON_Delete(json);
00125     return size * nitems;
00126 }
```

4.6.2.6 mqtt_gti_time()

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

```
00249 {
00250     bool ret = true;
00251     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00252     MQTTClient_deliveryToken token;
00253     ha_flag_vars_ss.deliveredtoken = 0;
00254
00255     E.link.mqtt_count++;
00256     pubmsg.payload = msg;
00257     pubmsg.payloadlen = strlen(msg);
00258     pubmsg.qos = QOS;
00259     pubmsg.retained = 0;
00260
00261     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run time commands
00262
00263     // a busy, wait loop for the async delivery thread to complete
00264     {
00265         uint32_t waiting = 0;
00266         while (ha_flag_vars_ss.deliveredtoken != token) {
00267             usleep(GTI_TOKEN_DELAY);
00268             if (waiting++ > MQTT_TIMEOUT) {
00269                 fprintf(fout, "%s %s GTI Time Still Waiting, timeout %d\r\n", log_time(false),
      topic_p, waiting);
00270                 break;
00271             }
00272         };
00273     }
00274     usleep(HA_SW_DELAY);
00275     return ret;
00276 }
```

4.6.2.7 print_im_vars()

```
void print_im_vars (
            void )
```

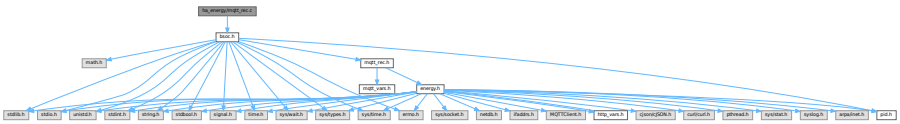
```
00211 {
00212     static char time_log[RBUF_SIZ] = {0};
00213     static uint32_t sync_time = TIME_SYNC_SEC - 1;
00214     time_t rawtime_log;
00215     char imvars[SYSLOG_SIZ];
00216
00217     fflush(fout);
00218     snprintf(imvars, SYSLOG_SIZ - 1, "House L1 %7.2fW, House L2 %7.2fW, GTI L1 %7.2fW, Server %7.2fW",
00219         E.print_vars[L1_P], E.print_vars[L2_P], E.print_vars[L3_P], E.print_vars[L4_P]);
00220     fprintf(fout, "%s", imvars);
00221     fflush(fout);
00222     time(&rawtime_log);
00223     if (sync_time++ > TIME_SYNC_SEC) {
00224         sync_time = 0;
00225         snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
      time commands
00226         mqtt_gti_time(E.client_p, TOPIC_P, time_log);
00227     }
00228 }
```

4.7 http_vars.h

```
00001 /*
00002  * File:   http_vars.h
00003  * Author: root
00004  *
00005  * Created on February 16, 2024, 8:37 AM
00006  */
00007
00008 #ifndef HTTP_VARS_H
00009 #define HTTP_VARS_H
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014
00015 #include "energy.h"
00016
00017 #define IAMM1 "http://10.1.1.101/monitorjson"
00018 #define IAMM2 "http://10.1.1.102/monitorjson"
00019
00020 //#define IM_DEBUG2
00021     extern FILE* fout;
00022
00023     size_t iammeter_write_callback1(char *, size_t, size_t, void *);
00024     size_t iammeter_write_callback2(char *, size_t, size_t, void *);
00025     void iammeter_read1(const char *);
00026     void iammeter_read2(const char *);
00027     void print_im_vars(void);
00028     void print_mvar_vars(void);
00029
00030 #ifdef __cplusplus
00031 }
00032 #endif
00033
00034 #endif /* HTTP_VARS_H */
00035
```

4.8 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)

4.8.1 Detailed Description

data received on topic from the broker, run processing thread max message length set by MBMQTT

4.8.2 Function Documentation

4.8.2.1 delivered()

```
void delivered (
            void * context,
            MQTTClient_deliveryToken dt)
```

```
00123 {
00124     struct ha_flag_type *ha_flag = context;
00125
00126     // bug-out if no context variables passed to callback
00127     if (context == NULL) {
00128         return;
00129     }
00130     ha_flag->deliveredtoken = dt;
00131 }
```

4.8.2.2 fm80_float()

