HA Energy

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 files with brief descriptions:

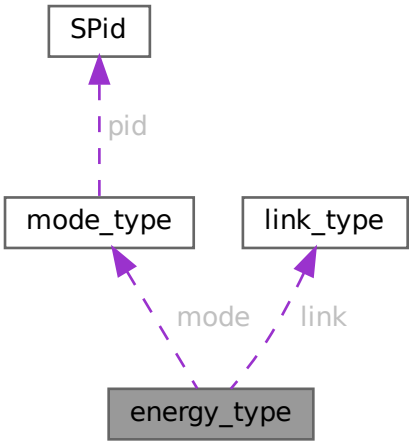
Chapter 3

Data Structure Documentation

3.1 energy_type Struct Reference

`#include <energy.h>`

Collaboration diagram for energy_type:



Data Fields

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

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

3.1.1 Field Documentation

3.1.1.1 ac_low_adj

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

3.1.1.2 ac_mismatch

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

3.1.1.3 ac_sw_on

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

3.1.1.4 ac_sw_status

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

3.1.1.5 client_ha

```
MQTTClient energy_type::client_ha
```

3.1.1.6 client_p

```
MQTTClient energy_type::client_p
```

3.1.1.7 client_sd

```
MQTTClient energy_type::client_sd
```

3.1.1.8 dc_mismatch

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

3.1.1.9 dl_excess

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

3.1.1.10 dl_excess_adj

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

3.1.1.11 dumpload

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

3.1.1.12 fm80

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

3.1.1.13 gti_delay

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

3.1.1.14 gti_low_adj

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

3.1.1.15 gti_sw_on

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

3.1.1.16 gti_sw_status

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

3.1.1.17 ha_lock

```
pthread_mutex_t energy_type::ha_lock
```

3.1.1.18 homeassistant

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

3.1.1.19 iammeter

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

3.1.1.20 im_delay

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

3.1.1.21 im_display

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

3.1.1.22 im_vars

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

3.1.1.23 link

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

3.1.1.24 log_spam

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

3.1.1.25 log_time_reset

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

3.1.1.26 mode

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

3.1.1.27 mode_mismatch

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

3.1.1.28 mvar

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

3.1.1.29 once_ac

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

3.1.1.30 once_gti

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

3.1.1.31 once_gti_zero

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

3.1.1.32 print_vars

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

3.1.1.33 rc

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

3.1.1.34 sane

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

3.1.1.35 solar_mode

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

3.1.1.36 solar_shutdown

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

3.1.1.37 speed_go

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

3.1.1.38 startup

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

3.1.1.39 ten_sec_clock

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

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

- ha_energy/energy.h

3.2 ha_flag_type Struct Reference

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

Data Fields

- volatile MQTTClient_deliveryToken deliveredtoken
- volatile MQTTClient_deliveryToken receivedtoken
- volatile bool runner
- volatile bool rec_ok
- int32_t ha_id
- volatile int32_t var_update
- volatile int32_t energy_mode

3.2.1 Field Documentation

3.2.1.1 deliveredtoken

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

3.2.1.2 energy_mode

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

3.2.1.3 ha_id

```
int32_t ha_flag_type::ha_id
```

3.2.1.4 rec_ok

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

3.2.1.5 receivedtoken

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

3.2.1.6 runner

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

3.2.1.7 var_update

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

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

- ha_energy/mqtt_rec.h

3.3 link_type Struct Reference

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

Data Fields

- volatile uint32_t iammeter_error
- volatile uint32_t iammeter_count
- volatile uint32_t mqtt_error
- volatile uint32_t mqtt_count
- volatile uint32_t shutdown

3.3.1 Field Documentation

3.3.1.1 iammeter_count

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

3.3.1.2 iammeter_error

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

3.3.1.3 mqtt_count

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

3.3.1.4 mqtt_error

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

3.3.1.5 shutdown

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

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

- ha_energy/energy.h

3.4 local_type Struct Reference

Data Fields

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

3.4.1 Field Documentation

3.4.1.1 ac_weight

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

3.4.1.2 bat_c_std_dev

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

3.4.1.3 bat_current

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

3.4.1.4 bat_voltage

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

3.4.1.5 batc_std_dev

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

3.4.1.6 coef

```
double local_type::coef
```

3.4.1.7 gti_weight

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

3.4.1.8 pv_voltage

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

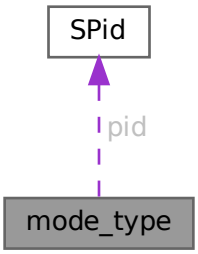
The documentation for this struct was generated from the following file:

- ha_energy/bsoc.c

3.5 mode_type Struct Reference

`#include <energy.h>`

Collaboration diagram for mode_type:



Data Fields

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

3.5.1 Field Documentation

3.5.1.1 bat_crit

`volatile bool mode_type::bat_crit`

3.5.1.2 con0

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

3.5.1.3 con1

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

3.5.1.4 con2

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

3.5.1.5 con3

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

3.5.1.6 con4

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

3.5.1.7 con5

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

3.5.1.8 con6

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

3.5.1.9 con7

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

3.5.1.10 data_error

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

3.5.1.11 dl_mqtt_max

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

3.5.1.12 E

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

3.5.1.13 error

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

3.5.1.14 gti_dumpload

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

3.5.1.15 in_pid_control

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

3.5.1.16 mode

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

3.5.1.17 mode_tmr

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

3.5.1.18 no_float

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

3.5.1.19 off_grid

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

3.5.1.20 pid

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

3.5.1.21 pv_bias

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

3.5.1.22 R

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

3.5.1.23 sequence

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

3.5.1.24 target

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

3.5.1.25 total_system

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

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

- ha_energy/energy.h

3.6 SPid Struct Reference

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

Data Fields

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

3.6.1 Field Documentation

3.6.1.1 dGain

```
double SPid::dGain
```

3.6.1.2 dState

```
double SPid::dState
```

3.6.1.3 iGain

```
double SPid::iGain
```

3.6.1.4 iMax

```
double SPid::iMax
```

3.6.1.5 iMin

```
double SPid::iMin
```

3.6.1.6 iState

```
double SPid::iState
```

3.6.1.7 pGain

```
double SPid::pGain
```

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

- ha_energy/pid.h

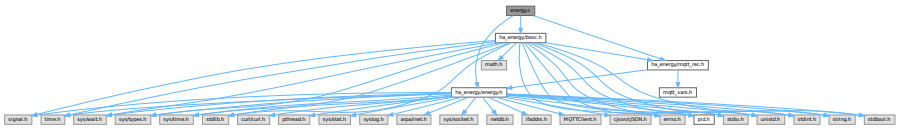
Chapter 4

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:



Macros

- #define _DEFAULT_SOURCE

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 Macro Definition Documentation

4.1.1.1 _DEFAULT_SOURCE

```
#define _DEFAULT_SOURCE
```

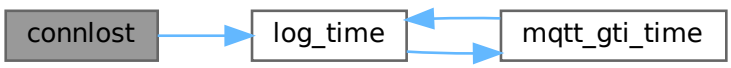
4.1.2 Function Documentation

4.1.2.1 connlost()

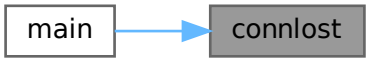
```
void connlost (
            void * context,
            char * cause)
```

```
00298 {
00299     struct ha_flag_type *ha_flag = context;
00300     int32_t id_num;
00301
00302     // bug-out if no context variables passed to callback
00303     if (context == NULL) {
00304         id_num = -1;
00305     } else {
00306         id_num = ha_flag->ha_id;
00307     }
00308     fprintf(fout, "\n%s Connection lost, exit ha_energy program\n", log_time(false));
00309     fprintf(fout, "%s     cause: %s, %d\n", log_time(false), cause, id_num);
00310     fprintf(fout, "%sDAEMON failure  LOG Version %s : MQTT Version %s\n", log_time(false),
    LOG_VERSION, MQTT_VERSION);
00311     fflush(fout);
00312     exit(EXIT_FAILURE);
00313 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

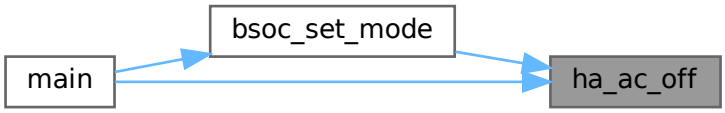


4.1.2.2 ha_ac_off()

```
void ha_ac_off (
            void )
00947 {
00948     mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
00949     E.ac_sw_status = false;
00950 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

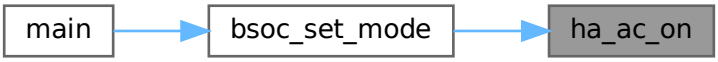


4.1.2.3 ha_ac_on()

```
void ha_ac_on (
            void )
00953 {
00954     mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00955     E.ac_sw_status = true;
00956 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

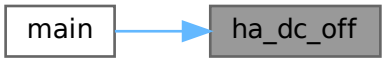


4.1.2.4 ha_dc_off()

```
void ha_dc_off (
              void )
00962 {
00963     mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
00964     E.gti_sw_status = false;
00965 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.1.2.5 ha_dc_on()

```
void ha_dc_on (
            void )
```

```
00968 {
00969     mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00970     E.gti_sw_status = true;
00971 }
```

Here is the call graph for this function:

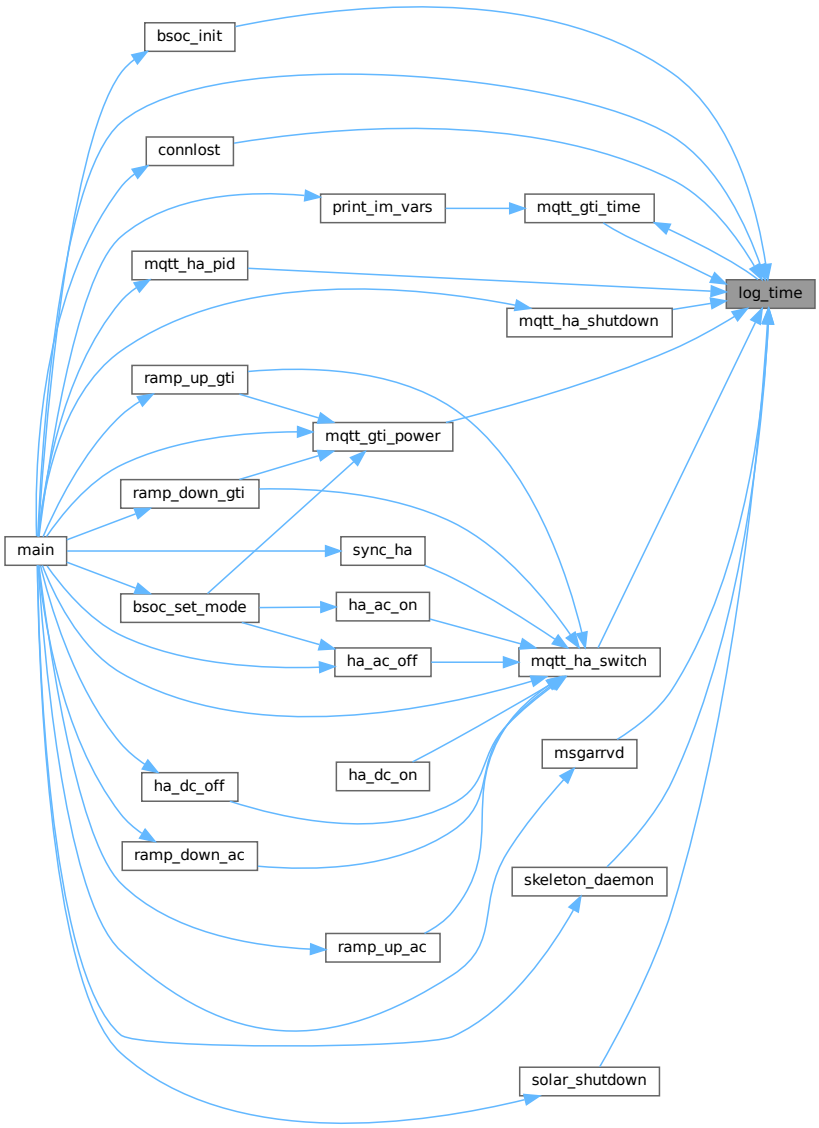
4.1.2.6 log_time()

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

```
01032 {
01033     static char time_log[RBUF_SIZ] = {0};
01034     static uint32_t len = 0, sync_time = TIME_SYNC_SEC - 1;
01035     time_t rawtime_log;
01036
01037     tzset();
01038     timezone = 0;
01039     daylight = 0;
01040     time(&rawtime_log);
01041     if (sync_time++ > TIME_SYNC_SEC) {
01042         sync_time = 0;
01043         snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
      time commands
01044         mqtt_gti_time(E.client_p, TOPIC_P, time_log);
01045     }
01046
01047     sprintf(time_log, "%s", ctime(&rawtime_log));
01048     len = strlen(time_log);
01049     time_log[len - 1] = 0; // munge out the return character
01050     if (log) {
01051         fprintf(fout, "%s ", time_log);
01052         fflush(fout);
01053     }
01054
01055     return time_log;
01056 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

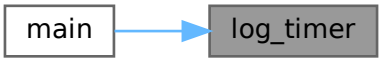


4.1.2.7 log_timer()

```
bool log_timer (
            void )
```

```
01091 {
01092     bool itstime = false;
01093
01094     if (E.log_spam < LOW_LOG_SPAM) {
01095         E.log_time_reset = 0;
01096         itstime = true;
01097     }
01098     if (E.log_time_reset > RESET_LOG_SPAM) {
01099         E.log_spam = 0;
01100         itstime = true;
01101     }
01102     return itstime;
01103 }
```

Here is the caller graph for this function:



4.1.2.8 main()

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

```
00322 {
00323     struct itimerval new_timer = {
00324         .it_value.tv_sec = CMD_SEC,
00325         .it_value.tv_usec = 0,
00326         .it_interval.tv_sec = CMD_SEC,
00327         .it_interval.tv_usec = 0,
00328     };
00329     struct itimerval old_timer;
00330     time_t rawtime;
00331     MQTTClient_connectOptions conn_opts_p = MQTTClient_connectOptions_initializer,
00332         conn_opts_sd = MQTTClient_connectOptions_initializer,
00333         conn_opts_ha = MQTTClient_connectOptions_initializer;
00334     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00335     MQTTClient_deliveryToken token;
00336     char hname[256], *hname_ptr = hname;
00337     size_t hname_len = 12;
00338
00339     gethostname(hname, hname_len);
00340     hname[12] = 0;
00341     printf("\r\n  LOG Version %s : MQTT Version %s : Host Name %s\r\n", LOG_VERSION, MQTT_VERSION,
      hname);
00342     showIP();
00343     skeleton_daemon();
00344
00345     while (true) {
00346         switch (E.mode.E) {
00347         case E_INIT:
00348
00349 #ifdef LOG_TO_FILE
00350             fout = fopen(LOG_TO_FILE, "a");
00351             if (fout == NULL) {
00352                 fout = fopen(LOG_TO_FILE_ALT, "a");
00353                 if (fout == NULL) {
00354                     fout = stdout;
00355                     printf("\r\n%s Unable to open LOG file %s \r\n", log_time(false),
      LOG_TO_FILE_ALT);
00356                 }
00357             }
00358 #else
00359             fout = stdout;
00360 #endif
00361             fprintf(fout, "\r\n%s LOG Version %s : MQTT Version %s\r\n", log_time(false), LOG_VERSION,
      MQTT_VERSION);
00362             fflush(fout);
00363
00364             if (!bsoc_init()) {
00365                 fprintf(fout, "\r\n%s bsoc_init failure \r\n", log_time(false));
00366                 fflush(fout);
00367                 exit(EXIT_FAILURE);
00368             }
00369             /*
00370              * set the timer for MQTT publishing sample speed
00371              * CMD_SEC         10
00372              */
00373             setitimer(ITIMER_REAL, &new_timer, &old_timer);
00374             signal(SIGALRM, timer_callback);
00375
```

