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

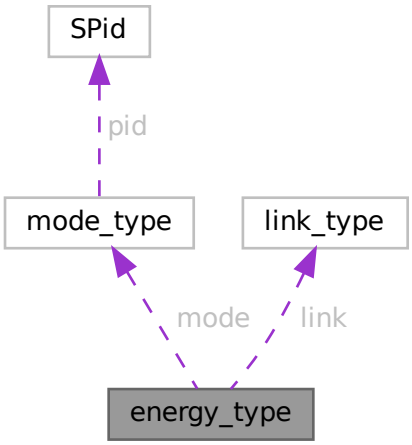
Here is a list of all documented files with brief descriptions:

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 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

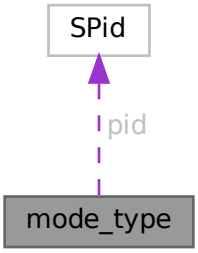
- volatile double **ac_weight**
- volatile double **gti_weight**
- volatile double **pv_voltage**
- volatile double **bat_current**
- volatile double **batc_std_dev**
- volatile double **bat_voltage**
- double **bat_c_std_dev** [DEV_SIZE]
- double **coef**

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

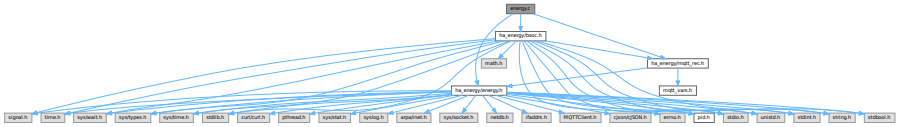
Chapter 4

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

4.1.2 Function Documentation

4.1.2.1 connlost()

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

trouble in River-city

```
00301 {
00302     struct ha_flag_type *ha_flag = context;
00303     int32_t id_num = ha_flag->ha_id;
00304     static uint32_t times = 0;
00305     char * where = "Missing Topic";
00306 
00307     switch (ha_flag->cid) {
00308     case ID_C1:
00309         where = TOPIC_SS;
00310         break;
00311     case ID_C2:
00312         where = TOPIC_SD;
00313         break;
00314     case ID_C3:
00315         where = TOPIC_HA;
00316         break;
```

```
00317     }
00318
00319     // bug-out if no context variables passed to callback
00320     if (context == NULL) {
00321         id_num = -1;
00322         goto bugout;
00323     } else {
00324         if (times++ > MQTT_RECONN) {
00325             goto bugout;
00326         } else {
00327
00328             fprintf(fout, "\n%s Connection lost, exit ha_energy program\n", log_time(false));
00329             fprintf(fout, "%s     cause: %s, h_id %d c_id %d %s \n", log_time(false), cause, id_num,
      ha_flag->cid, where);
00330             fprintf(fout, "%s MQTT DAEMON failure  LOG Version %s : MQTT Version %s\n",
      log_time(false), LOG_VERSION, MQTT_VERSION);
00331             fflush(fout);
00332         }
00333     }
00334
00335 bugout:
00336     fprintf(fout, "\n%s Connection lost, exit ha_energy program\n", log_time(false));
00337     fprintf(fout, "%s     cause: %s, h_id %d c_id %d %s \n", log_time(false), cause, id_num,
      ha_flag->cid, where);
00338     fprintf(fout, "%s MQTT DAEMON failure  LOG Version %s : MQTT Version %s\n", log_time(false),
      LOG_VERSION, MQTT_VERSION);
00339     fflush(fout);
00340     exit(EXIT_FAILURE);
00341 }
```

4.1.2.2 ha_ac_off()

```
void ha_ac_off (
              void )
00978 {
00979     mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
00980     E.ac_sw_status = false;
00981 }
```

4.1.2.3 ha_ac_on()

```
void ha_ac_on (
              void )
00984 {
00985     mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00986     E.ac_sw_status = true;
00987 }
```

4.1.2.4 ha_dc_off()

```
void ha_dc_off (
              void )
00993 {
00994     mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
00995     E.gti_sw_status = false;
00996 }
```

4.1.2.5 ha_dc_on()

```
void ha_dc_on (
              void )
00999 {
01000     mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
01001     E.gti_sw_status = true;
01002 }
```

4.1.2.6 log_time()

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

```
01063 {
01064     static char time_log[RBUF_SIZ] = {0};
01065     static uint32_t len = 0, sync_time = TIME_SYNC_SEC - 1;
01066     time_t rawtime_log;
01067
01068     tzset();
01069     timezone = 0;
01070     daylight = 0;
01071     time(&rawtime_log);
01072     if (sync_time++ > TIME_SYNC_SEC) {
01073         sync_time = 0;
01074         snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
      time commands
01075         mqtt_gti_time(E.client_p, TOPIC_P, time_log);
01076     }
01077
01078     sprintf(time_log, "%s", ctime(&rawtime_log));
01079     len = strlen(time_log);
01080     time_log[len - 1] = 0; // munge out the return character
01081     if (log) {
01082         fprintf(fout, "%s ", time_log);
01083         fflush(fout);
01084     }
01085
01086     return time_log;
01087 }
```

4.1.2.7 log_timer()

```
bool log_timer (
            void )
```

```
01122 {
01123     bool itstime = false;
01124
01125     if (E.log_spam < LOW_LOG_SPAM) {
01126         E.log_time_reset = 0;
01127         itstime = true;
01128     }
01129     if (E.log_time_reset > RESET_LOG_SPAM) {
01130         E.log_spam = 0;
01131         itstime = true;
01132     }
01133     return itstime;
01134 }
```

4.1.2.8 main()