```
bool fm80_float (
            const bool set_bias)
```

```
00247 {
00248     if ((uint32_t) E.mvar[V_FCCM] == FLOAT_CODE) {
00249         if (set_bias) {
00250             E.mode.pv_bias = PV_BIAS_FLOAT;
00251         }
00252         if (E.mode.R != R_IDLE) {
00253             E.mode.R = R_FLOAT;
00254         }
00255         return true;
00256     } else {
00257         if (E.mode.R == R_FLOAT) {
00258             E.mode.R = R_RUN;
00259         }
00260     }
00261     return false;
00262 }
```

4.8.2.3 fm80_sleep()

```
bool fm80_sleep (
            void )
```

```
00269 {
00270     if ((uint32_t) E.mvar[V_FCCM] == SLEEP_CODE) {
00271         return true;
00272     }
00273     return false;
00274 }
```

4.8.2.4 json_get_data()

```
bool json_get_data (
            cJSON * json_src,
            const char * data_id,
            cJSON * name,
            uint32_t i)
```

```
00138 {
00139     bool ret = false;
00140     static uint32_t j = 0;
00141
00142     // access the JSON data using the lookup string passed in data_id
00143     name = cJSON_GetObjectItemCaseSensitive(json_src, data_id);
00144
00145     /*
```

```
00146      * process string values
00147      */
00148     if (cJSON_IsString(name) && (name->valuestring != NULL)) {
00149 #ifdef GET_DEBUG
00150         fprintf(fout, "%s Name: %s\n", data_id, name->valuestring);
00151 #endif
00152         ret = true;
00153     }
00154
00155     /*
00156      * process numeric values
00157      */
00158     if (cJSON_IsNumber(name)) {
00159 #ifdef GET_DEBUG
00160         fprintf(fout, "%s Value: %f\n", data_id, name->valuedouble);
00161 #endif
00162         if (i > V_DLAST) { // check for out-of-range index
00163             i = V_DLAST;
00164         }
00165
00166         // lock the main value array during updates
00167         pthread_mutex_lock(&E.ha_lock);
00168         E.mvar[i] = name->valuedouble;
00169         pthread_mutex_unlock(&E.ha_lock);
00170
00171         /*
00172          * special processing for variable data received
00173          */
00174         if (i == V_DCMPPT) {
00175             /*
00176              * load battery current standard deviation array bat_c_std_dev with data
00177              */
00178             bsoc_set_std_dev(E.mvar[i], j++);
00179             if (j >= RDEV_SIZE) {
00180                 j = 0;
00181             }
00182         }
00183         /*
00184          * update local MATTER switch status from HA
00185          */
00186         if (i == V_HDCSW) {
00187             E.gti_sw_status = (bool) ((int32_t) E.mvar[i]);
00188             E.dc_mismatch = false;
00189         }
00190
00191         if (i == V_HACSW) {
00192             E.ac_sw_status = (bool) ((int32_t) E.mvar[i]);
00193             E.ac_mismatch = false;
00194         }
00195
00196         // command HA_ENERGY to shutdown mode
00197         if (i == V_HSHUT) {
00198             E.solar_shutdown = (bool) ((int32_t) E.mvar[i]);
00199         }
00200         // set HA_ENERGY energy processing mode
00201         if (i == V_HMODE) {
00202             ha_flag_vars_ss.energy_mode = (bool) ((int32_t) E.mvar[i]);
00203         }
00204         if (i == V_HCON0) {
00205             E.mode.con0 = (bool) ((int32_t) E.mvar[i]);
00206         }
00207         if (i == V_HCON1) {
00208             E.mode.con1 = (bool) ((int32_t) E.mvar[i]);
00209         }
00210         if (i == V_HCON2) {
00211             E.mode.con2 = (bool) ((int32_t) E.mvar[i]);
00212         }
00213         if (i == V_HCON3) {
00214             E.mode.con3 = (bool) ((int32_t) E.mvar[i]);
00215         }
00216         if (i == V_HCON4) { // set DL GTI excess load MODE
00217             E.mode.con4 = (bool) ((int32_t) E.mvar[i]);
00218         }
00219         if (i == V_HCON5) { // clear DL GTI excess load MODE
00220             E.mode.con5 = (bool) ((int32_t) E.mvar[i]);
00221         }
00222         if (i == V_HCON6) { // HA Energy program idle
00223             E.mode.con6 = (bool) ((int32_t) E.mvar[i]);
00224         }
00225         if (i == V_HCON7) { // HA Energy program exit
00226             E.mode.con7 = (bool) ((int32_t) E.mvar[i]);
00227         }
00228         ret = true;
00229     }
00230     return ret;
00231 }
```

4.8.2.5 msgarrvd()