```
00376             if (strncmp(hname, TNAME, 6) == 0) {
00377                 MQTTClient_create(&E.client_p, LADDRESS, CLIENTID1,
00378                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00379                 conn_opts_p.keepAliveInterval = 20;
00380                 conn_opts_p.cleansession = 1;
00381                 hname_ptr = LADDRESS;
00382             } else {
00383                 MQTTClient_create(&E.client_p, ADDRESS, CLIENTID1,
00384                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00385                 conn_opts_p.keepAliveInterval = 20;
00386                 conn_opts_p.cleansession = 1;
00387                 hname_ptr = ADDRESS;
00388             }
00389 
00390             fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID1);
00391             fflush(fout);
00392             MQTTClient_setCallbacks(E.client_p, &ha_flag_vars_ss, connlost, msgarrvd, delivered);
00393             if ((E.rc = MQTTClient_connect(E.client_p, &conn_opts_p)) != MQTTCLIENT_SUCCESS) {
00394                 fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
      log_time(false), E.rc, hname_ptr, CLIENTID1);
00395                 fflush(fout);
00396                 pthread_mutex_destroy(&E.ha_lock);
00397                 exit(EXIT_FAILURE);
00398             }
00399 
00400             if (strncmp(hname, TNAME, 6) == 0) {
00401                 MQTTClient_create(&E.client_sd, LADDRESS, CLIENTID2,
00402                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00403                 conn_opts_sd.keepAliveInterval = 20;
00404                 conn_opts_sd.cleansession = 1;
00405                 hname_ptr = LADDRESS;
00406             } else {
00407                 MQTTClient_create(&E.client_sd, ADDRESS, CLIENTID2,
00408                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00409                 conn_opts_sd.keepAliveInterval = 20;
00410                 conn_opts_sd.cleansession = 1;
00411                 hname_ptr = ADDRESS;
00412             }
00413 
00414             fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID2);
00415             fflush(fout);
00416             MQTTClient_setCallbacks(E.client_sd, &ha_flag_vars_sd, connlost, msgarrvd, delivered);
00417             if ((E.rc = MQTTClient_connect(E.client_sd, &conn_opts_sd)) != MQTTCLIENT_SUCCESS) {
00418                 fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
      log_time(false), E.rc, hname_ptr, CLIENTID2);
00419                 fflush(fout);
00420                 pthread_mutex_destroy(&E.ha_lock);
00421                 exit(EXIT_FAILURE);
00422             }
00423 
00424             /*
00425              * Home Assistant MQTT receive messages
00426              */
00427             if (strncmp(hname, TNAME, 6) == 0) {
00428                 MQTTClient_create(&E.client_ha, LADDRESS, CLIENTID3,
00429                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00430                 conn_opts_ha.keepAliveInterval = 20;
00431                 conn_opts_ha.cleansession = 1;
00432                 hname_ptr = LADDRESS;
00433             } else {
00434                 MQTTClient_create(&E.client_ha, ADDRESS, CLIENTID3,
00435                     MQTTCLIENT_PERSISTENCE_NONE, NULL);
00436                 conn_opts_ha.keepAliveInterval = 20;
00437                 conn_opts_ha.cleansession = 1;
00438                 hname_ptr = ADDRESS;
00439             }
00440 
00441             fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID3);
00442             fflush(fout);
00443             MQTTClient_setCallbacks(E.client_ha, &ha_flag_vars_ha, connlost, msgarrvd, delivered);
00444             if ((E.rc = MQTTClient_connect(E.client_ha, &conn_opts_ha)) != MQTTCLIENT_SUCCESS) {
00445                 fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
      log_time(false), E.rc, hname_ptr, CLIENTID3);
00446                 fflush(fout);
00447                 pthread_mutex_destroy(&E.ha_lock);
00448                 exit(EXIT_FAILURE);
00449             }
00450 
00451             /*
00452              * on topic received data will trigger the msgarrvd function
00453              */
00454             MQTTClient_subscribe(E.client_p, TOPIC_SS, QOS); // FM80 Q84
00455             MQTTClient_subscribe(E.client_sd, TOPIC_SD, QOS); // DUMPLOAD K42
00456             MQTTClient_subscribe(E.client_ha, TOPIC_HA, QOS); // Home Assistant Linux AMD64  and ARM64
00457 
00458             pubmsg.payload = "online";
00459             pubmsg.payloadlen = strlen("online");
```

```
00460             pubmsg.qos = QOS;
00461             pubmsg.retained = 0;
00462             ha_flag_vars_ss.deliveredtoken = 0;
00463             // notify HA we are running and controlling AC power plugs
00464             MQTTClient_publishMessage(E.client_p, TOPIC_PACA, &pubmsg, &token);
00465             MQTTClient_publishMessage(E.client_p, TOPIC_PDCA, &pubmsg, &token);
00466 
00467             // sync HA power switches
00468             mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
00469             mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
00470             mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00471             mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00472             mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
00473             mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
00474 
00475             E.ac_sw_on = true; // can be switched on once
00476             E.gti_sw_on = true; // can be switched on once
00477 
00478             /*
00479              * use libcurl to read AC power meter HTTP data
00480              * iammeter connected for split single phase monitoring and one leg GTI power exporting
00481              */
00482             iammeter_read();
00483 
00484             /*
00485              * start the main energy monitoring loop
00486              */
00487             fprintf(fout, "\r\n%s Solar Energy AC power controller\r\n", log_time(false));
00488 
00489 #ifdef FAKE_VPV
00490             fprintf(fout, "\r\n Faking dumpload PV voltage\r\n");
00491 #endif
00492             ha_flag_vars_ss.energy_mode = NORM_MODE;
00493             E.mode.E = E_WAIT;
00494             break;
00495         case E_WAIT:
00496             if (ha_flag_vars_ss.runner || E.speed_go++ > 1500000) {
00497                 E.speed_go = 0;
00498                 ha_flag_vars_ss.runner = false;
00499                 E.mode.E = E_RUN;
00500             }
00501 
00502             usleep(100);
00503             /*
00504              * main state-machine update sequence
00505              */
00506             bsoc_data_collect();
00507             if (!sanity_check()) {
00508                 fprintf(fout, "\r\n%s Sanity Check error %d %s \r\n", log_time(false), E.sane,
      mqtt_name[E.sane]);
00509                 fflush(fout);
00510             }
00511 
00512             /*
00513              * stop and restart the energy control processing
00514              * from inside the program or from a remote Home Assistant command
00515              */
00516             if (solar_shutdown()) {
00517                 if (!E.startup) {
00518                     fprintf(fout, "%s SHUTDOWN Solar Energy Control ---> \r\n", log_time(false));
00519                 }
00520                 fflush(fout);
00521                 ramp_down_gti(E.client_p, true);
00522                 usleep(100000); // wait
00523                 ramp_down_ac(E.client_p, true);
00524                 usleep(100000); // wait
00525                 ramp_down_gti(E.client_p, true);
00526                 usleep(100000); // wait
00527                 ramp_down_ac(E.client_p, true);
00528                 usleep(100000); // wait
00529                 if (!E.startup) {
00530                     fprintf(fout, "%s Completed SHUTDOWN, Press again to RESTART.\r\n",
      log_time(false));
00531                     fflush(fout);
00532                 }
00533                 fflush(fout);
00534 
00535                 uint8_t iam_delay = 0;
00536                 while (solar_shutdown()) {
00537                     mqtt_ha_shutdown(E.client_p, TOPIC_SHUTDOWN);
00538                     usleep(USEC_SEC); // wait
00539                     if ((int32_t) E.mvar[V_HACSW]) {
00540                         ha_ac_off();
00541                     }
00542                     if ((int32_t) E.mvar[V_HDCSW]) {
00543                         ha_dc_off();
00544                     }
```

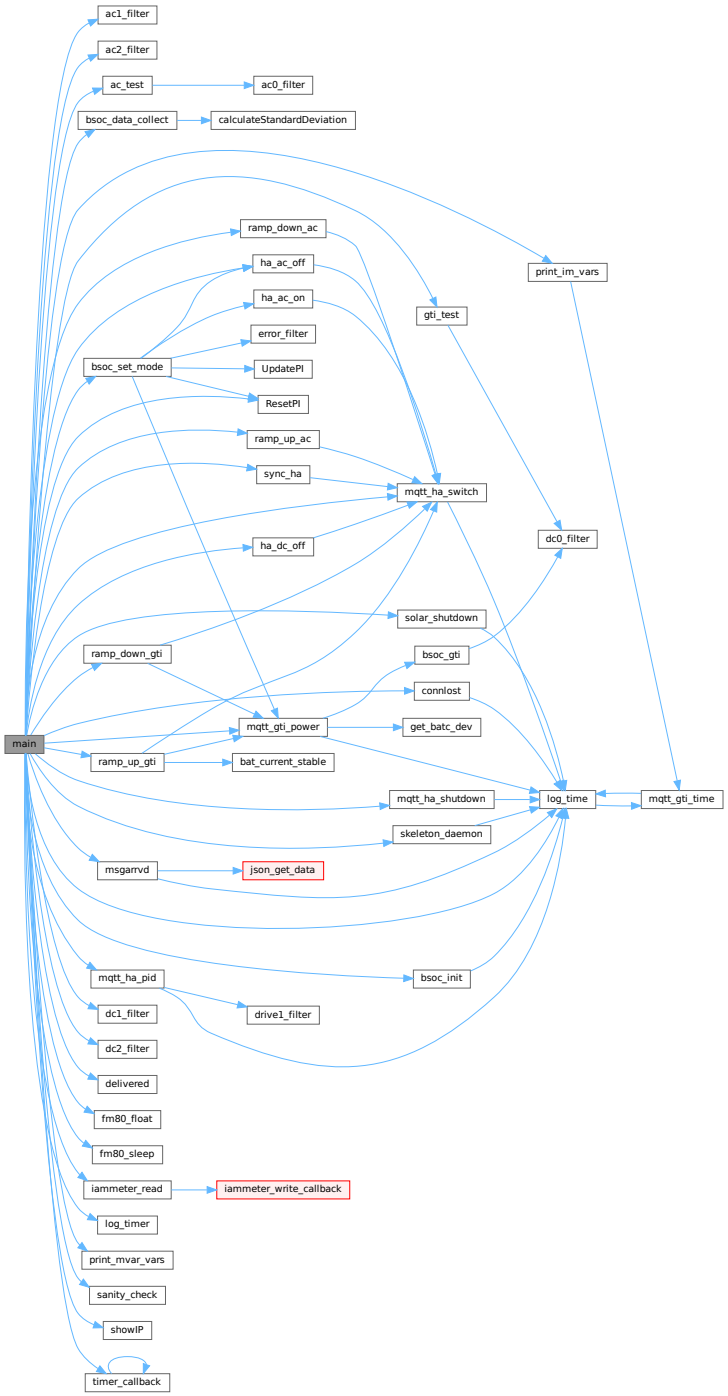
```
00545                     if ((iam_delay++ > IAM_DELAY) && E.link.shutdown) {
00546                         E.fm80 = true;
00547                         E.dumpload = true;
00548                         E.iammeter = true;
00549                         E.homeassistant = true;
00550                     }
00551                 }
00552                 E.link.shutdown = 0;
00553                 fprintf(fout, "%s RESTART Solar Energy Control\r\n", log_time(false));
00554                 fflush(fout);
00555                 bsoc_set_mode(E.mode.pv_bias, true, true);
00556                 E.dl_excess = true;
00557                 mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 1); // zero power at startup
00558                 E.dl_excess = false;
00559 #ifdef AUTO_CHARGE
00560                 mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00561 #endif
00562                 usleep(100000); // wait
00563                 E.gti_sw_status = true;
00564                 ResetPI(&E.mode.pid);
00565                 ha_flag_vars_ss.runner = true;
00566                 E.fm80 = true;
00567                 E.dumpload = true;
00568                 E.iammeter = true;
00569                 E.homeassistant = true;
00570                 E.mode.in_pid_control = false; // shutdown auto energy control
00571                 E.mode.R = R_INIT;
00572             }
00573             if (ha_flag_vars_ss.receivedtoken) {
00574                 ha_flag_vars_ss.receivedtoken = false;
00575             }
00576             if (ha_flag_vars_sd.receivedtoken) {
00577                 ha_flag_vars_sd.receivedtoken = false;
00578             }
00579             break;
00580         case E_RUN:
00581             usleep(100);
00582             switch (E.mode.R) {
00583             case R_INIT:
00584                 E.once_ac = true;
00585                 E.once_gti = true;
00586                 E.ac_sw_on = true;
00587                 E.gti_sw_on = true;
00588                 E.mode.R = R_RUN;
00589                 E.mode.no_float = true;
00590                 break;
00591             case R_FLOAT:
00592                 if (E.mode.no_float) {
00593                     E.once_ac = true;
00594                     E.once_gti = true;
00595                     E.ac_sw_on = true;
00596                     E.gti_sw_on = true;
00597                     E.gti_sw_status = false;
00598                     E.ac_sw_status = false;
00599                     E.mode.no_float = false;
00600                 }
00601                 if (!E.gti_sw_status) {
00602                     if (gti_test() > MIN_BAT_KW_GTI_HI) {
00603                         mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00604                         E.gti_sw_status = true;
00605                         fprintf(fout, "%s R_FLOAT DC switch true \r\n", log_time(false));
00606                     }
00607                 }
00608                 usleep(100000); // wait
00609                 if (!E.ac_sw_status) {
00610                     if (ac_test() > MIN_BAT_KW_AC_HI) {
00611                         mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00612                         E.ac_sw_status = true;
00613                         fprintf(fout, "%s R_FLOAT AC switch true \r\n", log_time(false));
00614                     }
00615                 }
00616                 E.mode.pv_bias = PV_BIAS;
00617                 fm80_float(true);
00618                 break;
00619             case R_RUN:
00620             default:
00621                 E.mode.R = R_RUN;
00622                 E.mode.no_float = true;
00623                 break;
00624             }
00625             /*
00626              * main state-machine update sequence and control logic
00627              */
00628             /*
00629              * check for idle/data errors flags from sensors and HA
00630              */
00631             if (!E.mode.data_error) {
```

```
00632                 bsoc_set_mode(E.mode.pv_bias, true, false);
00633                 if (E.gti_delay++ >= GTI_DELAY) {
00634                     char gti_str[SBUF_SIZ];
00635                     int32_t error_drive;
00636 
00637                     /*
00638                      * reset the control mode from simple switched power to PID control
00639                      */
00640                     if (!E.mode.in_pid_control) {
00641                         mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00642                         E.gti_sw_status = true;
00643                         usleep(100000); // wait
00644                         mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00645                         E.ac_sw_status = true;
00646                         E.mode.pv_bias = PV_BIAS;
00647                         fprintf(fout, "%s in_pid_mode AC/DC switch true \r\n", log_time(false));
00648                         fm80_float(true);
00649                     } else {
00650                         if (!fm80_float(true)) {
00651                             E.mode.pv_bias = (int32_t) E.mode.error - PV_BIAS;
00652                         }
00653                     }
00654                     /*
00655                      * use PID style set-point error correction
00656                      */
00657                     E.mode.in_pid_control = true;
00658                     E.gti_delay = 0;
00659                     /*
00660                      * adjust power balance if battery charging energy is low
00661                      */
00662                     if (E.mvar[V_DPBAT] > PV_DL_BIAS_RATE) {
00663                         error_drive = (int32_t) E.mode.error - E.mode.pv_bias; // PI feedback control
      signal
00664                     } else {
00665                         error_drive = (int32_t) E.mode.error - PV_BIAS_RATE;
00666                     }
00667                     /*
00668                      * when main battery is in float, crank-up the power draw from the solar panels
00669                      */
00670                     if (fm80_float(true)) {
00671                         error_drive = (int32_t) (E.mode.error + PV_BIAS);
00672                     }
00673                     /*
00674                      * don't drive to zero power
00675                      */
00676                     if (error_drive < 0) {
00677                         error_drive = PV_BIAS_LOW; // control wide power swings
00678                         if (!fm80_sleep()) { // check for using sleep bias
00679                             if ((E.mvar[V_FBEKW] > MIN_BAT_KW_BSOC_SLP) && (E.mvar[V_PWA] >
      PWA_SLEEP)) {
00680                                 error_drive = PV_BIAS_SLEEP; // use higher power when we still have
      sun for better inverter efficiency
00681                             }
00682                         }
00683                     }
00684 
00685                     /*
00686                      * reduce charging/diversion power to safe PS limits
00687                      */
00688                     if (E.mode.dl_mqtt_max > PV_DL_MPTT_MAX) {
00689                         if (!E.dl_excess) {
00690                             error_drive = PV_DL_MPTT_IDLE;
00691                         } else {
00692                             if (E.mode.dl_mqtt_max > PV_DL_MPTT_EXCESS) {
00693                                 error_drive = PV_DL_MPTT_IDLE;
00694                             }
00695                         }
00696                     } else {
00697                         if (E.dl_excess) {
00698                             error_drive = PV_DL_EXCESS + E.dl_excess_adj;
00699                         }
00700                     }
00701 
00702                     /*
00703                      * shutdown GTI power at low DL battery Ah or Voltage
00704                      */
00705                     if ((E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) || (E.mvar[V_DVBAT] < PV_DL_B_V_LOW)) {
00706                         error_drive = PV_BIAS_ZERO;
00707                     }
00708 
00709                     snprintf(gti_str, SBUF_SIZ - 1, "V%04dX", error_drive); // format for dumpload
      controller gti power commands
00710                     mqtt_gti_power(E.client_p, TOPIC_P, gti_str, 2);
00711                 }
00712 
00713 #ifndef  FAKE_VPV
00714                 if (fm80_float(true) || ((ac1_filter(E.mvar[V_BEN]) > BAL_MAX_ENERGY_AC) && (ac_test()
```