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

```
00350 {
00351     struct itimerval new_timer = {
00352         .it_value.tv_sec = CMD_SEC,
00353         .it_value.tv_usec = 0,
00354         .it_interval.tv_sec = CMD_SEC,
00355         .it_interval.tv_usec = 0,
00356     };
00357     struct itimerval old_timer;
00358     time_t rawtime;
00359     MQTTClient_connectOptions conn_opts_p = MQTTClient_connectOptions_initializer,
00360         conn_opts_sd = MQTTClient_connectOptions_initializer,
00361         conn_opts_ha = MQTTClient_connectOptions_initializer;
00362     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00363     MQTTClient_deliveryToken token;
00364     char hname[256], *hname_ptr = hname;
00365     size_t hname_len = 12;
00366
00367     gethostname(hname, hname_len);
00368     hname[12] = 0;
```

```
00369     printf("\r\n  LOG Version %s : MQTT Version %s : Host Name %s\r\n", LOG_VERSION, MQTT_VERSION,
      hname);
00370     showIP();
00371     skeleton_daemon();
00372 
00373     while (true) {
00374         switch (E.mode.E) {
00375         case E_INIT:
00376 
00377 #ifdef LOG_TO_FILE
00378             fout = fopen(LOG_TO_FILE, "a");
00379             if (fout == NULL) {
00380                 fout = fopen(LOG_TO_FILE_ALT, "a");
00381                 if (fout == NULL) {
00382                     fout = stdout;
00383                     printf("\r\n%s Unable to open LOG file %s \r\n", log_time(false),
      LOG_TO_FILE_ALT);
00384                 }
00385             }
00386 #else
00387             fout = stdout;
00388 #endif
00389             fprintf(fout, "\r\n%s LOG Version %s : MQTT Version %s\r\n", log_time(false), LOG_VERSION,
      MQTT_VERSION);
00390             fflush(fout);
00391 
00392             if (!bsoc_init()) {
00393                 fprintf(fout, "\r\n%s bsoc_init failure \r\n", log_time(false));
00394                 fflush(fout);
00395                 exit(EXIT_FAILURE);
00396             }
00397             /*
00398              * set the timer for MQTT publishing sample speed
00399              * CMD_SEC         10
00400              */
00401             setitimer(ITIMER_REAL, &new_timer, &old_timer);
00402             signal(SIGALRM, timer_callback);
00403 
00404             if (strncmp(hname, TNAME, 6) == 0) {
00405                 MQTTClient_create(&E.client_p, LADDRESS, CLIENTID1,
00406                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00407                 conn_opts_p.keepAliveInterval = KAI;
00408                 conn_opts_p.cleansession = 1;
00409                 hname_ptr = LADDRESS;
00410             } else {
00411                 MQTTClient_create(&E.client_p, ADDRESS, CLIENTID1,
00412                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00413                 conn_opts_p.keepAliveInterval = KAI;
00414                 conn_opts_p.cleansession = 1;
00415                 hname_ptr = ADDRESS;
00416             }
00417 
00418             fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID1);
00419             fflush(fout);
00420             ha_flag_vars_ss.cid = ID_C1;
00421             MQTTClient_setCallbacks(E.client_p, &ha_flag_vars_ss, connlost, msgarrvd, delivered);
00422             if ((E.rc = MQTTClient_connect(E.client_p, &conn_opts_p)) != MQTTCLIENT_SUCCESS) {
00423                 fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
      log_time(false), E.rc, hname_ptr, CLIENTID1);
00424                 fflush(fout);
00425                 pthread_mutex_destroy(&E.ha_lock);
00426                 exit(EXIT_FAILURE);
00427             }
00428 
00429             if (strncmp(hname, TNAME, 6) == 0) {
00430                 MQTTClient_create(&E.client_sd, LADDRESS, CLIENTID2,
00431                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00432                 conn_opts_sd.keepAliveInterval = KAI;
00433                 conn_opts_sd.cleansession = 1;
00434                 hname_ptr = LADDRESS;
00435             } else {
00436                 MQTTClient_create(&E.client_sd, ADDRESS, CLIENTID2,
00437                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00438                 conn_opts_sd.keepAliveInterval = KAI;
00439                 conn_opts_sd.cleansession = 1;
00440                 hname_ptr = ADDRESS;
00441             }
00442 
00443             fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID2);
00444             fflush(fout);
00445             ha_flag_vars_sd.cid = ID_C2;
00446             MQTTClient_setCallbacks(E.client_sd, &ha_flag_vars_sd, connlost, msgarrvd, delivered);
00447             if ((E.rc = MQTTClient_connect(E.client_sd, &conn_opts_sd)) != MQTTCLIENT_SUCCESS) {
00448                 fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
      log_time(false), E.rc, hname_ptr, CLIENTID2);
00449                 fflush(fout);
00450                 pthread_mutex_destroy(&E.ha_lock);
```

```
00451                 exit(EXIT_FAILURE);
00452             }
00453 
00454             /*
00455              * Home Assistant MQTT receive messages
00456              */
00457             if (strncmp(hname, TNAME, 6) == 0) {
00458                 MQTTClient_create(&E.client_ha, LADDRESS, CLIENTID3,
00459                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00460                 conn_opts_ha.keepAliveInterval = KAI;
00461                 conn_opts_ha.cleansession = 1;
00462                 hname_ptr = LADDRESS;
00463             } else {
00464                 MQTTClient_create(&E.client_ha, ADDRESS, CLIENTID3,
00465                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00466                 conn_opts_ha.keepAliveInterval = KAI;
00467                 conn_opts_ha.cleansession = 1;
00468                 hname_ptr = ADDRESS;
00469             }
00470 
00471             fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID3);
00472             fflush(fout);
00473             ha_flag_vars_ha.cid = ID_C3;
00474             MQTTClient_setCallbacks(E.client_ha, &ha_flag_vars_ha, connlost, msgarrvd, delivered);
00475             if ((E.rc = MQTTClient_connect(E.client_ha, &conn_opts_ha)) != MQTTCLIENT_SUCCESS) {
00476                 fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
      log_time(false), E.rc, hname_ptr, CLIENTID3);
00477                 fflush(fout);
00478                 pthread_mutex_destroy(&E.ha_lock);
00479                 exit(EXIT_FAILURE);
00480             }
00481 
00482             /*
00483              * on topic received data will trigger the msgarrvd function
00484              */
00485             MQTTClient_subscribe(E.client_p, TOPIC_SS, QOS); // FM80 Q84
00486             MQTTClient_subscribe(E.client_sd, TOPIC_SD, QOS); // DUMPLOAD K42
00487             MQTTClient_subscribe(E.client_ha, TOPIC_HA, QOS); // Home Assistant Linux AMD64  and ARM64
00488 
00489             pubmsg.payload = "online";
00490             pubmsg.payloadlen = strlen("online");
00491             pubmsg.qos = QOS;
00492             pubmsg.retained = 0;
00493             ha_flag_vars_ss.deliveredtoken = 0;
00494             // notify HA we are running and controlling AC power plugs
00495             MQTTClient_publishMessage(E.client_p, TOPIC_PACA, &pubmsg, &token);
00496             MQTTClient_publishMessage(E.client_p, TOPIC_PDCA, &pubmsg, &token);
00497 
00498             // sync HA power switches
00499             mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
00500             mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
00501             mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00502             mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00503             mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
00504             mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
00505 
00506             E.ac_sw_on = true; // can be switched on once
00507             E.gti_sw_on = true; // can be switched on once
00508 
00509             /*
00510              * use libcurl to read AC power meter HTTP data
00511              * iammeter connected for split single phase monitoring and one leg GTI power exporting
00512              */
00513             iammeter_read();
00514 
00515             /*
00516              * start the main energy monitoring loop
00517              */
00518             fprintf(fout, "\r\n%s Solar Energy AC power controller\r\n", log_time(false));
00519 
00520 #ifdef FAKE_VPV
00521             fprintf(fout, "\r\n Faking dumpload PV voltage\r\n");
00522 #endif
00523             ha_flag_vars_ss.energy_mode = NORM_MODE;
00524             E.mode.E = E_WAIT;
00525             break;
00526         case E_WAIT:
00527             if (ha_flag_vars_ss.runner || E.speed_go++ > 1500000) {
00528                 E.speed_go = 0;
00529                 ha_flag_vars_ss.runner = false;
00530                 E.mode.E = E_RUN;
00531             }
00532 
00533             usleep(100);
00534             /*
00535              * main state-machine update sequence
00536              */
```

```
00537             bsoc_data_collect();
00538             if (!sanity_check()) {
00539                 fprintf(fout, "\r\n%s Sanity Check error %d %s \r\n", log_time(false), E.sane,
      mqtt_name[E.sane]);
00540                 fflush(fout);
00541             }
00542
00543             /*
00544              * stop and restart the energy control processing
00545              * from inside the program or from a remote Home Assistant command
00546              */
00547             if (solar_shutdown()) {
00548                 if (!E.startup) {
00549                     fprintf(fout, "%s SHUTDOWN Solar Energy Control ---> \r\n", log_time(false));
00550                 }
00551                 fflush(fout);
00552                 ramp_down_gti(E.client_p, true);
00553                 usleep(100000); // wait
00554                 ramp_down_ac(E.client_p, true);
00555                 usleep(100000); // wait
00556                 ramp_down_gti(E.client_p, true);
00557                 usleep(100000); // wait
00558                 ramp_down_ac(E.client_p, true);
00559                 usleep(100000); // wait
00560                 if (!E.startup) {
00561                     fprintf(fout, "%s Completed SHUTDOWN, Press again to RESTART.\r\n",
      log_time(false));
00562                     fflush(fout);
00563                 }
00564                 fflush(fout);
00565
00566                 uint8_t iam_delay = 0;
00567                 while (solar_shutdown()) {
00568                     mqtt_ha_shutdown(E.client_p, TOPIC_SHUTDOWN);
00569                     usleep(USEC_SEC); // wait
00570                     if ((int32_t) E.mvar[V_HACSW]) {
00571                         ha_ac_off();
00572                     }
00573                     if ((int32_t) E.mvar[V_HDCSW]) {
00574                         ha_dc_off();
00575                     }
00576                     if ((iam_delay++ > IAM_DELAY) && E.link.shutdown) {
00577                         E.fm80 = true;
00578                         E.dumpload = true;
00579                         E.iammeter = true;
00580                         E.homeassistant = true;
00581                     }
00582                 }
00583                 E.link.shutdown = 0;
00584                 fprintf(fout, "%s RESTART Solar Energy Control\r\n", log_time(false));
00585                 fflush(fout);
00586                 bsoc_set_mode(E.mode.pv_bias, true, true);
00587                 E.dl_excess = true;
00588                 mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 1); // zero power at startup
00589                 E.dl_excess = false;
00590 #ifdef AUTO_CHARGE
00591                 mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00592 #endif
00593                 usleep(100000); // wait
00594                 E.gti_sw_status = true;
00595                 ResetPI(&E.mode.pid);
00596                 ha_flag_vars_ss.runner = true;
00597                 E.fm80 = true;
00598                 E.dumpload = true;
00599                 E.iammeter = true;
00600                 E.homeassistant = true;
00601                 E.mode.in_pid_control = false; // shutdown auto energy control
00602                 E.mode.R = R_INIT;
00603             }
00604             if (ha_flag_vars_ss.receivedtoken) {
00605                 ha_flag_vars_ss.receivedtoken = false;
00606             }
00607             if (ha_flag_vars_sd.receivedtoken) {
00608                 ha_flag_vars_sd.receivedtoken = false;