```
int32_t msgarrvd (
            void * context,
            char * topicName,
            int topicLen,
            MQTTClient_message * message)
```

```
00008 {
00009     int32_t i, ret = 1;
00010     const char* payloadptr;
00011     char buffer[MBMQTT];
00012     struct ha_flag_type *ha_flag = context;
00013 
00014     E.link.mqtt_count++;
00015     // bug-out if no context variables passed to callback
00016     if (context == NULL) {
00017         ret = -1;
00018         goto null_exit;
00019     }
00020 
00021 #ifdef DEBUG_REC
00022     fprintf(fout, "Message arrived\n");
00023 #endif
00024     /*
00025      * move the received message into a processing holding buffer
00026      */
00027     payloadptr = message->payload;
00028     for (i = 0; i < message->payloadlen; i++) {
00029         buffer[i] = *payloadptr++;
00030     }
00031     buffer[i] = 0; // make a null terminated C string
00032 
00033     // parse the JSON data in the holding buffer
00034     cJSON *json = cJSON_ParseWithLength(buffer, message->payloadlen);
00035     if (json == NULL) {
00036         const char *error_ptr = cJSON_GetErrorPtr();
00037         if (error_ptr != NULL) {
00038             fprintf(fout, "%s Error: %s NULL cJSON pointer\n", log_time(false), error_ptr);
00039         }
00040         ret = -1;
00041         ha_flag->rec_ok = false;
00042         E.fm80 = false;
00043         E.dumpload = false;
00044         E.homeassistant = false;
00045         E.link.mqtt_error++;
00046         goto error_exit;
00047     }
00048 
00049     /*
00050      * MQTT messages from the FM80 Q84 interface
00051      */
00052     if (ha_flag->ha_id == FM80_ID) {
00053 #ifdef DEBUG_REC
00054         fprintf(fout, "FM80 MQTT data\r\n");
00055 #endif
00056         cJSON *data_result = json;
00057 
00058         for (uint32_t ii = V_FCCM; ii < V_FLAST; ii++) {
00059             if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
00060                 ha_flag->var_update++;
00061             }
00062         }
00063         E.fm80 = true;
00064     }
00065 
00066     /*
00067      * MQTT messages from the K42 dumpload/gti interface
00068      */
00069     if (ha_flag->ha_id == DUMPLOAD_ID) {
00070 #ifdef DEBUG_REC
00071         fprintf(fout, "DUMPLOAD MQTT data\r\n");
00072 #endif
00073         cJSON *data_result = json;
00074 
00075         for (uint32_t ii = V_HDCSW; ii < V_DLAST; ii++) {
00076             if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
00077                 ha_flag->var_update++;
00078             }
00079         }
00080         E.dumpload = true;
00081     }
00082 
00083     /*
```

```
00084      * MQTT messages from the Linux HA_ENERGY interface
00085      */
00086     if (ha_flag->ha_id == HA_ID) {
00087 #ifdef DEBUG_REC
00088         fprintf(fout, "Home Assistant MQTT data\r\n");
00089 #endif
00090         cJSON *data_result = json;
00091 
00092         if (json_get_data(json, mqtt_name[V_HACSW], data_result, V_HACSW)) {
00093             ha_flag->var_update++;
00094         }
00095         data_result = json;
00096         if (json_get_data(json, mqtt_name[V_HDCSW], data_result, V_HDCSW)) {
00097             ha_flag->var_update++;
00098         }
00099 
00100         E.homeassistant = true;
00101     }
00102 
00103     // done with processing MQTT async message, set state flags
00104     ha_flag->receivedtoken = true;
00105     ha_flag->rec_ok = true;
00106     /*
00107      * exit and delete/free resources. In steps depending of possible error conditions
00108      */
00109 error_exit:
00110     // delete the JSON object
00111     cJSON_Delete(json);
00112 null_exit:
00113     // free the MQTT objects
00114     MQTTClient_freeMessage(&message);
00115     MQTTClient_free(topicName);
00116     return ret;
00117 }
```

4.8.2.6 print_mvar_vars()