```
    > MIN_BAT_KW_AC_HI))) {
00715                       ramp_up_ac(E.client_p, E.ac_sw_on); // use once control
00716 #ifdef PSW_DEBUG
00717                       fprintf(fout, "%s MIN_BAT_KW_AC_HI AC switch %d \r\n", log_time(false),
    E.ac_sw_on);
00718 #endif
00719                       E.ac_sw_on = false; // once flag
00720                   }
00721 #endif
00722                   if (((ac2_filter(E.mvar[V_BEN]) < BAL_MIN_ENERGY_AC) || ((ac_test() <
    (MIN_BAT_KW_AC_LO + E.ac_low_adj))))) {
00723                       if (!fm80_float(true)) {
00724                           ramp_down_ac(E.client_p, E.ac_sw_on);
00725                           if (log_timer()) {
00726                               fprintf(fout, "%s RAMP DOWN AC, MIN_BAT_KW_AC_LO AC switch %d \r\n",
    log_time(false), E.ac_sw_on);
00727                           }
00728                       }
00729                       E.ac_sw_on = true;
00730                   }
00731
00732
00733                   /*
00734                    * Dump Load Excess testing
00735                    * send excess power into the home power grid taking care not to export energy to the
    utility grid
00736                    */
00737                   if (((dc1_filter(E.mvar[V_BEN]) > BAL_MAX_ENERGY_GTI) && (gti_test() >
    MIN_BAT_KW_GTI_HI)) || E.dl_excess) {
00738 #ifndef  FAKE_VPV
00739 #ifdef B_DLE_DEBUG
00740                       if (E.dl_excess) {
00741                           fprintf(fout, "%s DL excess ramp_up_gti, DC switch %d\r\n", log_time(false),
    E.gti_sw_on);
00742                       }
00743 #endif
00744                       ramp_up_gti(E.client_p, E.gti_sw_on, E.dl_excess);
00745                       if (log_timer()) {
00746                           fprintf(fout, "%s RAMP DOWN DC, MIN_BAT_KW_GTI_HI DC switch %d \r\n",
    log_time(false), E.gti_sw_on);
00747                       }
00748                       E.gti_sw_on = false; // once flag
00749 #endif
00750                   } else {
00751                       if ((dc2_filter(E.mvar[V_BEN]) < BAL_MIN_ENERGY_GTI) || (gti_test() <
    (MIN_BAT_KW_GTI_LO + E.gti_low_adj))) {
00752                           if (!E.dl_excess) {
00753                               if (log_timer()) {
00754                                   ramp_down_gti(E.client_p, true);
00755 #ifdef PSW_DEBUG
00756                                   fprintf(fout, "%s MIN_BAT_KW_GTI_LO DC switch %d \r\n",
    log_time(false), E.gti_sw_on);
00757 #endif
00758                               }
00759                               E.gti_sw_on = true;
00760                           }
00761                       }
00762                   }
00763               };
00764
00765 #ifdef B_ADJ_DEBUG
00766               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);
00767 #endif
00768 #ifdef B_DLE_DEBUG
00769               if (E.dl_excess) {
00770                   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);
00771               }
00772 #endif
00773
00774               time(&rawtime);
00775
00776               if (E.im_delay++ >= IM_DELAY) {
00777                   E.im_delay = 0;
00778                   iammeter_read();
00779               }
00780               if (E.im_display++ >= IM_DISPLAY) {
00781                   char buffer[SYSLOG_SIZ];
00782                   uint32_t len;
00783
00784                   E.im_display = 0;
00785                   mqtt_ha_pid(E.client_p, TOPIC_PPID);
00786                   if (!(E.fm80 && E.dumpload && E.iammeter)) {
00787                       if (!E.iammeter) {
00788                           E.link.iammeter_error++;
00789                       } else {
```

```
00790                         E.link.mqtt_error++;
00791                     }
00792                     E.link.shutdown++;
00793                     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,
00794                         E.link.mqtt_count, E.link.mqtt_error, E.link.iammeter_count,
      E.link.iammeter_error);
00795                     fflush(fout);
00796                     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,
00797                         E.link.mqtt_count, E.link.mqtt_error, E.link.iammeter_count,
      E.link.iammeter_error);
00798                     syslog(LOG_NOTICE, buffer);
00799                     mqtt_ha_shutdown(E.client_p, TOPIC_SHUTDOWN);
00800                     E.mode.data_error = true;
00801                 } else {
00802                     E.mode.data_error = false;
00803                     E.link.shutdown = 0;
00804                 }
00805                 snprintf(buffer, RBUF_SIZ - 1, "%s", ctime(&rawtime));
00806                 len = strlen(buffer);
00807                 buffer[len - 1] = 0; // munge out the return character
00808                 fprintf(fout, "%s ", buffer);
00809                 fflush(fout);
00810                 E.fm80 = false;
00811                 E.dumpload = false;
00812                 E.homeassistant = false;
00813                 E.iammeter = false;
00814                 sync_ha();
00815                 print_im_vars();
00816                 print_mvar_vars();
00817                 fprintf(fout, "%s\r", ctime(&rawtime));
00818             }
00819             E.mode.E = E_WAIT;
00820             fflush(fout);
00821             if (E.mode.con6) {
00822                 E.mode.R = R_IDLE;
00823             }
00824             if (E.mode.con7) {
00825                 E.mode.E = E_STOP;
00826             }
00827             break;
00828         case E_STOP:
00829         default:
00830             fflush(fout);
00831             fprintf(fout, "\r\n%s HA Energy stopped and exited.\r\n", log_time(false));
00832             fflush(fout);
00833             return 0;
00834             break;
00835         }
00836     }
00837 }
```

Here is the call graph for this function:

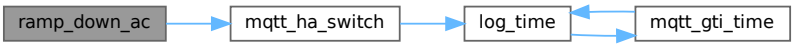


4.1.2.9 ramp_down_ac()

```
void ramp_down_ac (
            MQTTClient client_p,
            bool sw_off)
00937 {
```

```
00938     if (sw_off) {
00939         mqtt_ha_switch(client_p, TOPIC_PACC, false);
00940         E.ac_sw_status = false;
00941         usleep(500000);
00942     }
00943     E.once_ac = true;
00944 }
```

Here is the call graph for this function:

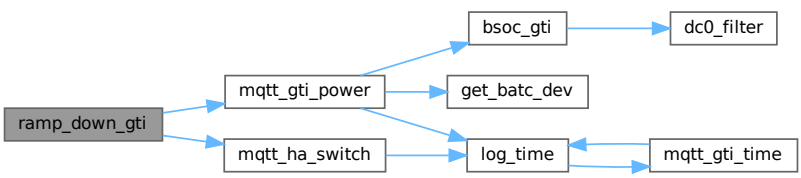


Here is the caller graph for this function:

4.1.2.10 ramp_down_gti()

```
void ramp_down_gti (
            MQTTClient client_p,
            bool sw_off)
00904 {
00905     if (sw_off) {
00906         mqtt_ha_switch(client_p, TOPIC_PDCC, false);
00907         E.once_gti_zero = true;
00908         E.gti_sw_status = false;
00909     }
00910     E.once_gti = true;
00911
00912     if (E.once_gti_zero) {
00913         mqtt_gti_power(client_p, TOPIC_P, "Z#", 7); // zero power
00914         E.once_gti_zero = false;
00915     }
00916 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

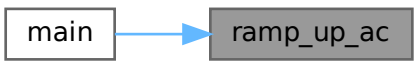
4.1.2.11 ramp_up_ac()

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

```
00922 {
00923
00924     if (start) {
00925         E.once_ac = true;
00926     }
00927
00928     if (E.once_ac) {
00929         E.once_ac = false;
00930         mqtt_ha_switch(client_p, TOPIC_PACC, true);
00931         E.ac_sw_status = true;
00932         usleep(500000); // wait for voltage to ramp
00933     }
00934 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

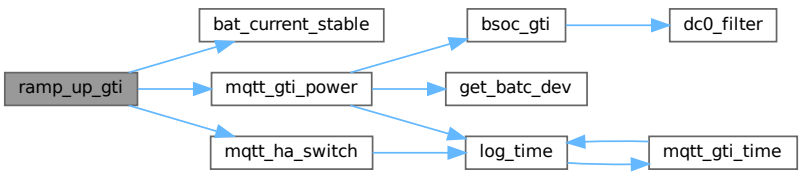


4.1.2.12 ramp_up_gti()

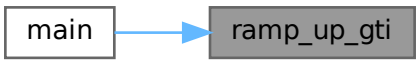
```
void ramp_up_gti (
            MQTTClient client_p,
            bool start,
            bool excess)
```

```
00843 {
00844     static uint32_t sequence = 0;
00845 
00846     if (start) {
00847         E.once_gti = true;
00848     }
00849 
00850     if (E.once_gti) {
00851         E.once_gti = false;
00852         sequence = 0;
00853         if (!excess) {
00854             mqtt_ha_switch(client_p, TOPIC_PDCC, true);
00855             E.gti_sw_status = true;
00856             usleep(500000); // wait for voltage to ramp
00857         } else {
00858             sequence = 1;
00859         }
00860     }
00861 
00862     switch (sequence) {
00863     case 4:
00864         E.once_gti_zero = true;
00865         break;
00866     case 3:
00867     case 2:
00868     case 1:
00869         E.once_gti_zero = true;
00870         if (bat_current_stable() || E.dl_excess) { // check battery current std dev, stop
      'motorboating'
00871             sequence++;
00872             if (!mqtt_gti_power(client_p, TOPIC_P, "+#", 3)) {
00873                 sequence = 0;
00874             }; // +100W power
00875         } else {
00876             usleep(500000); // wait a bit more for power to be stable
00877             sequence = 1; // do power ramps when ready
00878             if (!mqtt_gti_power(client_p, TOPIC_P, "-#", 4)) {
00879                 sequence = 0;
00880             }; // - 100W power
00881         }
00882         break;
00883     case 0:
00884         sequence++;
00885         if (E.once_gti_zero) {
00886             mqtt_gti_power(client_p, TOPIC_P, "Z#", 5); // zero power
00887             E.once_gti_zero = false;
00888         }
00889         break;
00890     default:
00891         if (E.once_gti_zero) {
00892             mqtt_gti_power(client_p, TOPIC_P, "Z#", 6); // zero power
00893             E.once_gti_zero = false;
00894         }
00895         sequence = 0;
00896         break;
00897     }
00898 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.2.13 sanity_check()

```
bool sanity_check (
             void )
```

```
00254 {
00255     if (E.mvar[V_PWA] > PWA_SANE) {
00256         E.sane = S_PWA;
00257         return false;
00258     }
00259     if (E.mvar[V_PAMPS] > PAMPS_SANE) {
00260         E.sane = S_PAMPS;
00261         return false;
00262     }
00263     if (E.mvar[V_PVOLTS] > PVOLTS_SANE) {
00264         E.sane = S_PVOLTS;
00265         return false;
00266     }
00267     if (E.mvar[V_FBAMPS] > BAMPS_SANE) {
00268         E.sane = S_FBAMPS;
00269         return false;
00270     }
00271     return true;
00272 }
```

Here is the caller graph for this function:

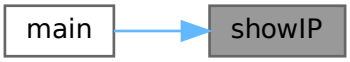
4.1.2.14 showIP()

```
void showIP (
             void )
```

```
00160 {
00161     struct ifaddrs *ifaddr, *ifa;
00162     int s;
00163     char host[NI_MAXHOST];
00164
00165     if (getifaddrs(&ifaddr) == -1) {
00166         perror("getifaddrs");
00167         exit(EXIT_FAILURE);
00168     }
```

```
00169
00170
00171     for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
00172         if (ifa->ifa_addr == NULL)
00173             continue;
00174
00175         s = getnameinfo(ifa->ifa_addr, sizeof(struct sockaddr_in), host, NI_MAXHOST, NULL, 0,
      NI_NUMERICHOST);
00176
00177         if (ifa->ifa_addr->sa_family == AF_INET) {
00178             if (s != 0) {
00179                 exit(EXIT_FAILURE);
00180             }
00181             printf("\tInterface : <%s>\n", ifa->ifa_name);
00182             printf("\t  Address : <%s>\n", host);
00183         }
00184     }
00185
00186     freeifaddrs(ifaddr);
00187 }
```

Here is the caller graph for this function:



4.1.2.15 skeleton_daemon()

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

```
00194 {
00195     pid_t pid;
00196
00197     /* Fork off the parent process */
00198     pid = fork();
00199
00200     /* An error occurred */
00201     if (pid < 0) {
00202         printf("\r\n%sDAEMON failure  LOG Version %s : MQTT Version %s\r\n", log_time(false),
      LOG_VERSION, MQTT_VERSION);
00203         exit(EXIT_FAILURE);
00204     }
00205
00206     /* Success: Let the parent terminate */
00207     if (pid > 0) {
00208         exit(EXIT_SUCCESS);
00209     }
00210
00211     /* On success: The child process becomes session leader */
00212     if (setsid() < 0) {
00213         exit(EXIT_FAILURE);
00214     }
00215
00216     /* Catch, ignore and handle signals */
00217     /*TODO: Implement a working signal handler */
00218     //    signal(SIGCHLD, SIG_IGN);
00219     //    signal(SIGHUP, SIG_IGN);
00220
00221     /* Fork off for the second time*/
00222     pid = fork();
00223
00224     /* An error occurred */
00225     if (pid < 0) {
00226         exit(EXIT_FAILURE);
00227     }
00228
00229     /* Success: Let the parent terminate */
00230     if (pid > 0) {
```

```
00231         exit(EXIT_SUCCESS);
00232     }
00233
00234     /* Set new file permissions */
00235     umask(0);
00236
00237     /* Change the working directory to the root directory */
00238     /* or another appropriated directory */
00239     chdir("/");
00240
00241     /* Close all open file descriptors */
00242     int x;
00243     for (x = sysconf(_SC_OPEN_MAX); x >= 0; x--) {
00244         close(x);
00245     }
00246
00247 }
```

Here is the call graph for this function:

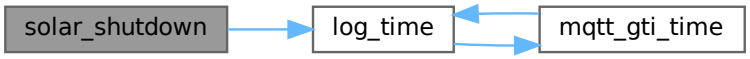
Here is the caller graph for this function:

4.1.2.16 solar_shutdown()

```
bool solar_shutdown (
             void )  [static]
00977 {
00978     static bool ret = false;
00979
00980     if (E.startup) {
00981         ret = true;
00982         E.startup = false;
00983         return ret;
00984     } else {
00985         ret = false;
00986
00987         /*
00988          * FIXME
00989          *
00990          */
00991     }
00992
00993     if (E.solar_shutdown) {
00994         ret = true;
00995     } else {
00996         ret = false;
00997     }
```

```
00998
00999     if ((E.mvar[V_FBEKW] < BAT_CRITICAL) && !E.startup) { // special case for low battery
01000         if (!E.mode.bat_crit) {
01001             ret = true;
01002 #ifdef CRITIAL_SHUTDOWN_LOG
01003             fprintf(fout, "%s Solar BATTERY CRITICAL shutdown comms check ret = %d \r\n",
      log_time(false), ret);
01004             fflush(fout);
01005 #endif
01006             E.mode.bat_crit = true;
01007             return ret;
01008         }
01009     } else {
01010         E.mode.bat_crit = false;
01011     }
01012
01013     if (E.link.shutdown >= MAX_ERROR) {
01014         ret = true;
01015         if (E.fm80 && E.dumpload && E.iammeter) {
01016             ret = false;
01017             E.link.shutdown = 0;
01018         }
01019
01020 #ifdef DEBUG_SHUTDOWN
01021         fprintf(fout, "%s Solar shutdown comms check ret = %d \r\n", log_time(false), ret);
01022         fflush(fout);
01023 #endif
01024     }
01025     return ret;
01026 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

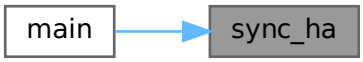
4.1.2.17 sync_ha()