00609             }
00610             break;
00611         case E_RUN:
00612             usleep(100);
00613             switch (E.mode.R) {
00614             case R_INIT:
00615                 E.once_ac = true;
00616                 E.once_gti = true;
00617                 E.ac_sw_on = true;
00618                 E.gti_sw_on = true;
00619                 E.mode.R = R_RUN;
00620                 E.mode.no_float = true;
00621                 break;
```

```
00622             case R_FLOAT:
00623                 if (E.mode.no_float) {
00624                     E.once_ac = true;
00625                     E.once_gti = true;
00626                     E.ac_sw_on = true;
00627                     E.gti_sw_on = true;
00628                     E.gti_sw_status = false;
00629                     E.ac_sw_status = false;
00630                     E.mode.no_float = false;
00631                 }
00632                 if (!E.gti_sw_status) {
00633                     if (gti_test() > MIN_BAT_KW_GTI_HI) {
00634                         mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00635                         E.gti_sw_status = true;
00636                         fprintf(fout, "%s R_FLOAT DC switch true \r\n", log_time(false));
00637                     }
00638                 }
00639                 usleep(100000); // wait
00640                 if (!E.ac_sw_status) {
00641                     if (ac_test() > MIN_BAT_KW_AC_HI) {
00642                         mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00643                         E.ac_sw_status = true;
00644                         fprintf(fout, "%s R_FLOAT AC switch true \r\n", log_time(false));
00645                     }
00646                 }
00647                 E.mode.pv_bias = PV_BIAS;
00648                 fm80_float(true);
00649                 break;
00650             case R_RUN:
00651             default:
00652                 E.mode.R = R_RUN;
00653                 E.mode.no_float = true;
00654                 break;
00655             }
00656             /*
00657              * main state-machine update sequence and control logic
00658              */
00659             /*
00660              * check for idle/data errors flags from sensors and HA
00661              */
00662             if (!E.mode.data_error) {
00663                 bsoc_set_mode(E.mode.pv_bias, true, false);
00664                 if (E.gti_delay++ >= GTI_DELAY) {
00665                     char gti_str[SBUF_SIZ];
00666                     int32_t error_drive;
00667 
00668                     /*
00669                      * reset the control mode from simple switched power to PID control
00670                      */
00671                     if (!E.mode.in_pid_control) {
00672                         mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00673                         E.gti_sw_status = true;
00674                         usleep(100000); // wait
00675                         mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00676                         E.ac_sw_status = true;
00677                         E.mode.pv_bias = PV_BIAS;
00678                         fprintf(fout, "%s in_pid_mode AC/DC switch true \r\n", log_time(false));
00679                         fm80_float(true);
00680                     } else {
00681                         if (!fm80_float(true)) {
00682                             E.mode.pv_bias = (int32_t) E.mode.error - PV_BIAS;
00683                         }
00684                     }
00685                     /*
00686                      * use PID style set-point error correction
00687                      */
00688                     E.mode.in_pid_control = true;
00689                     E.gti_delay = 0;
00690                     /*
00691                      * adjust power balance if battery charging energy is low
00692                      */
00693                     if (E.mvar[V_DPBAT] > PV_DL_BIAS_RATE) {
00694                         error_drive = (int32_t) E.mode.error - E.mode.pv_bias; // PI feedback control
    signal
00695                     } else {
00696                         error_drive = (int32_t) E.mode.error - PV_BIAS_RATE;
00697                     }
00698                     /*
00699                      * when main battery is in float, crank-up the power draw from the solar panels
00700                      */
00701                     if (fm80_float(true)) {
00702                         error_drive = (int32_t) (E.mode.error + PV_BIAS);
00703                     }
00704                     /*
00705                      * don't drive to zero power
00706                      */
00707                     if (error_drive < 0) {
```

```
00708                         error_drive = PV_BIAS_LOW; // control wide power swings
00709                         if (!fm80_sleep()) { // check for using sleep bias
00710                             if ((E.mvar[V_FBEKW] > MIN_BAT_KW_BSOC_SLP) && (E.mvar[V_PWA] >
      PWA_SLEEP)) {
00711                                 error_drive = PV_BIAS_SLEEP; // use higher power when we still have
      sun for better inverter efficiency
00712                             }
00713                         }
00714                     }
00715
00716                     /*
00717                      * reduce charging/diversion power to safe PS limits
00718                      */
00719                     if (E.mode.dl_mqtt_max > PV_DL_MPTT_MAX) {
00720                         if (!E.dl_excess) {
00721                             error_drive = PV_DL_MPTT_IDLE;
00722                         } else {
00723                             if (E.mode.dl_mqtt_max > PV_DL_MPTT_EXCESS) {
00724                                 error_drive = PV_DL_MPTT_IDLE;
00725                             }
00726                         }
00727                     } else {
00728                         if (E.dl_excess) {
00729                             error_drive = PV_DL_EXCESS + E.dl_excess_adj;
00730                         }
00731                     }
00732
00733                     /*
00734                      * shutdown GTI power at low DL battery Ah or Voltage
00735                      */
00736                     if ((E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) || (E.mvar[V_DVBAT] < PV_DL_B_V_LOW)) {
00737                         error_drive = PV_BIAS_ZERO;
00738                     }
00739
00740                     snprintf(gti_str, SBUF_SIZ - 1, "V%04dX", error_drive); // format for dumpload
      controller gti power commands
00741                     mqtt_gti_power(E.client_p, TOPIC_P, gti_str, 2);
00742                 }
00743
00744 #ifndef  FAKE_VPV
00745                 if (fm80_float(true) || ((ac1_filter(E.mvar[V_BEN]) > BAL_MAX_ENERGY_AC) && (ac_test()
      > MIN_BAT_KW_AC_HI))) {
00746                     ramp_up_ac(E.client_p, E.ac_sw_on); // use once control
00747 #ifdef PSW_DEBUG
00748                     fprintf(fout, "%s MIN_BAT_KW_AC_HI AC switch %d \r\n", log_time(false),
      E.ac_sw_on);
00749 #endif
00750                     E.ac_sw_on = false; // once flag
00751                 }
00752 #endif
00753                 if (((ac2_filter(E.mvar[V_BEN]) < BAL_MIN_ENERGY_AC) || ((ac_test() <
      (MIN_BAT_KW_AC_LO + E.ac_low_adj))))) {
00754                     if (!fm80_float(true)) {
00755                         ramp_down_ac(E.client_p, E.ac_sw_on);
00756                         if (log_timer()) {
00757                             fprintf(fout, "%s RAMP DOWN AC, MIN_BAT_KW_AC_LO AC switch %d \r\n",
      log_time(false), E.ac_sw_on);
00758                         }
00759                     }
00760                     E.ac_sw_on = true;
00761                 }
00762
00763
00764                 /*
00765                  * Dump Load Excess testing
00766                  * send excess power into the home power grid taking care not to export energy to the
      utility grid
00767                  */
00768                 if (((dc1_filter(E.mvar[V_BEN]) > BAL_MAX_ENERGY_GTI) && (gti_test() >
      MIN_BAT_KW_GTI_HI)) || E.dl_excess) {
00769 #ifndef  FAKE_VPV
00770 #ifdef B_DLE_DEBUG
00771                     if (E.dl_excess) {
00772                         fprintf(fout, "%s DL excess ramp_up_gti, DC switch %d\r\n", log_time(false),
      E.gti_sw_on);
00773                     }
00774 #endif
00775                     ramp_up_gti(E.client_p, E.gti_sw_on, E.dl_excess);
00776                     if (log_timer()) {
00777                         fprintf(fout, "%s RAMP DOWN DC, MIN_BAT_KW_GTI_HI DC switch %d \r\n",
      log_time(false), E.gti_sw_on);
00778                     }
00779                     E.gti_sw_on = false; // once flag
00780 #endif
00781                 } else {
00782                     if ((dc2_filter(E.mvar[V_BEN]) < BAL_MIN_ENERGY_GTI) || (gti_test() <
      (MIN_BAT_KW_GTI_LO + E.gti_low_adj))) {
```

```
00783                         if (!E.dl_excess) {
00784                             if (log_timer()) {
00785                                 ramp_down_gti(E.client_p, true);
00786 #ifdef PSW_DEBUG
00787                                 fprintf(fout, "%s MIN_BAT_KW_GTI_LO DC switch %d \r\n",
      log_time(false), E.gti_sw_on);
00788 #endif
00789                             }
00790                             E.gti_sw_on = true;
00791                         }
00792                     }
00793                 }
00794             };
00795
00796 #ifdef B_ADJ_DEBUG
00797             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);
00798 #endif
00799 #ifdef B_DLE_DEBUG
00800             if (E.dl_excess) {
00801                 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);
00802             }
00803 #endif
00804
00805             time(&rawtime);
00806
00807             if (E.im_delay++ >= IM_DELAY) {
00808                 E.im_delay = 0;
00809                 iammeter_read();
00810             }
00811             if (E.im_display++ >= IM_DISPLAY) {
00812                 char buffer[SYSLOG_SIZ];
00813                 uint32_t len;
00814
00815                 E.im_display = 0;
00816                 mqtt_ha_pid(E.client_p, TOPIC_PPID);
00817                 if (!(E.fm80 && E.dumpload && E.iammeter)) {
00818                     if (!E.iammeter) {
00819                         E.link.iammeter_error++;
00820                     } else {
00821                         E.link.mqtt_error++;
00822                     }
00823                     E.link.shutdown++;
00824                     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,
00825                         E.link.mqtt_count, E.link.mqtt_error, E.link.iammeter_count,
      E.link.iammeter_error);
00826                     fflush(fout);
00827                     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,
00828                         E.link.mqtt_count, E.link.mqtt_error, E.link.iammeter_count,
      E.link.iammeter_error);
00829                     syslog(LOG_NOTICE, buffer);
00830                     mqtt_ha_shutdown(E.client_p, TOPIC_SHUTDOWN);
00831                     E.mode.data_error = true;
00832                 } else {
00833                     E.mode.data_error = false;
00834                     E.link.shutdown = 0;
00835                 }
00836                 snprintf(buffer, RBUF_SIZ - 1, "%s", ctime(&rawtime));
00837                 len = strlen(buffer);
00838                 buffer[len - 1] = 0; // munge out the return character
00839                 fprintf(fout, "%s ", buffer);
00840                 fflush(fout);
00841                 E.fm80 = false;
00842                 E.dumpload = false;
00843                 E.homeassistant = false;
00844                 E.iammeter = false;
00845                 sync_ha();
00846                 print_im_vars();
00847                 print_mvar_vars();
00848                 fprintf(fout, "%s\r", ctime(&rawtime));
00849             }
00850             E.mode.E = E_WAIT;
00851             fflush(fout);
00852             if (E.mode.con6) {
00853                 E.mode.R = R_IDLE;
00854             }
00855             if (E.mode.con7) {
00856                 E.mode.E = E_STOP;
00857             }
00858             break;
00859         case E_STOP:
00860         default:
00861             fflush(fout);
```

```
00862             fprintf(fout, "\r\n%s HA Energy stopped and exited.\r\n", log_time(false));
00863             fflush(fout);
00864             return 0;
00865             break;
00866         }
00867     }
00868 }
```