```
void print_mvar_vars (
             void )
```

```
00237 {
00238     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",
00239         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],
00240         (int32_t) E.dc_mismatch, (int32_t) E.ac_mismatch, (int32_t) E.mode_mismatch);
00241 }
```

4.9 mqtt_rec.h

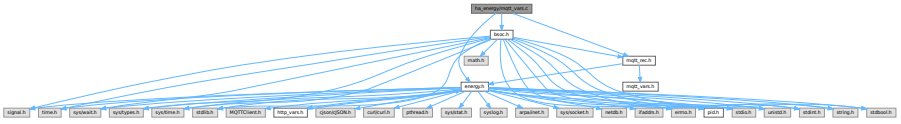
```
00001 /*
00002  * File:   mqtt_rec.h
00003  * Author: root
00004  *
00005  * Created on February 5, 2024, 2:54 PM
00006  */
00007 
00008 #ifndef MQTT_REC_H
00009 #define MQTT_REC_H
00010 
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014 
00015 #include "energy.h"
00016 #include "mqtt_vars.h"
00017 
00018 #define RDEV_SIZE        10
00019 
00020 #define SLEEP_CODE      0
00021 #define FLOAT_CODE      1
00022     //#define DEBUG_REC
00023     //#define GET_DEBUG
00024 
00025 #define MBMQTT  1024
00026 
00027     enum mqtt_id {
```

```
00028         P8055_ID,
00029         FM80_ID,
00030         DUMPLOAD_ID,
00031         HA_ID,
00032         LAST_MQTT_ID,
00033     };
00034
00035     struct ha_flag_type {
00036         volatile MQTTClient_deliveryToken deliveredtoken, receivedtoken;
00037         volatile bool runner, rec_ok;
00038         int32_t ha_id;
00039         volatile int32_t var_update, energy_mode;
00040         volatile enum client_id cid;
00041     };
00042
00043     extern FILE* fout;
00044
00045     int32_t msgarrvd(void *, char *, int, MQTTClient_message *);
00046     void delivered(void *, MQTTClient_deliveryToken);
00047
00048     bool json_get_data(cJSON *, const char *, cJSON *, uint32_t);
00049     bool fm80_float(const bool set_bias);
00050     bool fm80_sleep(void);
00051
00052 #ifdef __cplusplus
00053 }
00054 #endif
00055
00056 #endif /* MQTT_REC_H */
00057
```

4.10 ha_energy/mqtt_vars.c File Reference

```
#include "mqtt_rec.h"
#include "energy.h"
#include "bsoc.h"
```

Include dependency graph for mqtt_vars.c:



Functions

- void mqtt_ha_shutdown (MQTTClient client_p, const char *topic_p)
- void mqtt_ha_pid (MQTTClient client_p, const char *topic_p)
- void mqtt_ha_switch (MQTTClient client_p, const char *topic_p, const bool sw_state)
- bool mqtt_gti_power (MQTTClient client_p, const char *topic_p, char *msg, uint32_t trace)
- bool mqtt_gti_time (MQTTClient client_p, const char *topic_p, char *msg)

Variables

- static const char *const **FW_Date** = __DATE__
- static const char *const **FW_Time** = __TIME__

4.10.1 Detailed Description

send energy shutdown command to Home Assistant

4.10.2 Function Documentation

4.10.2.1 mqtt_gti_power()

```
bool mqtt_gti_power (
            MQTTClient client_p,
            const char * topic_p,
            char * msg,
            uint32_t trace)
```