```
bool sync_ha (
             void )
01062 {
01063     bool sync = false;
01064     if (E.gti_sw_status != (bool) ((int32_t) E.mvar[V_HDCSW])) {
01065         fprintf(fout, "DC_MM %d %d ", (bool) E.gti_sw_status, (bool) ((int32_t) E.mvar[V_HDCSW]));
01066         mqtt_ha_switch(E.client_p, TOPIC_PDCC, !E.gti_sw_status);
01067         E.dc_mismatch = true;
01068         fflush(fout);
01069         sync = true;
```

```
01070     } else {
01071         E.dc_mismatch = false;
01072     }
01073
01074     E.ac_sw_status = (bool) ((int32_t) E.mvar[V_HACSW]); // TEMP FIX for MISmatch errors
01075     if (E.ac_sw_status != (bool) ((int32_t) E.mvar[V_HACSW])) {
01076         fprintf(fout, "AC_MM %d %d ", (bool) E.ac_sw_status, (bool) ((int32_t) E.mvar[V_HACSW]));
01077         mqtt_ha_switch(E.client_p, TOPIC_PACC, !E.ac_sw_status);
01078         E.ac_mismatch = true;
01079         fflush(fout);
01080         sync = true;
01081     } else {
01082         E.ac_mismatch = false;
01083     }
01084     return sync;
01085 }
```

Here is the call graph for this function:

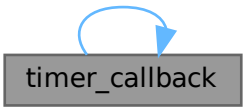
Here is the caller graph for this function:



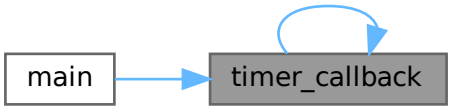
4.1.2.18 timer_callback()

```
void timer_callback (
             int32_t signum)
00283 {
00284     signal(signum, timer_callback);
00285     ha_flag_vars_ss.runner = true;
00286     E.ten_sec_clock++;
00287     E.log_spam++;
00288     E.log_time_reset++;
00289     if (E.log_spam > MAX_LOG_SPAM) {
00290         E.log_spam = 0;
00291     }
00292 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3 Variable Documentation

4.1.3.1 board_name

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

4.1.3.2 driver_name

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

4.1.3.3 E

```
struct energy_type E
00107                       {
00108     .once_gti = true,
00109     .once_ac = true,
00110     .once_gti_zero = true,
00111     .iammeter = false,
00112     .fm80 = false,
00113     .dumpload = false,
00114     .homeassistant = false,
00115     .ac_low_adj = 0.0f,
00116     .gti_low_adj = 0.0f,
00117     .ac_sw_on = true,
00118     .gti_sw_on = true,
00119     .im_delay = 0,
00120     .gti_delay = 0,
00121     .im_display = 0,
00122     .rc = 0,
```

```
00123     .speed_go = 0,
00124     .mode.pid.iMax = PV_IMAX,
00125     .mode.pid.iMin = 0.0f,
00126     .mode.pid.pGain = PV_PGAIN,
00127     .mode.pid.iGain = PV_IGAIN,
00128     .mode.mode_tmr = 0,
00129     .mode.mode = true,
00130     .mode.in_pid_control = false,
00131     .mode.dl_mqtt_max = PV_DL_MPTT_MAX,
00132     .mode.E = E_INIT,
00133     .mode.R = R_INIT,
00134     .mode.no_float = true,
00135     .mode.data_error = false,
00136     .ac_sw_status = false,
00137     .gti_sw_status = false,
00138     .solar_mode = false,
00139     .solar_shutdown = false,
00140     .mode.pv_bias = PV_BIAS_LOW,
00141     .sane = S_DLAST,
00142     .startup = true,
00143     .ac_mismatch = false,
00144     .dc_mismatch = false,
00145     .mode_mismatch = false,
00146     .link.shutdown = 0,
00147     .mode.bat_crit = false,
00148     .dl_excess = false,
00149     .dl_excess_adj = 0.0f,
00150 };
```

4.1.3.4 fout

```
FILE* fout
```

4.1.3.5 ha_flag_vars_ha

```
struct ha_flag_type ha_flag_vars_ha
```

Initial value:

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

```
00092                                       {
00093     .runner = false,
00094     .receivedtoken = false,
00095     .deliveredtoken = false,
00096     .rec_ok = false,
00097     .ha_id = HA_ID,
00098     .var_update = 0,
00099 };
```

4.1.3.6 ha_flag_vars_pc

```
struct ha_flag_type ha_flag_vars_pc
```

Initial value:

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

```
00061                                       {
00062     .runner = false,
00063     .receivedtoken = false,
00064     .deliveredtoken = false,
00065     .rec_ok = false,
00066     .ha_id = P8055_ID,
00067     .var_update = 0,
00068 };
```

4.1.3.7 ha_flag_vars_sd

```
struct ha_flag_type ha_flag_vars_sd
```

Initial value:

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

```
00082                                       {
00083     .runner = false,
00084     .receivedtoken = false,
00085     .deliveredtoken = false,
00086     .rec_ok = false,
00087     .ha_id = DUMPLOAD_ID,
00088     .var_update = 0,
00089 };
```

4.1.3.8 ha_flag_vars_ss

```
struct ha_flag_type ha_flag_vars_ss
```

Initial value:

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

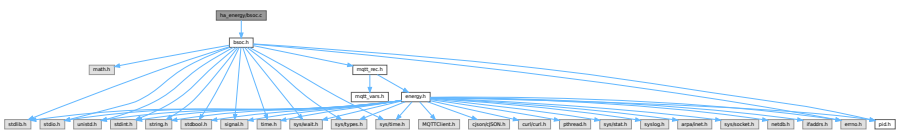
```
00071                                       {
00072     .runner = false,
00073     .receivedtoken = false,
00074     .deliveredtoken = false,
00075     .rec_ok = false,
00076     .ha_id = FM80_ID,
00077     .var_update = 0,
00078     .energy_mode = NORM_MODE,
00079 };
```

4.2 ha_energy/.dep.inc File Reference

4.3 ha_energy/bsoc.c File Reference

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

Include dependency graph for bsoc.c:



Data Structures

- struct local_type

Functions

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

Variables

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

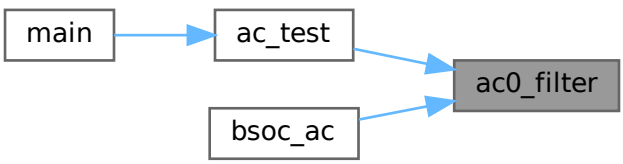
4.3.1 Function Documentation

4.3.1.1 ac0_filter()

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

```
00376 {
00377     static double accum = 0.0f;
00378     static double coef = COEFF;
00379     accum = accum - accum / coef + raw;
00380     return accum / coef;
00381 }
```

Here is the caller graph for this function:

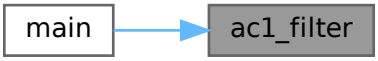


4.3.1.2 ac1_filter()

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

```
00384 {
00385     static double accum = 0.0f;
00386     static double coef = COEF;
00387     accum = accum - accum / coef + raw;
00388     return accum / coef;
00389 }
```

Here is the caller graph for this function:

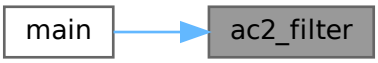


4.3.1.3 ac2_filter()

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

```
00392 {
00393     static double accum = 0.0f;
00394     static double coef = COEF;
00395     accum = accum - accum / coef + raw;
00396     return accum / coef;
00397 }
```

Here is the caller graph for this function:

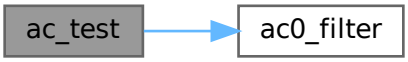


4.3.1.4 ac_test()

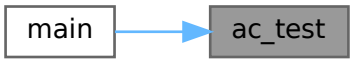
```
double ac_test (
            void )
```

```
00191 {
00192     return ac0_filter(L.ac_weight);
00193 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.1.5 bat_current_stable()

```
bool bat_current_stable (
            void )
```

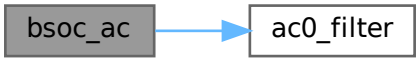
```
00240 {
00241     static double gap = 0.0f;
00242
00243     if (L.batc_std_dev <= (MAX_BATC_DEV + gap)) {
00244         gap = MAX_BATC_DEV;
00245         if (L.bat_c_std_dev[0] < BAT_C_DRAW) {
00246             return true;
00247         } else {
00248             gap = 0.0f;
00249             return false;
00250         }
00251     } else {
00252         gap = 0.0f;
00253         return false;
00254     }
00255 }
```

Here is the caller graph for this function:

4.3.1.6 bsoc_ac()

```
double bsoc_ac (
            void )
00136 {
00137
00138     return ac0_filter(L.ac_weight);
00139 };
```

Here is the call graph for this function:



4.3.1.7 bsoc_data_collect()

```
bool bsoc_data_collect (
            void )
00086 {
00087     bool ret = false;
00088     static uint32_t i = 0;
00089     // lockout threaded updates
00090     pthread_mutex_lock(&E.ha_lock); // lockout MQTT var updates
00091
00092     L.ac_weight = E.mvar[V_FBEKW];
00093     L.gti_weight = E.mvar[V_FBEKW];
00094 #ifdef FAKE_VPV // no DUMPLOAD AC charger
00095     if (E.gti_sw_on) {
00096         pv_voltage = PV_V_NOM;
00097     } else {
00098         pv_voltage = PV_V_FAKE;
00099     }
00100     E.mvar[V_DVPV] = pv_voltage;
00101 #else
00102     L.pv_voltage = E.mvar[V_DVPV];
00103 #endif
00104     L.bat_voltage = E.mvar[V_DVBAT];
00105     L.bat_current = E.mvar[V_DCMPPT];
00106     E.ac_low_adj = E.mvar[V_FSO]* -0.5f;
00107     E.gti_low_adj = E.mvar[V_FACE] * -0.5f;
00108     E.mode.dl_mqtt_max = E.mvar[V_DPMPPT];
00109
00110     pthread_mutex_unlock(&E.ha_lock); // resume remote MQTT var updates
00111
00112     if (E.ac_low_adj < -2000.0f) {
00113         E.ac_low_adj = -2000.0f;
00114     }
00115     if (E.gti_low_adj < -2000.0f) {
00116         E.gti_low_adj = -2000.0f;
00117     }
00118
00119     L.bat_c_std_dev[i++] = L.bat_current;
00120     if (i >= DEV_SIZE) {
00121         i = 0;
00122     }
00123
00124     calculateStandardDeviation(DEV_SIZE, L.bat_c_std_dev);
00125
00126 #ifdef BSOC_DEBUG
00127     fprintf(fout, "\r\nmqtt var bsoc update\r\n");
00128 #endif
00129     return ret;
00130 }
```

Here is the call graph for this function:

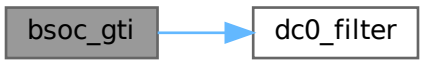
Here is the caller graph for this function:

4.3.1.8 bsoc_gti()

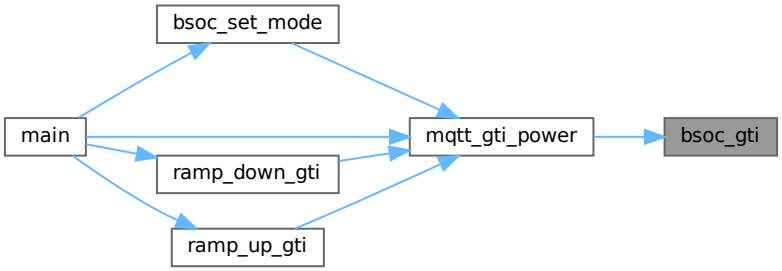
```
double bsoc_gti (
             void )
```

```
00146 {
00147 #ifdef BSOC_DEBUG
00148     fprintf(fout, "pvp %f, gweight %f, aweight %f, batv %f, batc %f\r\n", pv_voltage, gti_weight,
      ac_weight, bat_voltage, bat_current);
00149 #endif
00150     // check for 48VDC AC charger powered from the Solar battery bank AC inverter unless E.dl_excess
      is TRUE
00151     if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
00152         L.gti_weight = 0.0f; // reduce power to zero
00153     } else {
00154         if (E.dl_excess) {
00155             if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
00156                 L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
00157             } else {
00158                 L.gti_weight = 0.0f; // reduce power to zero
00159             }
00160         }
00161     }
00162
00163
00164     return dc0_filter(L.gti_weight);
00165 };
```

Here is the call graph for this function:



Here is the caller graph for this function:



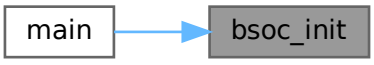
4.3.1.9 bsoc_init()

```
bool bsoc_init (
              void )
```

```
00061 {
00062     L.ac_weight = 0.0f;
00063     L.gti_weight = 0.0f;
00064     // use MUTEX locks for message passing between remote programs
00065     if (pthread_mutex_init(&E.ha_lock, NULL) != 0) {
00066         fprintf(fout, "\n%s mutex init has failed\n", log_time(false));
00067         return false;
00068     }
00069     return true;
00070 };
```

Here is the call graph for this function:

Here is the caller graph for this function:



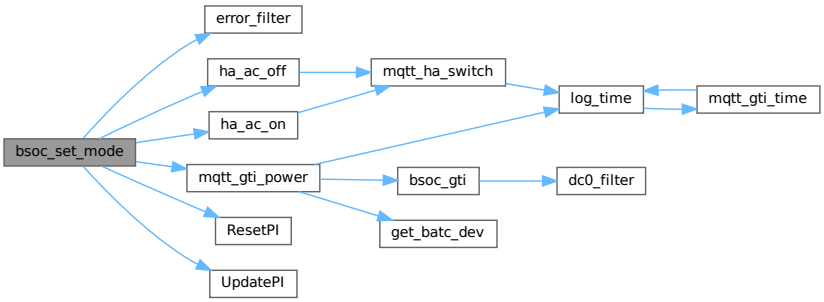
4.3.1.10 bsoc_set_mode()

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

```
00262 {
00263     static bool bsoc_mode = false;
00264     static bool bsoc_high = false, ha_ac_mode = true;
00265     static double accum = 0.0f, vpwa = 0.0f;
00266 
00267     if (init) {
00268         bsoc_mode = false;
00269         bsoc_high = false;
00270         ha_ac_mode = true;
00271         accum = 0.0f;
00272         vpwa = 0.0f;
00273         return true;
00274     }
00275     /*
00276      * running avg filter
00277      */
00278     accum = accum - accum / COEFN + E.mvar[V_PWA];
00279     vpwa = accum / COEFN;
00280 
00281     if ((vpwa >= PV_FULL_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
00282         if (!bsoc_mode) {
00283             ResetPI(&E.mode.pid);
00284         }
00285         bsoc_mode = true;
00286         bsoc_high = true;
00287         if (!ha_ac_mode) {
00288             ha_ac_on();
00289             ha_ac_mode = true;
00290         }
00291 
00292     } else {
00293         if (bsoc_high) { // turn off at min limit power
00294             if ((vpwa >= PV_MIN_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
00295                 bsoc_mode = true;
00296                 if (ha_ac_mode) {
00297                     ha_ac_off();
00298                     ha_ac_mode = false;
00299                 }
00300             } else {
00301                 bsoc_high = false;
00302                 ha_ac_mode = false;
00303             }
00304         }
00305     }
00306 
00307     E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) + E.mvar[V_DPPV]; // use as a temp variable
00308     E.mode.total_system = (E.mvar[V_FLO] - E.mode.gti_dumpload) + E.mvar[V_DPPV] +(E.print_vars[L3_P]*
    -1.0f);
00309     E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) - E.mvar[V_DPPV]; // use this value
00310 
00311     /*
00312      * look at system energy balance for power control drive
00313      */
00314     if (mode) { // add GTI power from dumpload
00315         E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + E.mode.gti_dumpload +
    PBAL_OFFSET);
00316     } else {
00317         E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + PBAL_OFFSET);
00318     }
00319 
00320     if (E.mode.error > 0.0f) {
00321         L.coef = COEF;
00322     } else {
00323         L.coef = COEFN;
00324     }
00325     E.mode.target = target;
00326     E.mode.error = round(error_filter(E.mode.error));
00327     /*
00328      * check for idle flag from HA
00329      */
00330     if (E.mode.con6) {
00331         ha_ac_mode = true;
00332         bsoc_mode = false;
00333     }
00334 
00335     /*
00336      * HA start excess button pressed
00337      */
```