4.1.2.9 ramp_down_ac()

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

```
00968 {
00969     if (sw_off) {
00970         mqtt_ha_switch(client_p, TOPIC_PACC, false);
00971         E.ac_sw_status = false;
00972         usleep(500000);
00973     }
00974     E.once_ac = true;
00975 }
```

4.1.2.10 ramp_down_gti()

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

```
00935 {
00936     if (sw_off) {
00937         mqtt_ha_switch(client_p, TOPIC_PDCC, false);
00938         E.once_gti_zero = true;
00939         E.gti_sw_status = false;
00940     }
00941     E.once_gti = true;
00942
00943     if (E.once_gti_zero) {
00944         mqtt_gti_power(client_p, TOPIC_P, "Z#", 7); // zero power
00945         E.once_gti_zero = false;
00946     }
00947 }
```

4.1.2.11 ramp_up_ac()

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

```
00953 {
00954
00955     if (start) {
00956         E.once_ac = true;
00957     }
00958
00959     if (E.once_ac) {
00960         E.once_ac = false;
00961         mqtt_ha_switch(client_p, TOPIC_PACC, true);
00962         E.ac_sw_status = true;
00963         usleep(500000); // wait for voltage to ramp
00964     }
00965 }
```

4.1.2.12 ramp_up_gti()

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

```
00874 {
00875     static uint32_t sequence = 0;
00876 
00877     if (start) {
00878         E.once_gti = true;
00879     }
00880 
00881     if (E.once_gti) {
00882         E.once_gti = false;
00883         sequence = 0;
00884         if (!excess) {
00885             mqtt_ha_switch(client_p, TOPIC_PDCC, true);
00886             E.gti_sw_status = true;
00887             usleep(500000); // wait for voltage to ramp
00888         } else {
00889             sequence = 1;
00890         }
00891     }
00892 
00893     switch (sequence) {
00894     case 4:
00895         E.once_gti_zero = true;
00896         break;
00897     case 3:
00898     case 2:
00899     case 1:
00900         E.once_gti_zero = true;
00901         if (bat_current_stable() || E.dl_excess) { // check battery current std dev, stop
    'motorboating'
00902             sequence++;
00903             if (!mqtt_gti_power(client_p, TOPIC_P, "+#", 3)) {
00904                 sequence = 0;
00905             }; // +100W power
00906         } else {
00907             usleep(500000); // wait a bit more for power to be stable
00908             sequence = 1; // do power ramps when ready
00909             if (!mqtt_gti_power(client_p, TOPIC_P, "-#", 4)) {
00910                 sequence = 0;
00911             }; // - 100W power
00912         }
00913         break;
00914     case 0:
00915         sequence++;
00916         if (E.once_gti_zero) {
00917             mqtt_gti_power(client_p, TOPIC_P, "Z#", 5); // zero power
00918             E.once_gti_zero = false;
00919         }
00920         break;
00921     default:
00922         if (E.once_gti_zero) {
00923             mqtt_gti_power(client_p, TOPIC_P, "Z#", 6); // zero power
00924             E.once_gti_zero = false;
00925         }
00926         sequence = 0;
00927         break;
00928     }
00929 }
```

4.1.2.13 sanity_check()

```
bool sanity_check (
            void )
```

```
00257 {
00258     if (E.mvar[V_PWA] > PWA_SANE) {
00259         E.sane = S_PWA;
00260         return false;
00261     }
00262     if (E.mvar[V_PAMPS] > PAMPS_SANE) {
00263         E.sane = S_PAMPS;
00264         return false;
00265     }
00266     if (E.mvar[V_PVOLTS] > PVOLTS_SANE) {
```

```
00267         E.sane = S_PVOLTS;
00268         return false;
00269     }
00270     if (E.mvar[V_FBAMPS] > BAMPS_SANE) {
00271         E.sane = S_FBAMPS;
00272         return false;
00273     }
00274     return true;
00275 }
```

4.1.2.14 showIP()

```
void showIP (
             void )
```

```
00163 {
00164     struct ifaddrs *ifaddr, *ifa;
00165     int s;
00166     char host[NI_MAXHOST];
00167
00168     if (getifaddrs(&ifaddr) == -1) {
00169         perror("getifaddrs");
00170         exit(EXIT_FAILURE);
00171     }
00172
00173
00174     for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
00175         if (ifa->ifa_addr == NULL)
00176             continue;
00177
00178         s = getnameinfo(ifa->ifa_addr, sizeof(struct sockaddr_in), host, NI_MAXHOST, NULL, 0,
      NI_NUMERICHOST);
00179
00180         if (ifa->ifa_addr->sa_family == AF_INET) {
00181             if (s != 0) {
00182                 exit(EXIT_FAILURE);
00183             }
00184             printf("\tInterface : <%s>\n", ifa->ifa_name);
00185             printf("\t  Address : <%s>\n", host);
00186         }
00187     }
00188
00189     freeifaddrs(ifaddr);
00190 }
```

4.1.2.15 skeleton_daemon()

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

```
00197 {
00198     pid_t pid;
00199
00200     /* Fork off the parent process */
00201     pid = fork();
00202
00203     /* An error occurred */
00204     if (pid < 0) {
00205         printf("\r\n%s DAEMON failure  LOG Version %s : MQTT Version %s\r\n", log_time(false),
      LOG_VERSION, MQTT_VERSION);
00206         exit(EXIT_FAILURE);
00207     }
00208
00209     /* Success: Let the parent terminate */
00210     if (pid > 0) {
00211         exit(EXIT_SUCCESS);
00212     }
00213
00214     /* On success: The child process becomes session leader */
00215     if (setsid() < 0) {
00216         exit(EXIT_FAILURE);
00217     }
00218
00219     /* Catch, ignore and handle signals */
00220     /*TODO: Implement a working signal handler */
00221     //    signal(SIGCHLD, SIG_IGN);
00222     //    signal(SIGHUP, SIG_IGN);
00223
00224     /* Fork off for the second time*/
```

```
00225     pid = fork();
00226
00227     /* An error occurred */
00228     if (pid < 0) {
00229         exit(EXIT_FAILURE);
00230     }
00231
00232     /* Success: Let the parent terminate */
00233     if (pid > 0) {
00234         exit(EXIT_SUCCESS);
00235     }
00236
00237     /* Set new file permissions */
00238     umask(0);
00239
00240     /* Change the working directory to the root directory */
00241     /* or another appropriated directory */
00242     chdir("/");
00243
00244     /* Close all open file descriptors */
00245     int x;
00246     for (x = sysconf(_SC_OPEN_MAX); x >= 0; x--) {
00247         close(x);
00248     }
00249
00250 }
```

4.1.2.16 solar_shutdown()

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

```
01008 {
01009     static bool ret = false;
01010
01011     if (E.startup) {
01012         ret = true;
01013         E.startup = false;
01014         return ret;
01015     } else {
01016         ret = false;
01017
01018         /*
01019          * FIXME
01020          *
01021          */
01022     }
01023
01024     if (E.solar_shutdown) {
01025         ret = true;
01026     } else {
01027         ret = false;
01028     }
01029
01030     if ((E.mvar[V_FBEKW] < BAT_CRITICAL) && !E.startup) { // special case for low battery
01031         if (!E.mode.bat_crit) {
01032             ret = true;
01033 #ifdef CRITIAL_SHUTDOWN_LOG
01034             fprintf(fout, "%s Solar BATTERY CRITICAL shutdown comms check ret = %d \r\n",
      log_time(false), ret);
01035             fflush(fout);
01036 #endif
01037             E.mode.bat_crit = true;
01038             return ret;
01039         }
01040     } else {
01041         E.mode.bat_crit = false;
01042     }
01043
01044     if (E.link.shutdown >= MAX_ERROR) {
01045         ret = true;
01046         if (E.fm80 && E.dumpload && E.iammeter) {
01047             ret = false;
01048             E.link.shutdown = 0;
01049         }
01050
01051 #ifdef DEBUG_SHUTDOWN
01052         fprintf(fout, "%s Solar shutdown comms check ret = %d \r\n", log_time(false), ret);
01053         fflush(fout);
01054 #endif
01055     }
01056     return ret;
01057 }
```

4.1.2.17 sync_ha()

```
bool sync_ha (
            void )
```

```
01093 {
01094     bool sync = false;
01095     if (E.gti_sw_status != (bool) ((int32_t) E.mvar[V_HDCSW])) {
01096         fprintf(fout, "DC_MM %d %d ", (bool) E.gti_sw_status, (bool) ((int32_t) E.mvar[V_HDCSW]));
01097         mqtt_ha_switch(E.client_p, TOPIC_PDCC, !E.gti_sw_status);
01098         E.dc_mismatch = true;
01099         fflush(fout);
01100         sync = true;
01101     } else {
01102         E.dc_mismatch = false;
01103     }
01104
01105     E.ac_sw_status = (bool) ((int32_t) E.mvar[V_HACSW]); // TEMP FIX for MISmatch errors
01106     if (E.ac_sw_status != (bool) ((int32_t) E.mvar[V_HACSW])) {
01107         fprintf(fout, "AC_MM %d %d ", (bool) E.ac_sw_status, (bool) ((int32_t) E.mvar[V_HACSW]));
01108         mqtt_ha_switch(E.client_p, TOPIC_PACC, !E.ac_sw_status);
01109         E.ac_mismatch = true;
01110         fflush(fout);
01111         sync = true;
01112     } else {
01113         E.ac_mismatch = false;
01114     }
01115     return sync;
01116 }
```

4.1.2.18 timer_callback()

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

```
00286 {
00287     signal(signum, timer_callback);
00288     ha_flag_vars_ss.runner = true;
00289     E.ten_sec_clock++;
00290     E.log_spam++;
00291     E.log_time_reset++;
00292     if (E.log_spam > MAX_LOG_SPAM) {
00293         E.log_spam = 0;
00294     }
00295 }
```

4.1.3 Variable Documentation

4.1.3.1 E

```
struct energy_type E
```

```
00110                       {
00111     .once_gti = true,
00112     .once_ac = true,
00113     .once_gti_zero = true,
00114     .iammeter = false,
00115     .fm80 = false,
00116     .dumpload = false,
00117     .homeassistant = false,
00118     .ac_low_adj = 0.0f,
00119     .gti_low_adj = 0.0f,
00120     .ac_sw_on = true,
00121     .gti_sw_on = true,
00122     .im_delay = 0,
00123     .gti_delay = 0,
00124     .im_display = 0,
00125     .rc = 0,
00126     .speed_go = 0,
00127     .mode.pid.iMax = PV_IMAX,
00128     .mode.pid.iMin = 0.0f,
00129     .mode.pid.pGain = PV_PGAIN,
00130     .mode.pid.iGain = PV_IGAIN,
00131     .mode.mode_tmr = 0,
```

```
00132     .mode.mode = true,
00133     .mode.in_pid_control = false,
00134     .mode.dl_mqtt_max = PV_DL_MPTT_MAX,
00135     .mode.E = E_INIT,
00136     .mode.R = R_INIT,
00137     .mode.no_float = true,
00138     .mode.data_error = false,
00139     .ac_sw_status = false,
00140     .gti_sw_status = false,
00141     .solar_mode = false,
00142     .solar_shutdown = false,
00143     .mode.pv_bias = PV_BIAS_LOW,
00144     .sane = S_DLAST,
00145     .startup = true,
00146     .ac_mismatch = false,
00147     .dc_mismatch = false,
00148     .mode_mismatch = false,
00149     .link.shutdown = 0,
00150     .mode.bat_crit = false,
00151     .dl_excess = false,
00152     .dl_excess_adj = 0.0f,
00153 };
```