```
00187 {
00188     bool ret = true;
00189     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00190     MQTTClient_deliveryToken token;
00191     ha_flag_vars_ss.deliveredtoken = 0;
00192     static bool spam = false;
00193 
00194     E.link.mqtt_count++;
00195     pubmsg.payload = msg;
00196     pubmsg.payloadlen = strlen(msg);
00197     pubmsg.qos = QOS;
00198     pubmsg.retained = 0;
00199 
00200     if (E.dl_excess) { // always run excess commands
00201         spam = false;
00202     }
00203 #ifdef GTI_NO_POWER
00204     MQTTClient_publishMessage(client_p, "mateq84/data/gticmd_nopower", &pubmsg, &token);
00205 #else
00206     if (bsoc_gti() > MIN_BAT_KW || E.dl_excess) {
00207 #ifdef DEBUG_HA_CMD
00208         log_time(true);
00209         fprintf(fout, "HA GTI power command %s, SDEV %5.2f trace %u\r\n", msg, get_batc_dev(), trace);
00210         fflush(fout);
00211         spam = true;
00212 #endif
00213         MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run power commands
00214     } else {
00215         ret = false;
00216         pubmsg.payload = DL_POWER_ZERO;
00217         pubmsg.payloadlen = strlen(DL_POWER_ZERO);
00218         if (!spam) {
00219             MQTTClient_publishMessage(client_p, TOPIC_SPAM, &pubmsg, &token);
00220         } else {
00221             MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // only shutdown GTI power
00222         }
00223 #ifdef DEBUG_HA_CMD
00224         if (spam) {
00225             log_time(true);
00226             fprintf(fout, "HA GTI power set to zero, trace %u\r\n", trace);
00227             fflush(fout);
00228             spam = false;
00229         }
00230 #endif
00231     }
00232 #endif
00233     // a busy, wait loop for the async delivery thread to complete
00234     {
00235         uint32_t waiting = 0;
00236         while (ha_flag_vars_ss.deliveredtoken != token) {
00237             usleep(TOKEN_DELAY);
00238             if (waiting++ > MQTT_TIMEOUT) {
00239                 fprintf(fout, "%s %s GTI Power Still Waiting, timeout %d\r\n", log_time(false),
    topic_p, waiting);
00240                 break;
00241             }
00242         };
00243     }
00244     usleep(HA_SW_DELAY);
00245     return ret;
00246 }
```

4.10.2.2 mqtt_gti_time()

```
bool mqtt_gti_time (
            MQTTClient client_p,
```

```
            const char * topic_p,
            char * msg)
00249 {
00250     bool ret = true;
00251     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00252     MQTTClient_deliveryToken token;
00253     ha_flag_vars_ss.deliveredtoken = 0;
00254
00255     E.link.mqtt_count++;
00256     pubmsg.payload = msg;
00257     pubmsg.payloadlen = strlen(msg);
00258     pubmsg.qos = QOS;
00259     pubmsg.retained = 0;
00260
00261     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run time commands
00262
00263     // a busy, wait loop for the async delivery thread to complete
00264     {
00265         uint32_t waiting = 0;
00266         while (ha_flag_vars_ss.deliveredtoken != token) {
00267             usleep(GTI_TOKEN_DELAY);
00268             if (waiting++ > MQTT_TIMEOUT) {
00269                 fprintf(fout, "%s %s GTI Time Still Waiting, timeout %d\r\n", log_time(false),
    topic_p, waiting);
00270                 break;
00271             }
00272         };
00273     }
00274     usleep(HA_SW_DELAY);
00275     return ret;
00276 }
```

4.10.2.3 mqtt_ha_pid()