```
00338     if (E.mode.con4) {
00339         E.dl_excess = true;
00340         E.mode.con4 = false;
00341     }
00342 
00343     /*
00344      * HA stop excess button pressed
00345      */
00346     if (E.mode.con5) {
00347         mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 9); // zero power at excess shutdown
00348         E.dl_excess = false;
00349         E.mode.con5 = false;
00350     }
00351 
00352     /*
00353      * DL buffer battery low set-point excess load shutdown
00354      */
00355     if (E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) {
00356         mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 10); // zero power at excess shutdown
00357         E.dl_excess = false;
00358         E.mode.con4 = false;
00359         E.mode.con5 = false;
00360     }
00361 
00362     return bsoc_mode;
00363 }
```

Here is the call graph for this function:

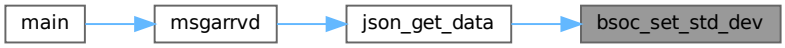


Here is the caller graph for this function:

4.3.1.11 bsoc_set_std_dev()

```
void bsoc_set_std_dev (
            const double value,
            const uint32_t i)
00076 {
00077     L.bat_c_std_dev[i] = value;
00078 }
```

Here is the caller graph for this function:



4.3.1.12 calculateStandardDeviation()

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

```
00205 {
00206     // variable to store sum of the given data
00207     double sum = 0;
00208
00209     for (int i = 0; i < N; i++) {
00210         sum += data[i];
00211     }
00212
00213     // calculating mean
00214     double mean = sum / N;
00215
00216     // temporary variable to store the summation of square
00217     // of difference between individual data items and mean
00218     double values = 0;
00219
00220     for (int i = 0; i < N; i++) {
00221         values += pow(data[i] - mean, 2);
00222     }
00223
00224     // variance is the square of standard deviation
00225     double variance = values / N;
00226
00227     // calculating standard deviation by finding square root
00228     // of variance
00229     double standardDeviation = sqrt(variance);
00230     L.batc_std_dev = standardDeviation;
00231
00232 #ifdef BSOC_DEBUG
00233     // printing standard deviation
00234     fprintf(fout, "STD DEV of Current %.2f\r\n", standardDeviation);
00235 #endif
00236     return standardDeviation;
00237 }
```

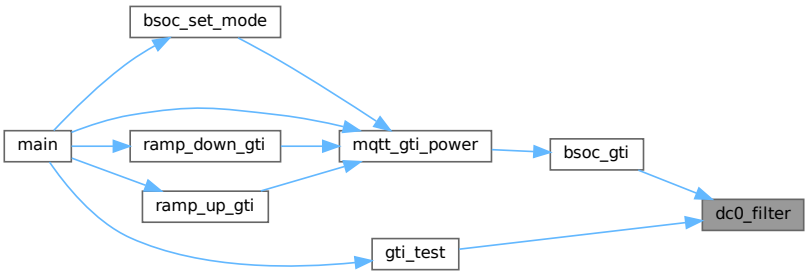
Here is the caller graph for this function:

4.3.1.13 dc0_filter()

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

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

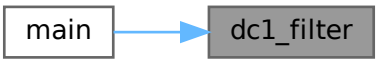
Here is the caller graph for this function:



4.3.1.14 dc1_filter()

```
double dc1_filter (
            const double raw)
00408 {
00409     static double accum = 0.0f;
00410     static double coef = COEF;
00411     accum = accum - accum / coef + raw;
00412     return accum / coef;
00413 }
```

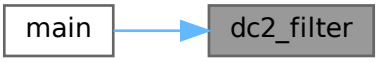
Here is the caller graph for this function:



4.3.1.15 dc2_filter()

```
double dc2_filter (
            const double raw)
00416 {
00417     static double accum = 0.0f;
00418     static double coef = COEF;
00419     accum = accum - accum / coef + raw;
00420     return accum / coef;
00421 }
```

Here is the caller graph for this function:



4.3.1.16 drive0_filter()

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

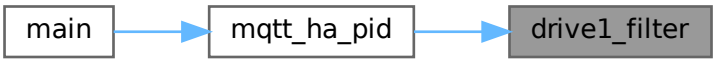
```
00424 {
00425     static double accum = 0.0f;
00426     static double coef = COEF;
00427     accum = accum - accum / coef + raw;
00428     return accum / coef;
00429 }
```

4.3.1.17 drive1_filter()

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

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

Here is the caller graph for this function:

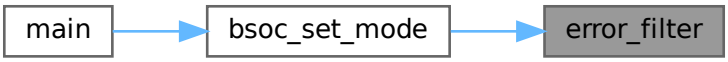


4.3.1.18 error_filter()

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

```
00369 {
00370     static double accum = 0.0f;
00371     accum = accum - accum / L.coef + raw;
00372     return accum / L.coef;
00373 }
```

Here is the caller graph for this function:

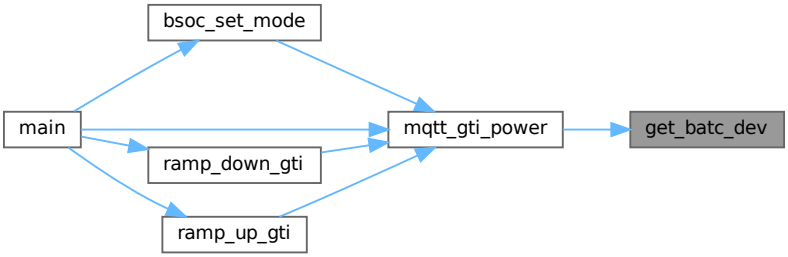


4.3.1.19 get_batc_dev()

```
double get_batc_dev (
            void )
```

```
00196 {
00197     return L.batc_std_dev;
00198 }
```

Here is the caller graph for this function:

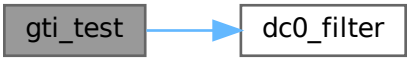


4.3.1.20 gti_test()

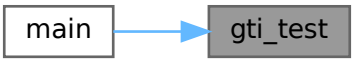
```
double gti_test (
            void )
```

```
00171 {
00172     // check for 48VDC AC charger powered from the Solar battery bank AC inverter
00173     if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
00174         L.gti_weight = 0.0f; // reduce power to zero
00175 #ifdef BSOC_DEBUG
00176         fprintf(fout, "pvp %8.2f, gweight %8.2f, aweight %8.2f, batv %8.2f, batc %8.2f\r\n",
      pv_voltage, gti_weight, ac_weight, bat_voltage, bat_current);
00177 #endif
00178     } else {
00179         if (E.dl_excess) {
00180             if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
00181                 L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
00182             } else {
00183                 L.gti_weight = 0.0f; // reduce power to zero
00184             }
00185         }
00186     }
00187     return dc0_filter(L.gti_weight);
00188 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.2 Variable Documentation

4.3.2.1 L

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

Initial value:

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

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

4.3.2.2 mqtt_name

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

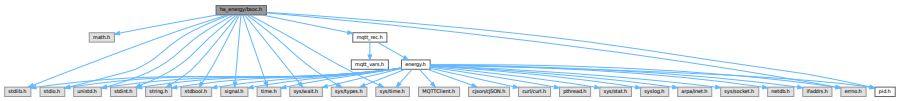
```
00003                                    {
00004     "pccmode",
00005     "batenergykw",
00006     "runtime",
00007     "bamps",
00008     "bvolts",
00009     "load",
```

```
00010     "solar",
00011     "acenergy",
00012     "benergy",
00013     "pwatts",
00014     "pamps",
00015     "pvolts",
00016     "flast",
00017     "HAdcsw",
00018     "HAacsw",
00019     "HAshut",
00020     "HAmode",
00021     "HAcon0",
00022     "HAcon1",
00023     "HAcon2",
00024     "HAcon3",
00025     "HAcon4",
00026     "HAcon5",
00027     "HAcon6",
00028     "HAcon7",
00029     "DLv_pv",
00030     "DLp_pv",
00031     "DLp_bat",
00032     "DLv_bat",
00033     "DLc_mppt",
00034     "DLp_mppt",
00035     "DLah_bat",
00036     "DLccmode",
00037     "DLgti",
00038 };
```

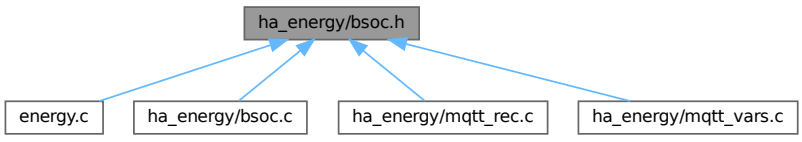
4.4 ha_energy/bsoc.h File Reference

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

Include dependency graph for bsoc.h:



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



Macros

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

Functions

- bool bsoc_init (void)
- bool bsoc_data_collect (void)
- double bsoc_ac (void)
- double bsoc_gti (void)
- double gti_test (void)
- double ac_test (void)
- double get_batc_dev (void)
- bool bat_current_stable (void)
- void bsoc_set_std_dev (const double, const uint32_t)
- double calculateStandardDeviation (const uint32_t, const double *)
- bool bsoc_set_mode (const double, const bool, const bool)
- double ac0_filter (const double)
- double ac1_filter (const double)
- double ac2_filter (const double)
- double dc0_filter (const double)
- double dc1_filter (const double)
- double dc2_filter (const double)
- double drive0_filter (const double)
- double drive1_filter (const double)

4.4.1 Macro Definition Documentation

4.4.1.1 BAT_C_DRAW

```
#define BAT_C_DRAW 3.0f
```

4.4.1.2 COEF

```
#define COEF 8.0f
```

4.4.1.3 COEFF

```
#define COEFF 2.0f
```

4.4.1.4 COEFN

```
#define COEFN 4.0f
```

4.4.1.5 DEV_SIZE

```
#define DEV_SIZE 10
```

4.4.1.6 MAX_BATC_DEV

```
#define MAX_BATC_DEV 1.5f
```

4.4.1.7 MIN_BAT_KW

```
#define MIN_BAT_KW 4100.0f
```

4.4.1.8 MIN_BAT_VOLTS

```
#define MIN_BAT_VOLTS 23.0f
```

4.4.1.9 MIN_PV_VOLTS

```
#define MIN_PV_VOLTS 5.0f
```

4.4.1.10 PBAL_OFFSET

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

4.4.1.11 PV_FULL_PWR

```
#define PV_FULL_PWR 300.0f
```

4.4.1.12 PV_MIN_PWR

```
#define PV_MIN_PWR 160.0f
```

4.4.1.13 PV_V_FAKE

```
#define PV_V_FAKE 0.336699f
```

4.4.1.14 PV_V_NOM

```
#define PV_V_NOM 60.0f
```

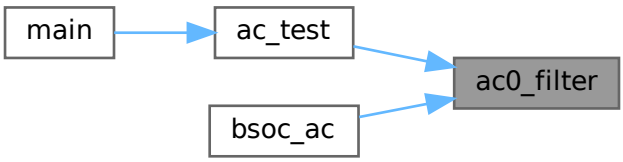
4.4.2 Function Documentation

4.4.2.1 ac0_filter()

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

```
00376 {
00377     static double accum = 0.0f;
00378     static double coef = COEFF;
00379     accum = accum - accum / coef + raw;
00380     return accum / coef;
00381 }
```

Here is the caller graph for this function:

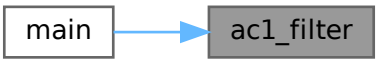


4.4.2.2 ac1_filter()

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

```
00384 {
00385     static double accum = 0.0f;
00386     static double coef = COEF;
00387     accum = accum - accum / coef + raw;
00388     return accum / coef;
00389 }
```

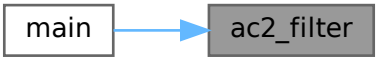
Here is the caller graph for this function:



4.4.2.3 ac2_filter()

```
double ac2_filter (
            const double raw)
00392 {
00393     static double accum = 0.0f;
00394     static double coef = COEF;
00395     accum = accum - accum / coef + raw;
00396     return accum / coef;
00397 }
```

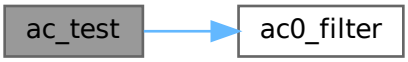
Here is the caller graph for this function:



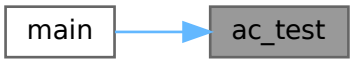
4.4.2.4 ac_test()

```
double ac_test (
            void )
00191 {
00192     return ac0_filter(L.ac_weight);
00193 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.4.2.5 bat_current_stable()

```
bool bat_current_stable (
            void )
```

```
00240 {
00241     static double gap = 0.0f;
00242
00243     if (L.batc_std_dev <= (MAX_BATC_DEV + gap)) {
00244         gap = MAX_BATC_DEV;
00245         if (L.bat_c_std_dev[0] < BAT_C_DRAW) {
00246             return true;
00247         } else {
00248             gap = 0.0f;
00249             return false;
00250         }
00251     } else {
00252         gap = 0.0f;
00253         return false;
00254     }
00255 }
```

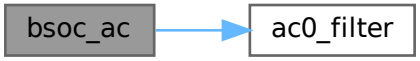
Here is the caller graph for this function:

4.4.2.6 bsoc_ac()

```
double bsoc_ac (
            void )
```

```
00136 {
00137
00138     return ac0_filter(L.ac_weight);
00139 };
```

Here is the call graph for this function:



4.4.2.7 bsoc_data_collect()

```
bool bsoc_data_collect (
            void )
```

```
00086 {
00087     bool ret = false;
00088     static uint32_t i = 0;
00089     // lockout threaded updates
00090     pthread_mutex_lock(&E.ha_lock); // lockout MQTT var updates
00091
00092     L.ac_weight = E.mvar[V_FBEKW];
00093     L.gti_weight = E.mvar[V_FBEKW];
00094 #ifdef FAKE_VPV // no DUMPLOAD AC charger
00095     if (E.gti_sw_on) {
00096         pv_voltage = PV_V_NOM;
00097     } else {
00098         pv_voltage = PV_V_FAKE;
00099     }
00100     E.mvar[V_DVPV] = pv_voltage;
00101 #else
00102     L.pv_voltage = E.mvar[V_DVPV];
00103 #endif
00104     L.bat_voltage = E.mvar[V_DVBAT];
00105     L.bat_current = E.mvar[V_DCMPPT];
00106     E.ac_low_adj = E.mvar[V_FSO]* -0.5f;
00107     E.gti_low_adj = E.mvar[V_FACE] * -0.5f;
00108     E.mode.dl_mqtt_max = E.mvar[V_DPMPPT];
00109
00110     pthread_mutex_unlock(&E.ha_lock); // resume remote MQTT var updates
00111
00112     if (E.ac_low_adj < -2000.0f) {
00113         E.ac_low_adj = -2000.0f;
00114     }
00115     if (E.gti_low_adj < -2000.0f) {
00116         E.gti_low_adj = -2000.0f;
00117     }
00118
00119     L.bat_c_std_dev[i++] = L.bat_current;
00120     if (i >= DEV_SIZE) {
00121         i = 0;
00122     }
00123
00124     calculateStandardDeviation(DEV_SIZE, L.bat_c_std_dev);
00125
00126 #ifdef BSOC_DEBUG
00127     fprintf(fout, "\r\nmqtt var bsoc update\r\n");
00128 #endif
00129     return ret;
00130 }
```

Here is the call graph for this function:

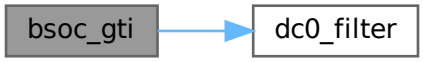
Here is the caller graph for this function:

4.4.2.8 bsoc_gti()

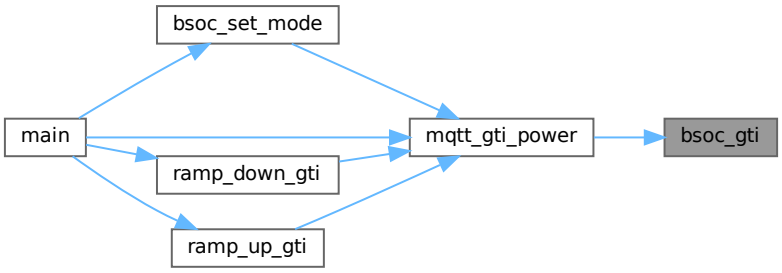
```
double bsoc_gti (
            void )
```