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,
}
```

```
00095                                       {
00096     .runner = false,
00097     .receivedtoken = false,
00098     .deliveredtoken = false,
00099     .rec_ok = false,
00100     .ha_id = HA_ID,
00101     .var_update = 0,
00102 };
```

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,
}
```

```
00064                                       {
00065     .runner = false,
00066     .receivedtoken = false,
00067     .deliveredtoken = false,
00068     .rec_ok = false,
00069     .ha_id = P8055_ID,
00070     .var_update = 0,
00071 };
```

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,
}
```

```
00085                                       {
00086     .runner = false,
00087     .receivedtoken = false,
00088     .deliveredtoken = false,
00089     .rec_ok = false,
00090     .ha_id = DUMPLOAD_ID,
00091     .var_update = 0,
00092 };
```

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,
}
```

```
00074                                       {
00075     .runner = false,
00076     .receivedtoken = false,
00077     .deliveredtoken = false,
00078     .rec_ok = false,
00079     .ha_id = FM80_ID,
00080     .var_update = 0,
00081     .energy_mode = NORM_MODE,
00082 };
```

4.2 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
00061     double calculateStandardDeviation(const uint32_t, const double *);
00062
00063     bool bsoc_set_mode(const double, const bool, const bool);
00064
00065     double ac0_filter(const double);
00066     double ac1_filter(const double);
00067     double ac2_filter(const double);
00068     double dc0_filter(const double);
00069     double dc1_filter(const double);
00070     double dc2_filter(const double);
00071     double drive0_filter(const double);
00072     double drive1_filter(const double);
00073
00074 #ifdef __cplusplus
00075 }
00076 #endif
00077
00078 #endif /* BSOC_H */
00079
```

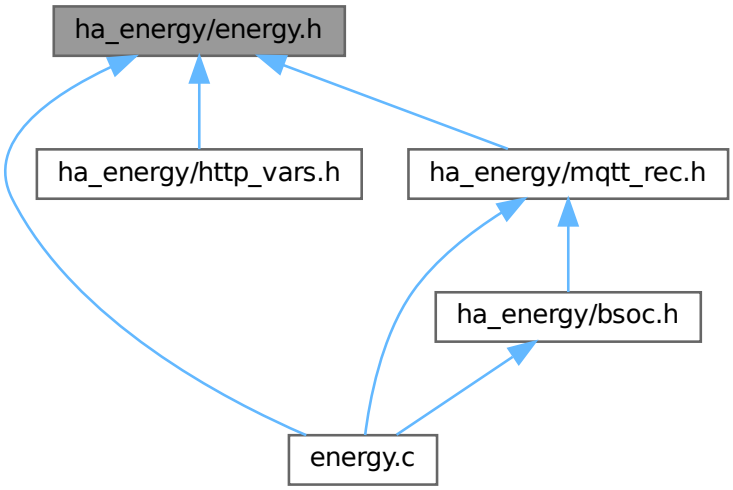
4.3 ha_energy/energy.h File Reference

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

```
#include <syslog.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netdb.h>
#include <ifaddrs.h>
#include "MQTTClient.h"
#include "pid.h"
```

Include dependency graph for energy.h:

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



Data Structures

- struct link_type
- struct mode_type
- struct energy_type

Macros

- #define **LOG_VERSION** "V0.75"
- #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 **MIN_BAT_KW_BSOC_SLP** 4000.0f
- #define **MIN_BAT_KW_BSOC_HI** 4550.0f
- #define **MIN_BAT_KW_GTI_HI** BAT_M_KW∗BAT_SOC_TOP
- #define **MIN_BAT_KW_GTI_LO** BAT_M_KW∗BAT_SOC_LOW
- #define **MIN_BAT_KW_AC_HI** BAT_M_KW∗BAT_SOC_HIGH
- #define **MIN_BAT_KW_AC_LO** BAT_M_KW∗BAT_SOC_LOW_AC
- #define **PV_PGAIN** 0.85f
- #define **PV_IGAIN** 0.12f
- #define **PV_IMAX** 1400.0f
- #define **PV_BIAS** 288.0f
- #define **PV_BIAS_ZERO** 0.0f
- #define **PV_BIAS_LOW** 222.0f
- #define **PV_BIAS_FLOAT** 399.0f
- #define **PV_BIAS_SLEEP** 480.0f
- #define **PV_BIAS_RATE** 320.0f

- #define **PV_DL_MPTT_MAX** 1200.0f
- #define **PV_DL_MPTT_EXCESS** 1300.0f
- #define **PV_DL_MPTT_IDLE** 57.0f
- #define **PV_DL_BIAS_RATE** 75.0f
- #define **PV_DL_EXCESS** 500.0f
- #define **PV_DL_B_AH_LOW** 100.0f
- #define **PV_DL_B_AH_MIN** 150.0f
- #define **PV_DL_B_V_LOW** 23.8f
- #define **PWA_SLEEP** 200.0f
- #define **DL_AC_DC_EFF** 1.24f
- #define **BAL_MIN_ENERGY_AC** -200.0f
- #define **BAL_MAX_ENERGY_AC** 200.0f
- #define **BAL_MIN_ENERGY_GTI** -1400.0f
- #define **BAL_MAX_ENERGY_GTI** 200.0f
- #define **LOG_TO_FILE** "/store/logs/energy.log"
- #define **LOG_TO_FILE_ALT** "/tmp/energy.log"
- #define **MAX_LOG_SPAM** 60
- #define **LOW_LOG_SPAM** 2
- #define **RESET_LOG_SPAM** 120
- #define **IM_DELAY** 1
- #define **IM_DISPLAY** 1
- #define **GTI_DELAY** 1
- #define **PWA_SANE** 1700.0f
- #define **PAMPS_SANE** 16.0f
- #define **PVOLTS_SANE** 150.0f
- #define **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

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_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)
- size_t iammeter_write_callback (char ∗, size_t, size_t, void ∗)
- void iammeter_read (void)
- void print_im_vars (void)
- void print_mvar_vars (void)
- bool sanity_check (void)
- char ∗ log_time (bool)
- bool sync_ha (void)
- bool log_timer (void)

Variables

- struct energy_type E
- struct ha_flag_type ha_flag_vars_ss
- FILE ∗ **fout**

4.3.1 Enumeration Type Documentation

4.3.1.1 client_id

```
enum client_id
00194                   {
00195         ID_C1,
00196         ID_C2,
00197         ID_C3,
00198     };
```

4.3.1.2 energy_state

```
enum energy_state
00200                       {
00201         E_INIT,
00202         E_RUN,
00203         E_WAIT,
00204         E_IDLE,
00205         E_STOP,
00206         E_LAST,
00207     };
```

4.3.1.3 iammeter_id

```
enum iammeter_id
00225                       {
00226         IA_VOLTAGE,
00227         IA_CURRENT,
00228         IA_POWER,
00229         IA_IMPORT,
00230         IA_EXPORT,
00231         IA_FREQ,
00232         IA_PF,
00233         IA_LAST,
00234     };
```

4.3.1.4 iammeter_phase

```
enum iammeter_phase
00218                           {
00219         PHASE_A,
00220         PHASE_B,
00221         PHASE_C,
00222         PHASE_LAST,
00223     };
```

4.3.1.5 mqtt_vars

```
enum mqtt_vars
00236                       {
00237         V_FCCM,
00238         V_FBEKW,
00239         V_FRUNT,
00240         V_FBAMPS,
00241         V_FBV,
00242         V_FLO,
00243         V_FSO,
00244         V_FACE,
00245         V_BEN,
00246         V_PWA,
00247         V_PAMPS,
00248         V_PVOLTS,
00249         V_FLAST,
00250         V_HDCSW,
00251         V_HACSW,
00252         V_HSHUT,
00253         V_HMODE,
00254         V_HCON0,
00255         V_HCON1,
00256         V_HCON2,
00257         V_HCON3,
00258         V_HCON4,
00259         V_HCON5,
00260         V_HCON6,
00261         V_HCON7,
00262         // add other data ranges here
00263         V_DVPV,
00264         V_DPPV,
00265         V_DPBAT,
00266         V_DVBAT,
00267         V_DCMPPT,
00268         V_DPMPPT,
00269         V_DAHBAT,
00270         V_DCCMODE,
00271         V_DGTI,
00272         V_DLAST,
00273     };
```

4.3.1.6 running_state

```
enum running_state
00209                       {
00210         R_INIT,
00211         R_FLOAT,
00212         R_SLEEP,
00213         R_RUN,
00214         R_IDLE,
00215         R_LAST,
00216     };
```

4.3.1.7 sane_vars