```
void mqtt_ha_pid (
            MQTTClient client_p,
            const char * topic_p)
00046 {
00047     cJSON *json;
00048     time_t rawtime;
00049
00050     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00051     MQTTClient_deliveryToken token;
00052     ha_flag_vars_ss.deliveredtoken = 0;
00053
00054     E.link.mqtt_count++;
00055     E.mode.sequence++;
00056     json = cJSON_CreateObject();
00057     cJSON_AddStringToObject(json, "name", CLIENTID1);
00058     cJSON_AddNumberToObject(json, "sequence", E.mode.sequence);
00059     cJSON_AddNumberToObject(json, "mqtt_count", (double) E.link.mqtt_count);
00060     cJSON_AddNumberToObject(json, "http_count", (double) E.link.iammeter_count);
00061     cJSON_AddNumberToObject(json, "piderror", E.mode.error);
00062     cJSON_AddNumberToObject(json, "totalsystem", E.mode.total_system);
00063     cJSON_AddNumberToObject(json, "gtinet", E.mode.gti_dumpload);
00064     cJSON_AddNumberToObject(json, "energy_state", (double) E.mode.E);
00065     cJSON_AddNumberToObject(json, "run_state", (double) E.mode.R);
00066     // correct for power sensed by GTI metering
00067     E.mode.off_grid = (E.mvar[V_FLO] - (E.mvar[V_DPPV] * DL_AC_DC_EFF));
00068     E.mode.off_grid = drive1_filter(E.mode.off_grid);
00069     if (E.mode.off_grid < 0.0f) { // only see power removed from grid usage
00070         E.mode.off_grid = 0.0f;
00071     }
00072     cJSON_AddNumberToObject(json, "off_grid", E.mode.off_grid);
00073     cJSON_AddNumberToObject(json, "excess_mode", (double) E.dl_excess);
00074     cJSON_AddStringToObject(json, "build_date", FW_Date);
00075     cJSON_AddStringToObject(json, "build_time", FW_Time);
00076     time(&rawtime);
00077     cJSON_AddNumberToObject(json, "sequence_time", (double) rawtime);
00078     // convert the cJSON object to a JSON string
00079     char *json_str = cJSON_Print(json);
00080
00081     pubmsg.payload = json_str;
00082     pubmsg.payloadlen = strlen(json_str);
00083     pubmsg.qos = QOS;
00084     pubmsg.retained = 0;
00085
00086     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
00087     // a busy, wait loop for the async delivery thread to complete
00088     {
```

```
00089         uint32_t waiting = 0;
00090         while (ha_flag_vars_ss.deliveredtoken != token) {
00091             usleep(TOKEN_DELAY);
00092             if (waiting++ > MQTT_TIMEOUT) {
00093                 fprintf(fout, "%s %s SW Still Waiting, timeout %d\r\n", log_time(false), topic_p,
      waiting);
00094                 break;
00095             }
00096         };
00097     }
00098
00099     cJSON_free(json_str);
00100     cJSON_Delete(json);
00101 }
```

4.10.2.4 mqtt_ha_shutdown()

```
void mqtt_ha_shutdown (
             MQTTClient client_p,
             const char * topic_p)
```

```
00013 {
00014     cJSON *json;
00015     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00016     MQTTClient_deliveryToken token;
00017     ha_flag_vars_ss.deliveredtoken = 0;
00018
00019     json = cJSON_CreateObject();
00020     cJSON_AddStringToObject(json, "shutdown", CLIENTID1);
00021     char *json_str = cJSON_Print(json);
00022
00023     pubmsg.payload = json_str;
00024     pubmsg.payloadlen = strlen(json_str);
00025     pubmsg.qos = QOS;
00026     pubmsg.retained = 0;
00027
00028     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
00029     // a busy, wait loop for the async delivery thread to complete
00030     {
00031         uint32_t waiting = 0;
00032         while (ha_flag_vars_ss.deliveredtoken != token) {
00033             usleep(TOKEN_DELAY);
00034             if (waiting++ > MQTT_TIMEOUT) {
00035                 fprintf(fout, "%s %s SW Still Waiting, timeout %d\r\n", log_time(false), topic_p,
      waiting);
00036                 break;
00037             }
00038         };
00039     }
00040 }
```

4.10.2.5 mqtt_ha_switch()

```
void mqtt_ha_switch (
             MQTTClient client_p,
             const char * topic_p,
             const bool sw_state)
```

```
00107 {
00108     cJSON *json;
00109 #ifdef DEBUG_HA_CMD
00110     static bool spam = false;
00111     static uint32_t less_spam = 0;
00112 #endif
00113
00114     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00115     MQTTClient_deliveryToken token;
00116     ha_flag_vars_ss.deliveredtoken = 0;
00117
00118     E.link.mqtt_count++;
00119     json = cJSON_CreateObject();
00120     if (sw_state) {
00121         cJSON_AddStringToObject(json, "state", "ON");
00122 #ifdef DEBUG_HA_CMD
00123         spam = true;
00124         less_spam = 0;
```