```
00146 {
00147 #ifdef BSOC_DEBUG
00148     fprintf(fout, "pvp %f, gweight %f, aweight %f, batv %f, batc %f\r\n", pv_voltage, gti_weight,
      ac_weight, bat_voltage, bat_current);
00149 #endif
00150     // check for 48VDC AC charger powered from the Solar battery bank AC inverter unless E.dl_excess
      is TRUE
00151     if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
00152         L.gti_weight = 0.0f; // reduce power to zero
00153     } else {
00154         if (E.dl_excess) {
00155             if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
00156                 L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
00157             } else {
00158                 L.gti_weight = 0.0f; // reduce power to zero
00159             }
00160         }
00161     }
00162
00163
00164     return dc0_filter(L.gti_weight);
00165 };
```

Here is the call graph for this function:



Here is the caller graph for this function:



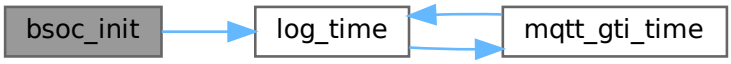
4.4.2.9 bsoc_init()

```
bool bsoc_init (
            void )
```

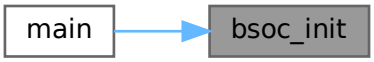
```
00061 {
```

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

Here is the call graph for this function:



Here is the caller graph for this function:

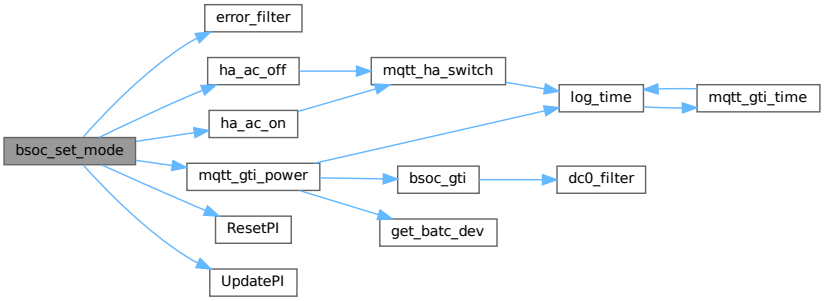


4.4.2.10 bsoc_set_mode()

```
bool bsoc_set_mode (
            const double target,
            const bool mode,
            const bool init)
00262 {
00263     static bool bsoc_mode = false;
00264     static bool bsoc_high = false, ha_ac_mode = true;
00265     static double accum = 0.0f, vpwa = 0.0f;
00266
00267     if (init) {
00268         bsoc_mode = false;
00269         bsoc_high = false;
00270         ha_ac_mode = true;
00271         accum = 0.0f;
00272         vpwa = 0.0f;
00273         return true;
00274     }
00275     /*
00276      * running avg filter
00277      */
00278     accum = accum - accum / COEFN + E.mvar[V_PWA];
00279     vpwa = accum / COEFN;
00280
00281     if ((vpwa >= PV_FULL_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
00282         if (!bsoc_mode) {
00283             ResetPI(&E.mode.pid);
00284         }
00285         bsoc_mode = true;
00286         bsoc_high = true;
```

```
00287         if (!ha_ac_mode) {
00288             ha_ac_on();
00289             ha_ac_mode = true;
00290         }
00291 
00292     } else {
00293         if (bsoc_high) { // turn off at min limit power
00294             if ((vpwa >= PV_MIN_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
00295                 bsoc_mode = true;
00296                 if (ha_ac_mode) {
00297                     ha_ac_off();
00298                     ha_ac_mode = false;
00299                 }
00300             } else {
00301                 bsoc_high = false;
00302                 ha_ac_mode = false;
00303             }
00304         }
00305     }
00306 
00307     E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) + E.mvar[V_DPPV]; // use as a temp variable
00308     E.mode.total_system = (E.mvar[V_FLO] - E.mode.gti_dumpload) + E.mvar[V_DPPV] +(E.print_vars[L3_P]*
      -1.0f);
00309     E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) - E.mvar[V_DPPV]; // use this value
00310 
00311     /*
00312      * look at system energy balance for power control drive
00313      */
00314     if (mode) { // add GTI power from dumpload
00315         E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + E.mode.gti_dumpload +
      PBAL_OFFSET);
00316     } else {
00317         E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + PBAL_OFFSET);
00318     }
00319 
00320     if (E.mode.error > 0.0f) {
00321         L.coef = COEF;
00322     } else {
00323         L.coef = COEFN;
00324     }
00325     E.mode.target = target;
00326     E.mode.error = round(error_filter(E.mode.error));
00327     /*
00328      * check for idle flag from HA
00329      */
00330     if (E.mode.con6) {
00331         ha_ac_mode = true;
00332         bsoc_mode = false;
00333     }
00334 
00335     /*
00336      * HA start excess button pressed
00337      */
00338     if (E.mode.con4) {
00339         E.dl_excess = true;
00340         E.mode.con4 = false;
00341     }
00342 
00343     /*
00344      * HA stop excess button pressed
00345      */
00346     if (E.mode.con5) {
00347         mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 9); // zero power at excess shutdown
00348         E.dl_excess = false;
00349         E.mode.con5 = false;
00350     }
00351 
00352     /*
00353      * DL buffer battery low set-point excess load shutdown
00354      */
00355     if (E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) {
00356         mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 10); // zero power at excess shutdown
00357         E.dl_excess = false;
00358         E.mode.con4 = false;
00359         E.mode.con5 = false;
00360     }
00361 
00362     return bsoc_mode;
00363 }
```

Here is the call graph for this function:

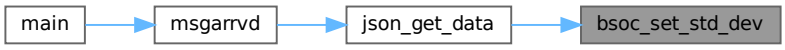


Here is the caller graph for this function:

4.4.2.11 bsoc_set_std_dev()

```
void bsoc_set_std_dev (
            const double value,
            const uint32_t i)
00076 {
00077     L.bat_c_std_dev[i] = value;
00078 }
```

Here is the caller graph for this function:



4.4.2.12 calculateStandardDeviation()

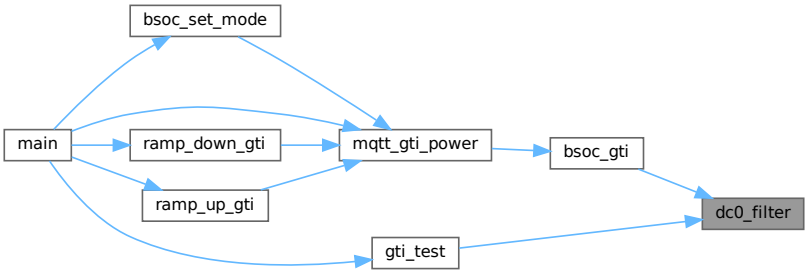
```
double calculateStandardDeviation (
            const uint32_t ,
            const double * )
```

4.4.2.13 dc0_filter()

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

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

Here is the caller graph for this function:

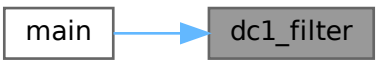


4.4.2.14 dc1_filter()

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

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

Here is the caller graph for this function:



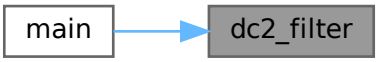
4.4.2.15 dc2_filter()

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

```
00416 {
```

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

Here is the caller graph for this function:



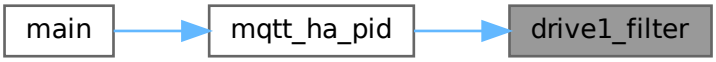
4.4.2.16 drive0_filter()

```
double drive0_filter (
            const double raw)
00424 {
00425     static double accum = 0.0f;
00426     static double coef = COEF;
00427     accum = accum - accum / coef + raw;
00428     return accum / coef;
00429 }
```

4.4.2.17 drive1_filter()

```
double drive1_filter (
            const double raw)
00432 {
00433     static double accum = 0.0f;
00434     static double coef = COEFF;
00435     accum = accum - accum / coef + raw;
00436     return accum / coef;
00437 }
```

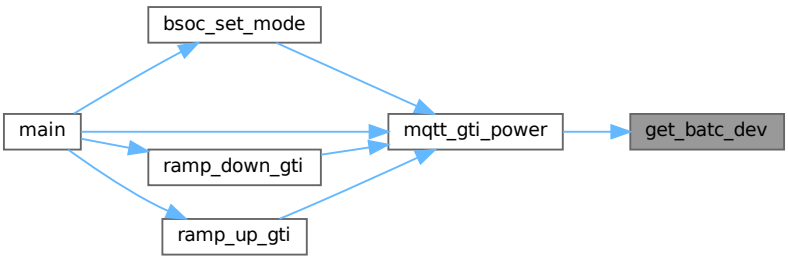
Here is the caller graph for this function:



4.4.2.18 get_batc_dev()

```
double get_batc_dev (
            void )
00196 {
00197     return L.batc_std_dev;
00198 }
```

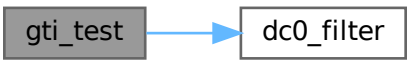
Here is the caller graph for this function:



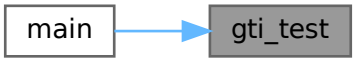
4.4.2.19 gti_test()

```
double gti_test (
             void )
00171 {
00172     // check for 48VDC AC charger powered from the Solar battery bank AC inverter
00173     if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
00174         L.gti_weight = 0.0f; // reduce power to zero
00175 #ifdef BSOC_DEBUG
00176         fprintf(fout, "pvp %8.2f, gweight %8.2f, aweight %8.2f, batv %8.2f, batc %8.2f\r\n",
      pv_voltage, gti_weight, ac_weight, bat_voltage, bat_current);
00177 #endif
00178     } else {
00179         if (E.dl_excess) {
00180             if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
00181                 L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
00182             } else {
00183                 L.gti_weight = 0.0f; // reduce power to zero
00184             }
00185         }
00186     }
00187     return dc0_filter(L.gti_weight);
00188 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.5 bsoc.h

Go to the documentation of this file.

```
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
```

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

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

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

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

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

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

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

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

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

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

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

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

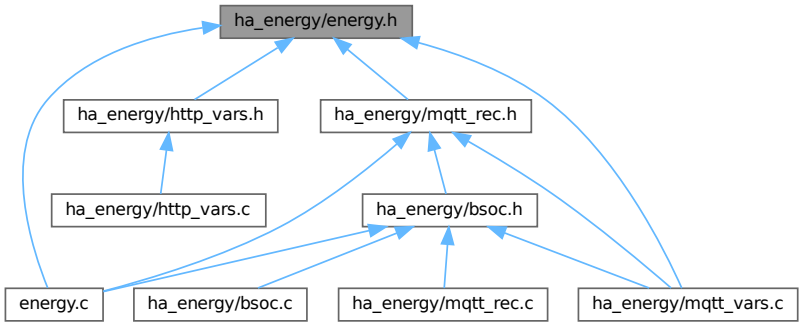
4.18 ha_energy/energy.h File Reference

```
#include <stdlib.h>
#include <stdio.h>
```

```
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#include <signal.h>
#include <time.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/time.h>
#include <errno.h>
#include <cjson/cJSON.h>
#include <curl/curl.h>
#include <pthread.h>
#include <sys/stat.h>
#include <syslog.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <netdb.h>
#include <ifaddrs.h>
#include "MQTTClient.h"
#include "pid.h"
```

Include dependency graph for energy.h:

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



Data Structures

- struct link_type
- struct mode_type
- struct energy_type

Macros

- #define LOG_VERSION "V0.73"

- #define MQTT_VERSION "V3.11"
- #define TNAME "maint9"
- #define LADDRESS "tcp://127.0.0.1:1883"
- #define ADDRESS "tcp://10.1.1.30:1883"
- #define CLIENTID1 "Energy_Mqtt_HA1"
- #define CLIENTID2 "Energy_Mqtt_HA2"
- #define CLIENTID3 "Energy_Mqtt_HA3"
- #define TOPIC_P "mateq84/data/gticmd"
- #define TOPIC_SPAM "mateq84/data/spam"
- #define TOPIC_PACA "home-assistant/gtiac/availability"
- #define TOPIC_PDCA "home-assistant/gtidc/availability"
- #define TOPIC_PACC "home-assistant/gtiac/contact"
- #define TOPIC_PDCC "home-assistant/gtidc/contact"
- #define TOPIC_PPID "home-assistant/solar/pid"
- #define TOPIC_SHUTDOWN "home-assistant/solar/shutdown"
- #define TOPIC_SS "mateq84/data/solar"
- #define TOPIC_SD "mateq84/data/dumpload"
- #define TOPIC_HA "home-assistant/status/switch"
- #define QOS 1
- #define TIMEOUT 10000L
- #define SPACING_USEC 500 ∗ 1000
- #define USEC_SEC 1000000L
- #define DAQ_STR 32
- #define DAQ_STR_M DAQ_STR-1
- #define SBUF_SIZ 16
- #define RBUF_SIZ 82
- #define SYSLOG_SIZ 512
- #define MQTT_TIMEOUT 900
- #define SW_QOS 1
- #define NO_CYLON
- #define CRITIAL_SHUTDOWN_LOG
- #define UNIT_TEST 2
- #define NORM_MODE 0
- #define PID_MODE 1
- #define MAX_ERROR 5
- #define IAM_DELAY 120
- #define CMD_SEC 10
- #define TIME_SYNC_SEC 30
- #define BAT_M_KW 5120.0f
- #define BAT_SOC_TOP 0.98f
- #define BAT_SOC_HIGH 0.95f
- #define BAT_SOC_LOW 0.64f
- #define BAT_SOC_LOW_AC 0.70f
- #define BAT_CRITICAL 200.0f
- #define MIN_BAT_KW_BSOC_SLP 4000.0f
- #define MIN_BAT_KW_BSOC_HI 4550.0f
- #define MIN_BAT_KW_GTI_HI BAT_M_KW∗BAT_SOC_TOP
- #define MIN_BAT_KW_GTI_LO BAT_M_KW∗BAT_SOC_LOW
- #define MIN_BAT_KW_AC_HI BAT_M_KW∗BAT_SOC_HIGH
- #define MIN_BAT_KW_AC_LO BAT_M_KW∗BAT_SOC_LOW_AC
- #define PV_PGAIN 0.85f
- #define PV_IGAIN 0.12f
- #define PV_IMAX 1400.0f
- #define PV_BIAS 288.0f
- #define PV_BIAS_ZERO 0.0f

- #define PV_BIAS_LOW 222.0f
- #define PV_BIAS_FLOAT 399.0f
- #define PV_BIAS_SLEEP 480.0f
- #define PV_BIAS_RATE 320.0f
- #define PV_DL_MPTT_MAX 1200.0f
- #define PV_DL_MPTT_EXCESS 1300.0f
- #define PV_DL_MPTT_IDLE 57.0f
- #define PV_DL_BIAS_RATE 75.0f
- #define PV_DL_EXCESS 500.0f
- #define PV_DL_B_AH_LOW 100.0f
- #define PV_DL_B_AH_MIN 150.0f
- #define PV_DL_B_V_LOW 23.8f
- #define PWA_SLEEP 200.0f
- #define DL_AC_DC_EFF 1.24f
- #define BAL_MIN_ENERGY_AC -200.0f
- #define BAL_MAX_ENERGY_AC 200.0f
- #define BAL_MIN_ENERGY_GTI -1400.0f
- #define BAL_MAX_ENERGY_GTI 200.0f
- #define LOG_TO_FILE "/store/logs/energy.log"
- #define LOG_TO_FILE_ALT "/tmp/energy.log"
- #define MAX_LOG_SPAM 60
- #define LOW_LOG_SPAM 2
- #define RESET_LOG_SPAM 120
- #define IM_DELAY 1
- #define IM_DISPLAY 1
- #define GTI_DELAY 1
- #define PWA_SANE 1700.0f
- #define PAMPS_SANE 16.0f
- #define PVOLTS_SANE 150.0f
- #define 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 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