```
enum sane_vars
00275                   {
00276         S_FCCM,
00277         S_FBEKW,
00278         S_FRUNT,
00279         S_FBAMPS,
00280         S_FBV,
00281         S_FLO,
00282         S_FSO,
00283         S_FACE,
00284         S_BEN,
00285         S_PWA,
00286         S_PAMPS,
00287         S_PVOLTS,
00288         S_FLAST,
00289         S_HDCSW,
00290         S_HACSW,
00291         S_HSHUT,
00292         S_HMODE,
00293         // add other data ranges here
00294         S_DVPV,
00295         S_DPPV,
00296         S_DPBAT,
00297         S_DVBAT,
00298         S_DCMPPT,
00299         S_DPMPPT,
00300         S_DAHBAT,
00301         S_DCCMODE,
00302         S_DGTI,
00303         S_DLAST,
00304     };
```

4.3.2 Function Documentation

4.3.2.1 connlost()

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

trouble in River-city

```
00301 {
00302     struct ha_flag_type *ha_flag = context;
00303     int32_t id_num = ha_flag->ha_id;
00304     static uint32_t times = 0;
00305     char * where = "Missing Topic";
00306
00307     switch (ha_flag->cid) {
00308     case ID_C1:
00309         where = TOPIC_SS;
00310         break;
00311     case ID_C2:
00312         where = TOPIC_SD;
00313         break;
00314     case ID_C3:
00315         where = TOPIC_HA;
00316         break;
```

```
00317     }
00318
00319     // bug-out if no context variables passed to callback
00320     if (context == NULL) {
00321         id_num = -1;
00322         goto bugout;
00323     } else {
00324         if (times++ > MQTT_RECONN) {
00325             goto bugout;
00326         } else {
00327
00328             fprintf(fout, "\n%s Connection lost, exit ha_energy program\n", log_time(false));
00329             fprintf(fout, "%s     cause: %s, h_id %d c_id %d %s \n", log_time(false), cause, id_num,
      ha_flag->cid, where);
00330             fprintf(fout, "%s MQTT DAEMON failure  LOG Version %s : MQTT Version %s\n",
      log_time(false), LOG_VERSION, MQTT_VERSION);
00331             fflush(fout);
00332         }
00333     }
00334
00335 bugout:
00336     fprintf(fout, "\n%s Connection lost, exit ha_energy program\n", log_time(false));
00337     fprintf(fout, "%s     cause: %s, h_id %d c_id %d %s \n", log_time(false), cause, id_num,
      ha_flag->cid, where);
00338     fprintf(fout, "%s MQTT DAEMON failure  LOG Version %s : MQTT Version %s\n", log_time(false),
      LOG_VERSION, MQTT_VERSION);
00339     fflush(fout);
00340     exit(EXIT_FAILURE);
00341 }
```

4.3.2.2 ha_ac_off()

```
void ha_ac_off (
              void )
00978 {
00979     mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
00980     E.ac_sw_status = false;
00981 }
```

4.3.2.3 ha_ac_on()

```
void ha_ac_on (
              void )
00984 {
00985     mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00986     E.ac_sw_status = true;
00987 }
```

4.3.2.4 ha_dc_off()

```
void ha_dc_off (
              void )
00993 {
00994     mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
00995     E.gti_sw_status = false;
00996 }
```

4.3.2.5 ha_dc_on()

```
void ha_dc_on (
              void )
00999 {
01000     mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
01001     E.gti_sw_status = true;
01002 }
```

4.3.2.6 iammeter_read()

```
void iammeter_read (
            void )
```

```
00076 {
00077
00078     curl = curl_easy_init();
00079     if (curl) {
00080         E.link.iammeter_count++;
00081         curl_easy_setopt(curl, CURLOPT_URL, "http://10.1.1.101/monitorjson");
00082         curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, iammeter_write_callback);
00083         curl_easy_setopt(curl, CURLOPT_WRITEDATA, E.print_vars); // external data array for iammeter
      values
00084
00085         res = curl_easy_perform(curl);
00086         /* Check for errors */
00087         if (res != CURLE_OK) {
00088             fprintf(fout, "curl_easy_perform() failed in iammeter_read: %s\n",
00089                 curl_easy_strerror(res));
00090             E.iammeter = false;
00091             E.link.iammeter_error++;
00092         } else {
00093             E.iammeter = true;
00094         }
00095         curl_easy_cleanup(curl);
00096     }
00097 }
```

4.3.2.7 iammeter_write_callback()

```
size_t iammeter_write_callback (
            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, "Error in iammeter_write_callback %u: %s\n", E.link.iammeter_error,
      error_ptr);
00026         }
00027         goto iammeter_exit;
00028     }
00029 #ifdef IM_DEBUG
00030     fprintf(fout, "\n iammeter_read_callback %s \n", buffer);
00031 #endif
00032
00033     cJSON *data_result = cJSON_GetObjectItemCaseSensitive(json, "Datas");
00034
00035     if (!data_result) {
00036         size = 0;
00037         nitems = 0;
00038         goto iammeter_exit;
00039     }
00040
00041     cJSON *jname;
00042     uint32_t phase = PHASE_A;
00043
00044     cJSON_ArrayForEach(jname, data_result)
00045     {
00046         cJSON *ianame;
00047 #ifdef IM_DEBUG
00048         fprintf(fout, "\n iammeter variables ");
00049 #endif
00050
00051         cJSON_ArrayForEach(ianame, jname)
00052         {
00053             uint32_t phase_var = IA_VOLTAGE;
00054             iammeter_get_data(ianame->valuedouble, phase_var, phase);
```

```
00055             e->print_vars[next_var++] = ianame->valuedouble;
00056 #ifdef IM_DEBUG
00057             fprintf(fout, "%8.2f ", im_vars[phase_var][phase]);
00058 #endif
00059             phase_var++;
00060         }
00061         phase++;
00062     }
00063 #ifdef IM_DEBUG
00064     fprintf(fout, "\n");
00065 #endif
00066
00067 iammeter_exit:
00068     cJSON_Delete(json);
00069     return size * nitems;
00070 }
```

4.3.2.8 log_time()

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

```
01063 {
01064     static char time_log[RBUF_SIZ] = {0};
01065     static uint32_t len = 0, sync_time = TIME_SYNC_SEC - 1;
01066     time_t rawtime_log;
01067
01068     tzset();
01069     timezone = 0;
01070     daylight = 0;
01071     time(&rawtime_log);
01072     if (sync_time++ > TIME_SYNC_SEC) {
01073         sync_time = 0;
01074         snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
      time commands
01075         mqtt_gti_time(E.client_p, TOPIC_P, time_log);
01076     }
01077
01078     sprintf(time_log, "%s", ctime(&rawtime_log));
01079     len = strlen(time_log);
01080     time_log[len - 1] = 0; // munge out the return character
01081     if (log) {
01082         fprintf(fout, "%s ", time_log);
01083         fflush(fout);
01084     }
01085
01086     return time_log;
01087 }
```

4.3.2.9 log_timer()

```
bool log_timer (
            void )
```

```
01122 {
01123     bool itstime = false;
01124
01125     if (E.log_spam < LOW_LOG_SPAM) {
01126         E.log_time_reset = 0;
01127         itstime = true;
01128     }
01129     if (E.log_time_reset > RESET_LOG_SPAM) {
01130         E.log_spam = 0;
01131         itstime = true;
01132     }
01133     return itstime;
01134 }
```

4.3.2.10 print_im_vars()

```
void print_im_vars (
            void )
```

```
00111 {
00112     static char time_log[RBUF_SIZ] = {0};
00113     static uint32_t sync_time = TIME_SYNC_SEC - 1;
00114     time_t rawtime_log;
00115     char imvars[SYSLOG_SIZ];
00116 
00117     fflush(fout);
00118     snprintf(imvars, SYSLOG_SIZ-1, "House L1 %7.2fW, House L2 %7.2fW, GTI L1 %7.2fW",
      E.print_vars[L1_P], E.print_vars[L2_P], E.print_vars[L3_P]);
00119     fprintf(fout, "%s", imvars);
00120     fflush(fout);
00121     time(&rawtime_log);
00122     if (sync_time++ > TIME_SYNC_SEC) {
00123         sync_time = 0;
00124         snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
      time commands
00125         mqtt_gti_time(E.client_p, TOPIC_P, time_log);
00126     }
00127 }
```

4.3.2.11 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 }
```

4.3.2.12 ramp_down_ac()

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

```
00968 {
00969     if (sw_off) {
00970         mqtt_ha_switch(client_p, TOPIC_PACC, false);
00971         E.ac_sw_status = false;
00972         usleep(500000);
00973     }
00974     E.once_ac = true;
00975 }
```

4.3.2.13 ramp_down_gti()

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

```
00935 {
00936     if (sw_off) {
00937         mqtt_ha_switch(client_p, TOPIC_PDCC, false);
00938         E.once_gti_zero = true;
00939         E.gti_sw_status = false;
00940     }
00941     E.once_gti = true;
00942 
00943     if (E.once_gti_zero) {
00944         mqtt_gti_power(client_p, TOPIC_P, "Z#", 7); // zero power
00945         E.once_gti_zero = false;
00946     }
00947 }
```

4.3.2.14 ramp_up_ac()

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

```
00953 {
00954
00955     if (start) {
00956         E.once_ac = true;
00957     }
00958
00959     if (E.once_ac) {
00960         E.once_ac = false;
00961         mqtt_ha_switch(client_p, TOPIC_PACC, true);
00962         E.ac_sw_status = true;
00963         usleep(500000); // wait for voltage to ramp
00964     }
00965 }
```

4.3.2.15 ramp_up_gti()