```
00125 #endif
00126     } else {
00127         if ((uint32_t) E.mvar[V_FCCM] != FLOAT_CODE) { // use max power in FLOAT mode
00128             cJSON_AddStringToObject(json, "state", "OFF");
00129         } else {
00130             cJSON_AddStringToObject(json, "state", "ON");
00131 #ifdef DEBUG_HA_CMD
00132             spam = true;
00133             less_spam = 0;
00134 #endif
00135         }
00136     }
00137     // convert the cJSON object to a JSON string
00138     char *json_str = cJSON_Print(json);
00139 
00140     pubmsg.payload = json_str;
00141     pubmsg.payloadlen = strlen(json_str);
00142     pubmsg.qos = QOS;
00143     pubmsg.retained = 0;
00144 
00145 #ifdef DEBUG_HA_CMD
00146     if (spam) {
00147         log_time(true);
00148         fflush(fout);
00149         fprintf(fout, "HA switch command %s, %d\r\n", topic_p, sw_state);
00150         fflush(fout);
00151         if (!sw_state) {
00152             if (less_spam++ > 3) {
00153                 spam = false;
00154                 less_spam = 0;
00155             }
00156         }
00157     }
00158 #endif
00159 
00160     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
00161     // a busy, wait loop for the async delivery thread to complete
00162     {
00163         uint32_t waiting = 0;
00164         while (ha_flag_vars_ss.deliveredtoken != token) {
00165             usleep(TOKEN_DELAY);
00166             if (waiting++ > MQTT_TIMEOUT) {
00167 #ifdef DEBUG_HA_CMD
00168                 fflush(fout);
00169                 fprintf(fout, "\r\nSW Still Waiting, timeout\r\n");
00170                 fflush(fout);
00171 #endif
00172                 break;
00173             }
00174         };
00175     }
00176 
00177     cJSON_free(json_str);
00178     cJSON_Delete(json);
00179     usleep(HA_SW_DELAY);
00180     fflush(fout);
00181 }
```

4.11 mqtt_vars.h

```
00001 /*
00002  * File:   mqtt_vars.h
00003  * Author: root
00004  *
00005  * Created on February 9, 2024, 6:50 AM
00006  */
00007 
00008 #ifndef MQTT_VARS_H
00009 #define MQTT_VARS_H
00010 
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014 
00015     //#define GTI_NO_POWER      // do we actually run power commands
00016 
00017     //#define DEBUG_HA_CMD    // show debug text
00018 #define HA_SW_DELAY     400000  // usecs
00019 #define TOKEN_DELAY     500
00020 #define GTI_TOKEN_DELAY 600
00021 
00022 #define QOS             1
00023 
```

```
00024     extern const char* mqtt_name[V_DLAST];
00025
00026     void mqtt_ha_switch(MQTTClient, const char *, const bool);
00027     void mqtt_ha_pid(MQTTClient, const char *);
00028     void mqtt_ha_shutdown(MQTTClient, const char *);
00029     bool mqtt_gti_power(MQTTClient, const char *, char *, uint32_t);
00030     bool mqtt_gti_time(MQTTClient, const char *, char *);
00031
00032 #ifdef __cplusplus
00033 }
00034 #endif
00035
00036 #endif /* MQTT_VARS_H */
00037
```

4.12 pid.h

```
00001 /*
00002  * File:   pid.h
00003  * Author: root
00004  *
00005  * Created on March 6, 2024, 7:03 AM
00006  */
00007
00008 #ifndef PID_H
00009 #define PID_H
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014
00015     struct SPid {
00016         double dState; // Last position input
00017         double iState; // Integrator state
00018         double iMax, iMin; // Maximum and minimum allowable integrator state
00019         double iGain, // integral gain
00020         pGain, // proportional gain
00021         dGain; // derivative gain
00022     };
00023
00024     double UpdatePI(volatile struct SPid * const, const double);
00025     void ResetPI(volatile struct SPid * const);
00026
00027
00028 #ifdef __cplusplus
00029 }
00030 #endif
00031
00032 #endif /* PID_H */
00033
```

Index