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

```
00874 {
00875     static uint32_t sequence = 0;
00876
00877     if (start) {
00878         E.once_gti = true;
00879     }
00880
00881     if (E.once_gti) {
00882         E.once_gti = false;
00883         sequence = 0;
00884         if (!excess) {
00885             mqtt_ha_switch(client_p, TOPIC_PDCC, true);
00886             E.gti_sw_status = true;
00887             usleep(500000); // wait for voltage to ramp
00888         } else {
00889             sequence = 1;
00890         }
00891     }
00892
00893     switch (sequence) {
00894     case 4:
00895         E.once_gti_zero = true;
00896         break;
00897     case 3:
00898     case 2:
00899     case 1:
00900         E.once_gti_zero = true;
00901         if (bat_current_stable() || E.dl_excess) { // check battery current std dev, stop
    'motorboating'
00902             sequence++;
00903             if (!mqtt_gti_power(client_p, TOPIC_P, "+#", 3)) {
00904                 sequence = 0;
00905             }; // +100W power
00906         } else {
00907             usleep(500000); // wait a bit more for power to be stable
00908             sequence = 1; // do power ramps when ready
00909             if (!mqtt_gti_power(client_p, TOPIC_P, "-#", 4)) {
00910                 sequence = 0;
00911             }; // - 100W power
00912         }
00913         break;
00914     case 0:
00915         sequence++;
00916         if (E.once_gti_zero) {
00917             mqtt_gti_power(client_p, TOPIC_P, "Z#", 5); // zero power
00918             E.once_gti_zero = false;
00919         }
00920         break;
00921     default:
00922         if (E.once_gti_zero) {
00923             mqtt_gti_power(client_p, TOPIC_P, "Z#", 6); // zero power
00924             E.once_gti_zero = false;
```

```
00925         }
00926         sequence = 0;
00927         break;
00928     }
00929 }
```

4.3.2.16 sanity_check()

```
bool sanity_check (
             void )
```

```
00257 {
00258     if (E.mvar[V_PWA] > PWA_SANE) {
00259         E.sane = S_PWA;
00260         return false;
00261     }
00262     if (E.mvar[V_PAMPS] > PAMPS_SANE) {
00263         E.sane = S_PAMPS;
00264         return false;
00265     }
00266     if (E.mvar[V_PVOLTS] > PVOLTS_SANE) {
00267         E.sane = S_PVOLTS;
00268         return false;
00269     }
00270     if (E.mvar[V_FBAMPS] > BAMPS_SANE) {
00271         E.sane = S_FBAMPS;
00272         return false;
00273     }
00274     return true;
00275 }
```

4.3.2.17 sync_ha()

```
bool sync_ha (
             void )
```

```
01093 {
01094     bool sync = false;
01095     if (E.gti_sw_status != (bool) ((int32_t) E.mvar[V_HDCSW])) {
01096         fprintf(fout, "DC_MM %d %d ", (bool) E.gti_sw_status, (bool) ((int32_t) E.mvar[V_HDCSW]));
01097         mqtt_ha_switch(E.client_p, TOPIC_PDCC, !E.gti_sw_status);
01098         E.dc_mismatch = true;
01099         fflush(fout);
01100         sync = true;
01101     } else {
01102         E.dc_mismatch = false;
01103     }
01104
01105     E.ac_sw_status = (bool) ((int32_t) E.mvar[V_HACSW]); // TEMP FIX for MISmatch errors
01106     if (E.ac_sw_status != (bool) ((int32_t) E.mvar[V_HACSW])) {
01107         fprintf(fout, "AC_MM %d %d ", (bool) E.ac_sw_status, (bool) ((int32_t) E.mvar[V_HACSW]));
01108         mqtt_ha_switch(E.client_p, TOPIC_PACC, !E.ac_sw_status);
01109         E.ac_mismatch = true;
01110         fflush(fout);
01111         sync = true;
01112     } else {
01113         E.ac_mismatch = false;
01114     }
01115     return sync;
01116 }
```

4.3.2.18 timer_callback()

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

```
00286 {
00287     signal(signum, timer_callback);
00288     ha_flag_vars_ss.runner = true;
00289     E.ten_sec_clock++;
00290     E.log_spam++;
00291     E.log_time_reset++;
00292     if (E.log_spam > MAX_LOG_SPAM) {
00293         E.log_spam = 0;
00294     }
00295 }
```

4.3.3 Variable Documentation

4.3.3.1 E

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

```
00110                       {
00111     .once_gti = true,
00112     .once_ac = true,
00113     .once_gti_zero = true,
00114     .iammeter = false,
00115     .fm80 = false,
00116     .dumpload = false,
00117     .homeassistant = false,
00118     .ac_low_adj = 0.0f,
00119     .gti_low_adj = 0.0f,
00120     .ac_sw_on = true,
00121     .gti_sw_on = true,
00122     .im_delay = 0,
00123     .gti_delay = 0,
00124     .im_display = 0,
00125     .rc = 0,
00126     .speed_go = 0,
00127     .mode.pid.iMax = PV_IMAX,
00128     .mode.pid.iMin = 0.0f,
00129     .mode.pid.pGain = PV_PGAIN,
00130     .mode.pid.iGain = PV_IGAIN,
00131     .mode.mode_tmr = 0,
00132     .mode.mode = true,
00133     .mode.in_pid_control = false,
00134     .mode.dl_mqtt_max = PV_DL_MPTT_MAX,
00135     .mode.E = E_INIT,
00136     .mode.R = R_INIT,
00137     .mode.no_float = true,
00138     .mode.data_error = false,
00139     .ac_sw_status = false,
00140     .gti_sw_status = false,
00141     .solar_mode = false,
00142     .solar_shutdown = false,
00143     .mode.pv_bias = PV_BIAS_LOW,
00144     .sane = S_DLAST,
00145     .startup = true,
00146     .ac_mismatch = false,
00147     .dc_mismatch = false,
00148     .mode_mismatch = false,
00149     .link.shutdown = 0,
00150     .mode.bat_crit = false,
00151     .dl_excess = false,
00152     .dl_excess_adj = 0.0f,
00153 };
```

4.3.3.2 ha_flag_vars_ss

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

```
00074                                    {
00075     .runner = false,
00076     .receivedtoken = false,
00077     .deliveredtoken = false,
00078     .rec_ok = false,
00079     .ha_id = FM80_ID,
00080     .var_update = 0,
00081     .energy_mode = NORM_MODE,
00082 };
```

4.4 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
00035
00036 #define LOG_VERSION     "V0.75"
00037 #define MQTT_VERSION    "V3.11"
00038 #define TNAME  "maint9"
00039 #define LADDRESS        "tcp://127.0.0.1:1883"
00040 #ifdef __amd64
00041 #define ADDRESS         "tcp://10.1.1.172:1883"
00042 #else
00043 #define ADDRESS         "tcp://10.1.1.30:1883"
00044 #endif
00045 #define CLIENTID1       "Energy_Mqtt_HA1"
00046 #define CLIENTID2       "Energy_Mqtt_HA2"
00047 #define CLIENTID3       "Energy_Mqtt_HA3"
00048 #define TOPIC_P         "mateq84/data/gticmd"
00049 #define TOPIC_SPAM      "mateq84/data/spam"
00050 #define TOPIC_PACA      "home-assistant/gtiac/availability"
00051 #define TOPIC_PDCA      "home-assistant/gtidc/availability"
00052 #define TOPIC_PACC      "home-assistant/gtiac/contact"
00053 #define TOPIC_PDCC      "home-assistant/gtidc/contact"
00054 #define TOPIC_PPID      "home-assistant/solar/pid"
00055 #define TOPIC_SHUTDOWN  "home-assistant/solar/shutdown"
00056 #define TOPIC_SS        "mateq84/data/solar"
00057 #define TOPIC_SD        "mateq84/data/dumpload"
00058 #define TOPIC_HA        "home-assistant/status/switch"
00059 #define QOS             1
00060 #define TIMEOUT         10000L
00061 #define SPACING_USEC    500 * 1000
00062 #define USEC_SEC        1000000L
00063
00064 #define DAQ_STR 32
00065 #define DAQ_STR_M DAQ_STR-1
00066
00067 #define SBUF_SIZ        16  // short buffer string size
00068 #define RBUF_SIZ        82
00069 #define SYSLOG_SIZ      512
00070
00071 #define MQTT_TIMEOUT    900
00072 #define SW_QOS          1
00073
00074 #define MQTT_RECONN     3
00075 #define KAI             60
00076
00077 #define NO_CYLON
00078 #define CRITIAL_SHUTDOWN_LOG
00079
00080 #define UNIT_TEST       2
00081 #define NORM_MODE       0
00082 #define PID_MODE        1
00083 #define MAX_ERROR       5
00084 #define IAM_DELAY       120
00085
00086 #define CMD_SEC         10
00087 #define TIME_SYNC_SEC   30
00088
00089     /*
00090      * Battery SoC cycle limits parameters
00091      */
00092 #define BAT_M_KW            5120.0f
00093 #define BAT_SOC_TOP         0.98f
00094 #define BAT_SOC_HIGH        0.95f
```

```
00095 #define BAT_SOC_LOW         0.68f
00096 #define BAT_SOC_LOW_AC      0.72f
00097 #define BAT_CRITICAL        746.0f
00098 #define MIN_BAT_KW_BSOC_SLP 4000.0f
00099 #define MIN_BAT_KW_BSOC_HI  4550.0f
00100
00101 #define MIN_BAT_KW_GTI_HI   BAT_M_KW*BAT_SOC_TOP
00102 #define MIN_BAT_KW_GTI_LO   BAT_M_KW*BAT_SOC_LOW
00103
00104 #define MIN_BAT_KW_AC_HI    BAT_M_KW*BAT_SOC_HIGH
00105 #define MIN_BAT_KW_AC_LO    BAT_M_KW*BAT_SOC_LOW_AC
00106
00107     /*
00108      * PV panel cycle limits parameters
00109      */
00110 #define PV_PGAIN            0.85f
00111 #define PV_IGAIN            0.12f
00112 #define PV_IMAX             1400.0f
00113 #define PV_BIAS             288.0f
00114 #define PV_BIAS_ZERO          0.0f
00115 #define PV_BIAS_LOW         222.0f
00116 #define PV_BIAS_FLOAT       399.0f
00117 #define PV_BIAS_SLEEP       480.0f
00118 #define PV_BIAS_RATE        320.0f
00119 #define PV_DL_MPTT_MAX     1200.0f
00120 #define PV_DL_MPTT_EXCESS  1300.0f
00121 #define PV_DL_MPTT_IDLE     57.0f
00122 #define PV_DL_BIAS_RATE     75.0f
00123 #define PV_DL_EXCESS       500.0f
00124 #define PV_DL_B_AH_LOW     100.0f
00125 #define PV_DL_B_AH_MIN     150.0f // DL battery should be at least 175Ah
00126 #define PV_DL_B_V_LOW       23.8f // Battery low-voltqage cutoff
00127 #define PWA_SLEEP          200.0f
00128 #define DL_AC_DC_EFF        1.24f
00129
00130     /*
00131      * Energy control loop parameters
00132      */
00133 #define BAL_MIN_ENERGY_AC   -200.0f
00134 #define BAL_MAX_ENERGY_AC   200.0f
00135 #define BAL_MIN_ENERGY_GTI  -1400.0f
00136 #define BAL_MAX_ENERGY_GTI  200.0f
00137
00138 #define LOG_TO_FILE         "/store/logs/energy.log"
00139 #define LOG_TO_FILE_ALT     "/tmp/energy.log"
00140
00141 #define MAX_LOG_SPAM  60
00142 #define LOW_LOG_SPAM  2
00143 #define RESET_LOG_SPAM  120
00144
00145     //#define IM_DEBUG                      // WEM3080T LOGGING
00146     //#define B_ADJ_DEBUG                   // debug printing
00147     //#define FAKE_VPV                      // NO AC CHARGER for DUMPLOAD, batteries are
      cross-connected to a parallel bank
00148     //#define PSW_DEBUG
00149     //#define DEBUG_SHUTDOWN
00150
00151     //#define AUTO_CHARGE                   // turn on dumpload charger during restarts
00152     //#define B_DLE_DEBUG    // Dump Load debugging
00153     //#define BSOC_DEGUB
00154     //#define DEBUG_HA_CMD
00155
00156 #define IM_DELAY            1   // tens of second updates
00157 #define IM_DISPLAY          1
00158 #define GTI_DELAY           1
00159
00160     /*
00161      * sane limits for system data elements
00162      */
00163 #define PWA_SANE            1700.0f
00164 #define PAMPS_SANE          16.0f
00165 #define PVOLTS_SANE         150.0f
00166 #define BAMPS_SANE          70.0f
00167
00168     /*
00169         Three Phase WiFi Energy Meter (WEM3080T)
00170     name    Unit    Description
00171     wem3080t_voltage_a  V   A phase voltage
00172     wem3080t_current_a  A   A phase current
00173     wem3080t_power_a    W   A phase active power
00174     wem3080t_importenergy_a kWh A phase import energy
00175     wem3080t_exportgrid_a   kWh A phase export energy
00176     wem3080t_frequency_a    kWh A phase frequency
00177     wem3080t_pf_a   kWh A phase power factor
00178     wem3080t_voltage_b  V   B phase voltage
00179     wem3080t_current_b  A   B phase current
00180     wem3080t_power_b    W   B phase active power
```

```
00181     wem3080t_importenergy_b kWh B phase import energy
00182     wem3080t_exportgrid_b   kWh B phase export energy
00183     wem3080t_frequency_b    kWh B phase frequency
00184     wem3080t_pf_b   kWh B phase power factor
00185     wem3080t_voltage_c  V   C phase voltage
00186     wem3080t_current_c  A   C phase current
00187     wem3080t_power_c    W   C phase active power
00188     wem3080t_importenergy_c kWh C phase import energy
00189     wem3080t_exportgrid_c   kWh C phase export energy
00190     wem3080t_frequency_c    kWh C phase frequency
00191     wem3080t_pf_c   kWh C phase power factor
00192      */
00193
00194     enum client_id {
00195         ID_C1,
00196         ID_C2,
00197         ID_C3,
00198     };
00199
00200     enum energy_state {
00201         E_INIT,
00202         E_RUN,
00203         E_WAIT,
00204         E_IDLE,
00205         E_STOP,
00206         E_LAST,
00207     };
00208
00209     enum running_state {
00210         R_INIT,
00211         R_FLOAT,
00212         R_SLEEP,
00213         R_RUN,
00214         R_IDLE,
00215         R_LAST,
00216     };
00217
00218     enum iammeter_phase {
00219         PHASE_A,
00220         PHASE_B,
00221         PHASE_C,
00222         PHASE_LAST,
00223     };
00224
00225     enum iammeter_id {
00226         IA_VOLTAGE,
00227         IA_CURRENT,
00228         IA_POWER,
00229         IA_IMPORT,
00230         IA_EXPORT,
00231         IA_FREQ,
00232         IA_PF,
00233         IA_LAST,
00234     };
00235
00236     enum mqtt_vars {
00237         V_FCCM,
00238         V_FBEKW,
00239         V_FRUNT,
00240         V_FBAMPS,
00241         V_FBV,
00242         V_FLO,
00243         V_FSO,
00244         V_FACE,
00245         V_BEN,
00246         V_PWA,
00247         V_PAMPS,
00248         V_PVOLTS,
00249         V_FLAST,
00250         V_HDCSW,
00251         V_HACSW,
00252         V_HSHUT,
00253         V_HMODE,
00254         V_HCON0,
00255         V_HCON1,
00256         V_HCON2,
00257         V_HCON3,
00258         V_HCON4,
00259         V_HCON5,
00260         V_HCON6,
00261         V_HCON7,
00262         // add other data ranges here
00263         V_DVPV,
00264         V_DPPV,
00265         V_DPBAT,
00266         V_DVBAT,
00267         V_DCMPPT,
```

```
00268         V_DPMPPT,
00269         V_DAHBAT,
00270         V_DCCMODE,
00271         V_DGTI,
00272         V_DLAST,
00273     };
00274
00275     enum sane_vars {
00276         S_FCCM,
00277         S_FBEKW,
00278         S_FRUNT,
00279         S_FBAMPS,
00280         S_FBV,
00281         S_FLO,
00282         S_FSO,
00283         S_FACE,
00284         S_BEN,
00285         S_PWA,
00286         S_PAMPS,
00287         S_PVOLTS,
00288         S_FLAST,
00289         S_HDCSW,
00290         S_HACSW,
00291         S_HSHUT,
00292         S_HMODE,
00293         // add other data ranges here
00294         S_DVPV,
00295         S_DPPV,
00296         S_DPBAT,
00297         S_DVBAT,
00298         S_DCMPPT,
00299         S_DPMPPT,
00300         S_DAHBAT,
00301         S_DCCMODE,
00302         S_DGTI,
00303         S_DLAST,
00304     };
00305
00306 #define MAX_IM_VAR  IA_LAST*PHASE_LAST
00307
00308 #define L1_P    IA_POWER
00309 #define L2_P    L1_P+IA_LAST
00310 #define L3_P    L2_P+IA_LAST
00311
00312     struct link_type {
00313         volatile uint32_t iammeter_error, iammeter_count;
00314         volatile uint32_t mqtt_error, mqtt_count;
00315         volatile uint32_t shutdown;
00316     };
00317
00318     struct mode_type {
00319         volatile double error, target, total_system, gti_dumpload, pv_bias, dl_mqtt_max, off_grid,
    sequence;
00320         volatile bool mode, in_pid_control, con0, con1, con2, con3, con4, con5, con6, con7, no_float,
    data_error, bat_crit;
00321         volatile uint32_t mode_tmr;
00322         volatile struct SPid pid;
00323         volatile enum energy_state E;
00324         volatile enum running_state R;
00325     };
00326
00327     struct energy_type {
00328         volatile double print_vars[MAX_IM_VAR];
00329         volatile double im_vars[IA_LAST][PHASE_LAST];
00330         volatile double mvar[V_DLAST + 1];
00331         volatile bool once_gti, once_ac, iammeter, fm80, dumpload, homeassistant, once_gti_zero;
00332         volatile double gti_low_adj, ac_low_adj, dl_excess_adj;
00333         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;
00334         volatile uint32_t speed_go, im_delay, im_display, gti_delay;
00335         volatile int32_t rc, sane;
00336         volatile uint32_t ten_sec_clock, log_spam, log_time_reset;
00337         pthread_mutex_t ha_lock;
00338         struct mode_type mode;
00339         struct link_type link;
00340         MQTTClient client_p, client_sd, client_ha;
00341     };
00342
00343     extern struct energy_type E;
00344     extern struct ha_flag_type ha_flag_vars_ss;
00345     extern FILE* fout;
00346
00347     void timer_callback(int32_t);
00348     void connlost(void *, char *);
00349
00350     void ramp_up_gti(MQTTClient, bool, bool);
00351     void ramp_up_ac(MQTTClient, bool);
```

```
00352     void ramp_down_gti(MQTTClient, bool);
00353     void ramp_down_ac(MQTTClient, bool);
00354     void ha_ac_off(void);
00355     void ha_ac_on(void);
00356     void ha_dc_off(void);
00357     void ha_dc_on(void);
00358 
00359     size_t iammeter_write_callback(char *, size_t, size_t, void *);
00360     void iammeter_read(void);
00361     void print_im_vars(void);
00362     void print_mvar_vars(void);
00363 
00364     bool sanity_check(void);
00365     char * log_time(bool);
00366     bool sync_ha(void);
00367     bool log_timer(void);
00368 
00369 #ifdef __cplusplus
00370 }
00371 #endif
00372 
00373 #endif /* BMC_H */
00374 
```

4.5 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     extern FILE* fout;
00018 
00019 #ifdef __cplusplus
00020 }
00021 #endif
00022 
00023 #endif /* HTTP_VARS_H */
00024 
```

4.6 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
```

4.7 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     250
00020 #define GTI_TOKEN_DELAY 300
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
```

4.8 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