4.18.1 Macro Definition Documentation

4.18.1.1 ADDRESS

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

4.18.1.2 BAL_MAX_ENERGY_AC

```
#define BAL_MAX_ENERGY_AC 200.0f
```

4.18.1.3 BAL_MAX_ENERGY_GTI

```
#define BAL_MAX_ENERGY_GTI 200.0f
```

4.18.1.4 BAL_MIN_ENERGY_AC

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

4.18.1.5 BAL_MIN_ENERGY_GTI

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

4.18.1.6 BAMPS_SANE

```
#define BAMPS_SANE 70.0f
```

4.18.1.7 BAT_CRITICAL

```
#define BAT_CRITICAL 200.0f
```

4.18.1.8 BAT_M_KW

```
#define BAT_M_KW 5120.0f
```

4.18.1.9 BAT_SOC_HIGH

```
#define BAT_SOC_HIGH 0.95f
```

4.18.1.10 BAT_SOC_LOW

```
#define BAT_SOC_LOW 0.64f
```

4.18.1.11 BAT_SOC_LOW_AC

```
#define BAT_SOC_LOW_AC 0.70f
```

4.18.1.12 BAT_SOC_TOP

```
#define BAT_SOC_TOP 0.98f
```

4.18.1.13 CLIENTID1

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

4.18.1.14 CLIENTID2

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

4.18.1.15 CLIENTID3

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

4.18.1.16 CMD_SEC

```
#define CMD_SEC 10
```

4.18.1.17 CRITIAL_SHUTDOWN_LOG

```
#define CRITIAL_SHUTDOWN_LOG
```

4.18.1.18 DAQ_STR

```
#define DAQ_STR 32
```

4.18.1.19 DAQ_STR_M

```
#define DAQ_STR_M DAQ_STR-1
```

4.18.1.20 DL_AC_DC_EFF

```
#define DL_AC_DC_EFF 1.24f
```

4.18.1.21 GTI_DELAY

```
#define GTI_DELAY 1
```

4.18.1.22 IAM_DELAY

```
#define IAM_DELAY 120
```

4.18.1.23 IM_DELAY

```
#define IM_DELAY 1
```

4.18.1.24 IM_DISPLAY

```
#define IM_DISPLAY 1
```

4.18.1.25 L1_P

```
#define L1_P IA_POWER
```

4.18.1.26 L2_P

```
#define L2_P L1_P+IA_LAST
```

4.18.1.27 L3_P

```
#define L3_P L2_P+IA_LAST
```

4.18.1.28 LADDRESS

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

4.18.1.29 LOG_TO_FILE

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

4.18.1.30 LOG_TO_FILE_ALT

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

4.18.1.31 LOG_VERSION

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

4.18.1.32 LOW_LOG_SPAM

```
#define LOW_LOG_SPAM 2
```

4.18.1.33 MAX_ERROR

```
#define MAX_ERROR 5
```

4.18.1.34 MAX_IM_VAR

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

4.18.1.35 MAX_LOG_SPAM

```
#define MAX_LOG_SPAM 60
```

4.18.1.36 MIN_BAT_KW_AC_HI

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

4.18.1.37 MIN_BAT_KW_AC_LO

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

4.18.1.38 MIN_BAT_KW_BSOC_HI

```
#define MIN_BAT_KW_BSOC_HI 4550.0f
```

4.18.1.39 MIN_BAT_KW_BSOC_SLP

```
#define MIN_BAT_KW_BSOC_SLP 4000.0f
```

4.18.1.40 MIN_BAT_KW_GTI_HI

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

4.18.1.41 MIN_BAT_KW_GTI_LO

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

4.18.1.42 MQTT_TIMEOUT

```
#define MQTT_TIMEOUT 900
```

4.18.1.43 MQTT_VERSION

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

4.18.1.44 NO_CYLON

```
#define NO_CYLON
```

4.18.1.45 NORM_MODE

```
#define NORM_MODE 0
```

4.18.1.46 PAMPS_SANE

```
#define PAMPS_SANE 16.0f
```

4.18.1.47 PID_MODE

```
#define PID_MODE 1
```

4.18.1.48 PV_BIAS

```
#define PV_BIAS 288.0f
```

4.18.1.49 PV_BIAS_FLOAT

```
#define PV_BIAS_FLOAT 399.0f
```

4.18.1.50 PV_BIAS_LOW

```
#define PV_BIAS_LOW 222.0f
```

4.18.1.51 PV_BIAS_RATE

```
#define PV_BIAS_RATE 320.0f
```

4.18.1.52 PV_BIAS_SLEEP

```
#define PV_BIAS_SLEEP 480.0f
```

4.18.1.53 PV_BIAS_ZERO

```
#define PV_BIAS_ZERO 0.0f
```

4.18.1.54 PV_DL_B_AH_LOW

```
#define PV_DL_B_AH_LOW 100.0f
```

4.18.1.55 PV_DL_B_AH_MIN

```
#define PV_DL_B_AH_MIN 150.0f
```

4.18.1.56 PV_DL_B_V_LOW

```
#define PV_DL_B_V_LOW 23.8f
```

4.18.1.57 PV_DL_BIAS_RATE

```
#define PV_DL_BIAS_RATE 75.0f
```

4.18.1.58 PV_DL_EXCESS

```
#define PV_DL_EXCESS 500.0f
```

4.18.1.59 PV_DL_MPTT_EXCESS

```
#define PV_DL_MPTT_EXCESS 1300.0f
```

4.18.1.60 PV_DL_MPTT_IDLE

```
#define PV_DL_MPTT_IDLE 57.0f
```

4.18.1.61 PV_DL_MPTT_MAX

```
#define PV_DL_MPTT_MAX 1200.0f
```

4.18.1.62 PV_IGAIN

```
#define PV_IGAIN 0.12f
```

4.18.1.63 PV_IMAX

```
#define PV_IMAX 1400.0f
```

4.18.1.64 PV_PGAIN

```
#define PV_PGAIN 0.85f
```

4.18.1.65 PVOLTS_SANE

```
#define PVOLTS_SANE 150.0f
```

4.18.1.66 PWA_SANE

```
#define PWA_SANE 1700.0f
```

4.18.1.67 PWA_SLEEP

```
#define PWA_SLEEP 200.0f
```

4.18.1.68 QOS

```
#define QOS 1
```

4.18.1.69 RBUF_SIZ

```
#define RBUF_SIZ 82
```

4.18.1.70 RESET_LOG_SPAM

```
#define RESET_LOG_SPAM 120
```

4.18.1.71 SBUF_SIZ

```
#define SBUF_SIZ 16
```

4.18.1.72 SPACING_USEC

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

4.18.1.73 SW_QOS

```
#define SW_QOS 1
```

4.18.1.74 SYSLOG_SIZ

```
#define SYSLOG_SIZ 512
```

4.18.1.75 TIME_SYNC_SEC

```
#define TIME_SYNC_SEC 30
```

4.18.1.76 TIMEOUT

```
#define TIMEOUT 10000L
```

4.18.1.77 TNAME

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

4.18.1.78 TOPIC_HA

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

4.18.1.79 TOPIC_P

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

4.18.1.80 TOPIC_PACA

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

4.18.1.81 TOPIC_PACC

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

4.18.1.82 TOPIC_PDCA

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

4.18.1.83 TOPIC_PDCC

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

4.18.1.84 TOPIC_PPID

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

4.18.1.85 TOPIC_SD

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

4.18.1.86 TOPIC_SHUTDOWN

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

4.18.1.87 TOPIC_SPAM

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

4.18.1.88 TOPIC_SS

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

4.18.1.89 UNIT_TEST

```
#define UNIT_TEST 2
```

4.18.1.90 USEC_SEC

```
#define USEC_SEC 1000000L
```

4.18.2 Enumeration Type Documentation

4.18.2.1 energy_state

```
enum energy_state
```

Enumerator

E_INIT	
E_RUN	
E_WAIT	
E_IDLE	
E_STOP	
E_LAST	

```
00194                       {
00195         E_INIT,
00196         E_RUN,
00197         E_WAIT,
00198         E_IDLE,
00199         E_STOP,
00200         E_LAST,
00201     };
```

4.18.2.2 iammeter_id

```
enum iammeter_id
```

Enumerator

IA_VOLTAGE	
IA_CURRENT	
IA_POWER	
IA_IMPORT	
IA_EXPORT	
IA_FREQ	
IA_PF	
IA_LAST	

```
00219                       {
00220         IA_VOLTAGE,
00221         IA_CURRENT,
00222         IA_POWER,
00223         IA_IMPORT,
00224         IA_EXPORT,
00225         IA_FREQ,
00226         IA_PF,
00227         IA_LAST,
00228     };
```

4.18.2.3 iammeter_phase

```
enum iammeter_phase
```

Enumerator

PHASE_A	
PHASE_B	
PHASE_C	
PHASE_LAST	

```
00212                         {
00213         PHASE_A,
00214         PHASE_B,
00215         PHASE_C,
00216         PHASE_LAST,
00217     };
```

4.18.2.4 mqtt_vars

```
enum mqtt_vars
```

Enumerator

V_FCCM	
V_FBEKW	
V_FRUNT	
V_FBAMPS	
V_FBV	
V_FLO	
V_FSO	
V_FACE	
V_BEN	
V_PWA	

Enumerator

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	

```
00230                   {
00231         V_FCCM,
00232         V_FBEKW,
00233         V_FRUNT,
00234         V_FBAMPS,
00235         V_FBV,
00236         V_FLO,
00237         V_FSO,
00238         V_FACE,
00239         V_BEN,
00240         V_PWA,
00241         V_PAMPS,
00242         V_PVOLTS,
00243         V_FLAST,
00244         V_HDCSW,
00245         V_HACSW,
00246         V_HSHUT,
00247         V_HMODE,
00248         V_HCON0,
00249         V_HCON1,
00250         V_HCON2,
00251         V_HCON3,
00252         V_HCON4,
00253         V_HCON5,
00254         V_HCON6,
00255         V_HCON7,
00256         // add other data ranges here
00257         V_DVPV,
00258         V_DPPV,
00259         V_DPBAT,
00260         V_DVBAT,
00261         V_DCMPPT,
00262         V_DPMPPT,
00263         V_DAHBAT,
00264         V_DCCMODE,
00265         V_DGTI,
00266         V_DLAST,
00267     };
```

4.18.2.5 running_state

```
enum running_state
```

Enumerator

R_INIT	
R_FLOAT	
R_SLEEP	
R_RUN	
R_IDLE	
R_LAST	

```
00203                       {
00204         R_INIT,
00205         R_FLOAT,
00206         R_SLEEP,
00207         R_RUN,
00208         R_IDLE,
00209         R_LAST,
00210     };
```

4.18.2.6 sane_vars

```
enum sane_vars
```

Enumerator

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

S_DLAST	

```
00269                   {
00270         S_FCCM,
00271         S_FBEKW,
00272         S_FRUNT,
00273         S_FBAMPS,
00274         S_FBV,
00275         S_FLO,
00276         S_FSO,
00277         S_FACE,
00278         S_BEN,
00279         S_PWA,
00280         S_PAMPS,
00281         S_PVOLTS,
00282         S_FLAST,
00283         S_HDCSW,
00284         S_HACSW,
00285         S_HSHUT,
00286         S_HMODE,
00287         // add other data ranges here
00288         S_DVPV,
00289         S_DPPV,
00290         S_DPBAT,
00291         S_DVBAT,
00292         S_DCMPPT,
00293         S_DPMPPT,
00294         S_DAHBAT,
00295         S_DCCMODE,
00296         S_DGTI,
00297         S_DLAST,
00298     };
```

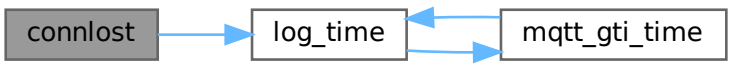
4.18.3 Function Documentation

4.18.3.1 connlost()

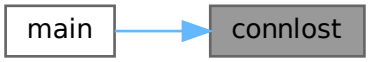
```
void connlost (
             void * context,
             char * cause)
```

```
00298 {
00299     struct ha_flag_type *ha_flag = context;
00300     int32_t id_num;
00301
00302     // bug-out if no context variables passed to callback
00303     if (context == NULL) {
00304         id_num = -1;
00305     } else {
00306         id_num = ha_flag->ha_id;
00307     }
00308     fprintf(fout, "\n%s Connection lost, exit ha_energy program\n", log_time(false));
00309     fprintf(fout, "%s     cause: %s, %d\n", log_time(false), cause, id_num);
00310     fprintf(fout, "%sDAEMON failure  LOG Version %s : MQTT Version %s\n", log_time(false),
      LOG_VERSION, MQTT_VERSION);
00311     fflush(fout);
00312     exit(EXIT_FAILURE);
00313 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

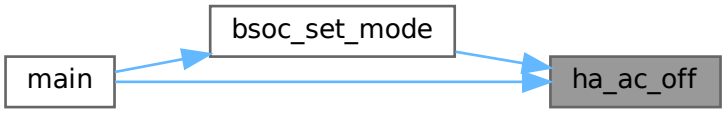


4.18.3.2 ha_ac_off()

```
void ha_ac_off (
            void )
00947 {
00948     mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
00949     E.ac_sw_status = false;
00950 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

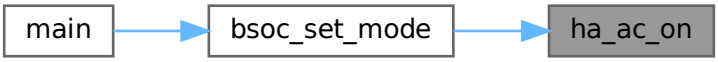


4.18.3.3 ha_ac_on()

```
void ha_ac_on (
            void )
00953 {
00954     mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
00955     E.ac_sw_status = true;
00956 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

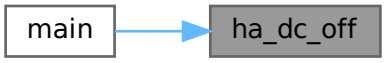


4.18.3.4 ha_dc_off()

```
void ha_dc_off (
              void )
00962 {
00963     mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
00964     E.gti_sw_status = false;
00965 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.18.3.5 ha_dc_on()

```
void ha_dc_on (
            void )
```

```
00968 {
00969     mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
00970     E.gti_sw_status = true;
00971 }
```

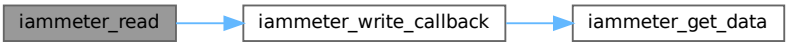
Here is the call graph for this function:

4.18.3.6 iammeter_read()

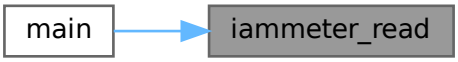
```
void iammeter_read (
            void )
```

```
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 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.18.3.7 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 }
```

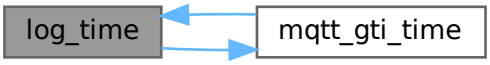
Here is the call graph for this function:

Here is the caller graph for this function:

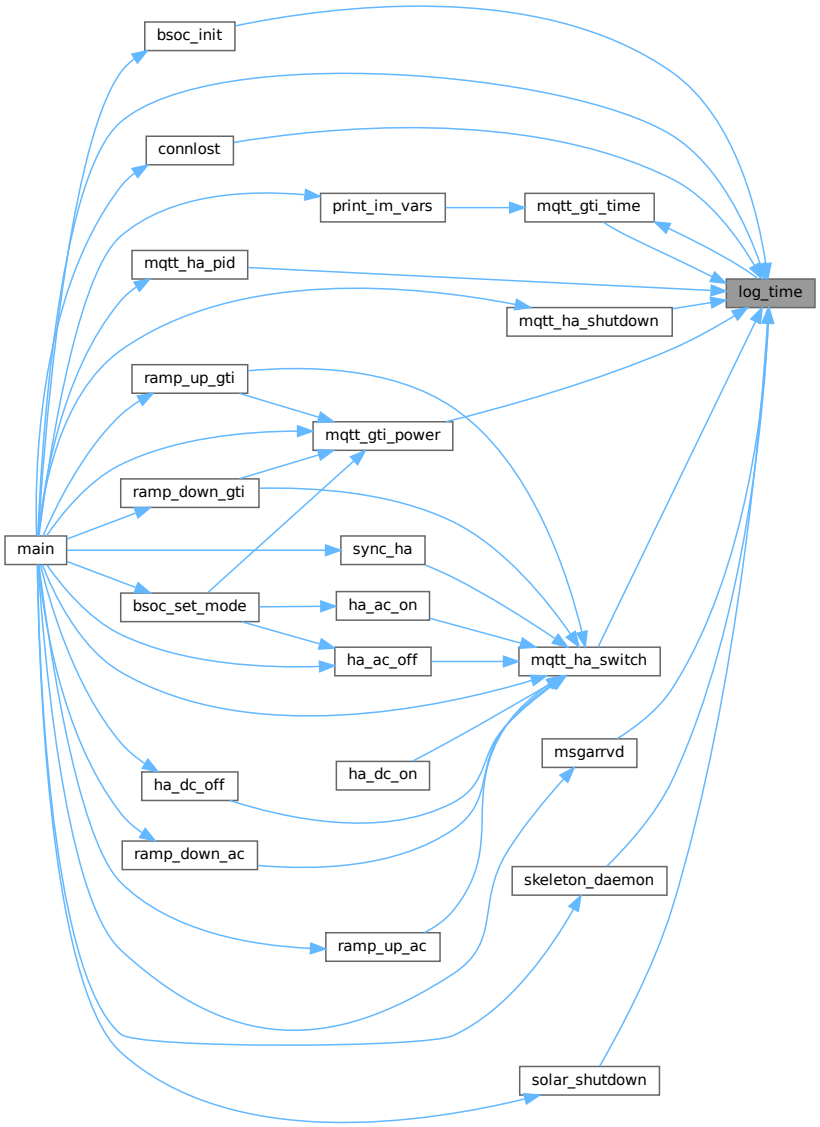
4.18.3.8 log_time()

```
char * log_time (
            bool log)
01032 {
01033     static char time_log[RBUF_SIZ] = {0};
01034     static uint32_t len = 0, sync_time = TIME_SYNC_SEC - 1;
01035     time_t rawtime_log;
01036
01037     tzset();
01038     timezone = 0;
01039     daylight = 0;
01040     time(&rawtime_log);
01041     if (sync_time++ > TIME_SYNC_SEC) {
01042         sync_time = 0;
01043         snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti
      time commands
01044         mqtt_gti_time(E.client_p, TOPIC_P, time_log);
01045     }
01046
01047     sprintf(time_log, "%s", ctime(&rawtime_log));
01048     len = strlen(time_log);
01049     time_log[len - 1] = 0; // munge out the return character
01050     if (log) {
01051         fprintf(fout, "%s ", time_log);
01052         fflush(fout);
01053     }
01054
01055     return time_log;
01056 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

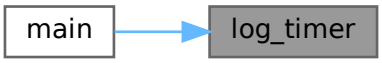


4.18.3.9 log_timer()

```
bool log_timer (
            void )
```

```
01091 {
01092     bool itstime = false;
01093 
01094     if (E.log_spam < LOW_LOG_SPAM) {
01095         E.log_time_reset = 0;
01096         itstime = true;
01097     }
01098     if (E.log_time_reset > RESET_LOG_SPAM) {
01099         E.log_spam = 0;
01100         itstime = true;
01101     }
01102     return itstime;
01103 }
```

Here is the caller graph for this function:

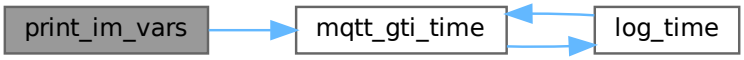


4.18.3.10 print_im_vars()

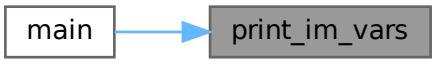
```
void print_im_vars (
            void )
```

```
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 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.18.3.11 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 }
```

Here is the caller graph for this function:

4.18.3.12 ramp_down_ac()

```
void ramp_down_ac (
            MQTTClient client_p,
            bool sw_off)
00937 {
00938     if (sw_off) {
00939         mqtt_ha_switch(client_p, TOPIC_PACC, false);
00940         E.ac_sw_status = false;
00941         usleep(500000);
00942     }
00943     E.once_ac = true;
00944 }
```

Here is the call graph for this function:

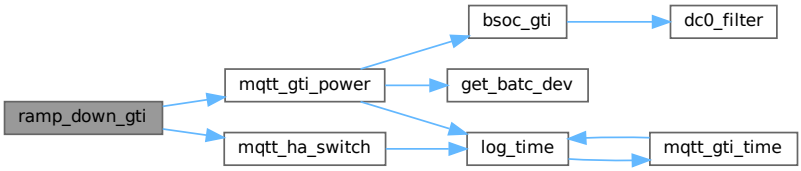
Here is the caller graph for this function:

4.18.3.13 ramp_down_gti()

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

```
00904 {
00905     if (sw_off) {
00906         mqtt_ha_switch(client_p, TOPIC_PDCC, false);
00907         E.once_gti_zero = true;
00908         E.gti_sw_status = false;
00909     }
00910     E.once_gti = true;
00911
00912     if (E.once_gti_zero) {
00913         mqtt_gti_power(client_p, TOPIC_P, "Z#", 7); // zero power
00914         E.once_gti_zero = false;
00915     }
00916 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

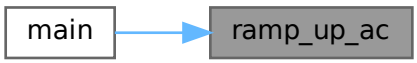
4.18.3.14 ramp_up_ac()

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

```
00922 {
00923
00924     if (start) {
00925         E.once_ac = true;
00926     }
00927
00928     if (E.once_ac) {
00929         E.once_ac = false;
00930         mqtt_ha_switch(client_p, TOPIC_PACC, true);
00931         E.ac_sw_status = true;
00932         usleep(500000); // wait for voltage to ramp
00933     }
00934 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



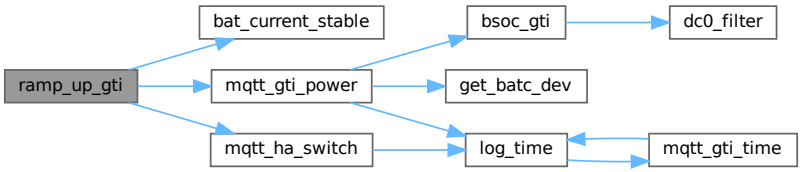
4.18.3.15 ramp_up_gti()

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

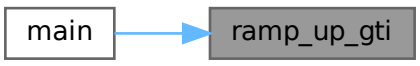
```
00843 {
00844     static uint32_t sequence = 0;
00845
00846     if (start) {
00847         E.once_gti = true;
00848     }
00849
00850     if (E.once_gti) {
00851         E.once_gti = false;
00852         sequence = 0;
00853         if (!excess) {
00854             mqtt_ha_switch(client_p, TOPIC_PDCC, true);
00855             E.gti_sw_status = true;
00856             usleep(500000); // wait for voltage to ramp
00857         } else {
00858             sequence = 1;
```

```
00859         }
00860     }
00861
00862     switch (sequence) {
00863     case 4:
00864         E.once_gti_zero = true;
00865         break;
00866     case 3:
00867     case 2:
00868     case 1:
00869         E.once_gti_zero = true;
00870         if (bat_current_stable() || E.dl_excess) { // check battery current std dev, stop
      'motorboating'
00871             sequence++;
00872             if (!mqtt_gti_power(client_p, TOPIC_P, "+#", 3)) {
00873                 sequence = 0;
00874             }; // +100W power
00875         } else {
00876             usleep(500000); // wait a bit more for power to be stable
00877             sequence = 1; // do power ramps when ready
00878             if (!mqtt_gti_power(client_p, TOPIC_P, "-#", 4)) {
00879                 sequence = 0;
00880             }; // - 100W power
00881         }
00882         break;
00883     case 0:
00884         sequence++;
00885         if (E.once_gti_zero) {
00886             mqtt_gti_power(client_p, TOPIC_P, "Z#", 5); // zero power
00887             E.once_gti_zero = false;
00888         }
00889         break;
00890     default:
00891         if (E.once_gti_zero) {
00892             mqtt_gti_power(client_p, TOPIC_P, "Z#", 6); // zero power
00893             E.once_gti_zero = false;
00894         }
00895         sequence = 0;
00896         break;
00897     }
00898 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.18.3.16 sanity_check()

```
bool sanity_check (
            void )
```

```
00254 {
00255     if (E.mvar[V_PWA] > PWA_SANE) {
00256         E.sane = S_PWA;
00257         return false;
00258     }
00259     if (E.mvar[V_PAMPS] > PAMPS_SANE) {
00260         E.sane = S_PAMPS;
00261         return false;
00262     }
00263     if (E.mvar[V_PVOLTS] > PVOLTS_SANE) {
00264         E.sane = S_PVOLTS;
00265         return false;
00266     }
00267     if (E.mvar[V_FBAMPS] > BAMPS_SANE) {
00268         E.sane = S_FBAMPS;
00269         return false;
00270     }
00271     return true;
00272 }
```

Here is the caller graph for this function:

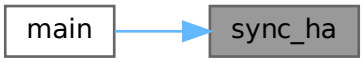
4.18.3.17 sync_ha()

```
bool sync_ha (
            void )
```

```
01062 {
01063     bool sync = false;
01064     if (E.gti_sw_status != (bool) ((int32_t) E.mvar[V_HDCSW])) {
01065         fprintf(fout, "DC_MM %d %d ", (bool) E.gti_sw_status, (bool) ((int32_t) E.mvar[V_HDCSW]));
01066         mqtt_ha_switch(E.client_p, TOPIC_PDCC, !E.gti_sw_status);
01067         E.dc_mismatch = true;
01068         fflush(fout);
01069         sync = true;
01070     } else {
01071         E.dc_mismatch = false;
01072     }
01073
01074     E.ac_sw_status = (bool) ((int32_t) E.mvar[V_HACSW]); // TEMP FIX for MISmatch errors
01075     if (E.ac_sw_status != (bool) ((int32_t) E.mvar[V_HACSW])) {
01076         fprintf(fout, "AC_MM %d %d ", (bool) E.ac_sw_status, (bool) ((int32_t) E.mvar[V_HACSW]));
01077         mqtt_ha_switch(E.client_p, TOPIC_PACC, !E.ac_sw_status);
01078         E.ac_mismatch = true;
01079         fflush(fout);
01080         sync = true;
01081     } else {
01082         E.ac_mismatch = false;
01083     }
01084     return sync;
01085 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

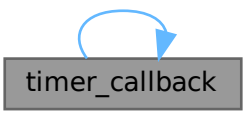


4.18.3.18 timer_callback()

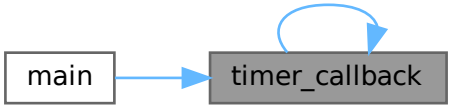
```
void timer_callback (
             int32_t signum)
```

```
00283 {
00284     signal(signum, timer_callback);
00285     ha_flag_vars_ss.runner = true;
00286     E.ten_sec_clock++;
00287     E.log_spam++;
00288     E.log_time_reset++;
00289     if (E.log_spam > MAX_LOG_SPAM) {
00290         E.log_spam = 0;
00291     }
00292 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.18.4 Variable Documentation

4.18.4.1 E

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

```
00107                           {
00108     .once_gti = true,
00109     .once_ac = true,
00110     .once_gti_zero = true,
00111     .iammeter = false,
00112     .fm80 = false,
00113     .dumpload = false,
00114     .homeassistant = false,
00115     .ac_low_adj = 0.0f,
00116     .gti_low_adj = 0.0f,
00117     .ac_sw_on = true,
00118     .gti_sw_on = true,
00119     .im_delay = 0,
00120     .gti_delay = 0,
00121     .im_display = 0,
00122     .rc = 0,
00123     .speed_go = 0,
00124     .mode.pid.iMax = PV_IMAX,
00125     .mode.pid.iMin = 0.0f,
00126     .mode.pid.pGain = PV_PGAIN,
00127     .mode.pid.iGain = PV_IGAIN,
00128     .mode.mode_tmr = 0,
00129     .mode.mode = true,
00130     .mode.in_pid_control = false,
00131     .mode.dl_mqtt_max = PV_DL_MPTT_MAX,
00132     .mode.E = E_INIT,
00133     .mode.R = R_INIT,
00134     .mode.no_float = true,
00135     .mode.data_error = false,
00136     .ac_sw_status = false,
00137     .gti_sw_status = false,
00138     .solar_mode = false,
00139     .solar_shutdown = false,
00140     .mode.pv_bias = PV_BIAS_LOW,
00141     .sane = S_DLAST,
00142     .startup = true,
00143     .ac_mismatch = false,
00144     .dc_mismatch = false,
00145     .mode_mismatch = false,
00146     .link.shutdown = 0,
00147     .mode.bat_crit = false,
00148     .dl_excess = false,
00149     .dl_excess_adj = 0.0f,
00150 };
```

4.18.4.2 fout

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

4.18.4.3 ha_flag_vars_ss

```
struct ha_flag_type ha_flag_vars_ss  [extern]
00071                                        {
00072     .runner = false,
00073     .receivedtoken = false,
00074     .deliveredtoken = false,
00075     .rec_ok = false,
00076     .ha_id = FM80_ID,
00077     .var_update = 0,
00078     .energy_mode = NORM_MODE,
00079 };
```

4.19 energy.h

Go to the documentation of this file.

```
00001 /*
00002  * File:   bmc.h
00003  * Author: root
00004  *
00005  * Created on September 21, 2012, 12:54 PM
00006  */
00007
00008 #ifndef BMC_H
00009 #define BMC_H
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014 #include <stdlib.h>
00015 #include <stdio.h> /* for printf() */
00016 #include <unistd.h>
00017 #include <stdint.h>
00018 #include <string.h>
00019 #include <stdbool.h>
00020 #include <signal.h>
00021 #include <time.h>
00022 #include <sys/wait.h>
00023 #include <sys/types.h>
00024 #include <sys/time.h>
00025 #include <errno.h>
00026 #include <cjson/cJSON.h>
00027 #include <curl/curl.h>
00028 #include <pthread.h>
00029 #include <sys/stat.h>
00030 #include <syslog.h>
00031 #include <arpa/inet.h>
00032 #include <sys/socket.h>
00033 #include <netdb.h>
00034 #include <ifaddrs.h>
00035 #include "MQTTClient.h"
00036 #include "pid.h"
00037
00038
00039 #define LOG_VERSION     "V0.73"
00040 #define MQTT_VERSION    "V3.11"
00041 #define TNAME  "maint9"
00042 #define LADDRESS        "tcp://127.0.0.1:1883"
00043 #ifdef __amd64
00044 #define ADDRESS         "tcp://10.1.1.172:1883"
00045 #else
00046 #define ADDRESS         "tcp://10.1.1.30:1883"
00047 #endif
00048 #define CLIENTID1       "Energy_Mqtt_HA1"
00049 #define CLIENTID2       "Energy_Mqtt_HA2"
00050 #define CLIENTID3       "Energy_Mqtt_HA3"
00051 #define TOPIC_P         "mateq84/data/gticmd"
00052 #define TOPIC_SPAM      "mateq84/data/spam"
00053 #define TOPIC_PACA      "home-assistant/gtiac/availability"
00054 #define TOPIC_PDCA      "home-assistant/gtidc/availability"
00055 #define TOPIC_PACC      "home-assistant/gtiac/contact"
00056 #define TOPIC_PDCC      "home-assistant/gtidc/contact"
00057 #define TOPIC_PPID      "home-assistant/solar/pid"
00058 #define TOPIC_SHUTDOWN  "home-assistant/solar/shutdown"
00059 #define TOPIC_SS        "mateq84/data/solar"
00060 #define TOPIC_SD        "mateq84/data/dumpload"
00061 #define TOPIC_HA        "home-assistant/status/switch"
00062 #define QOS             1
```

```
00063 #define TIMEOUT         10000L
00064 #define SPACING_USEC    500 * 1000
00065 #define USEC_SEC        1000000L
00066
00067 #define DAQ_STR 32
00068 #define DAQ_STR_M DAQ_STR-1
00069
00070 #define SBUF_SIZ        16  // short buffer string size
00071 #define RBUF_SIZ        82
00072 #define SYSLOG_SIZ      512
00073
00074 #define MQTT_TIMEOUT    900
00075 #define SW_QOS          1
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         0.64f
00096 #define BAT_SOC_LOW_AC      0.70f
00097 #define BAT_CRITICAL        200.0f
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 energy_state {
00195         E_INIT,
00196         E_RUN,
00197         E_WAIT,
00198         E_IDLE,
00199         E_STOP,
00200         E_LAST,
00201     };
00202
00203     enum running_state {
00204         R_INIT,
00205         R_FLOAT,
00206         R_SLEEP,
00207         R_RUN,
00208         R_IDLE,
00209         R_LAST,
00210     };
00211
00212     enum iammeter_phase {
00213         PHASE_A,
00214         PHASE_B,
00215         PHASE_C,
00216         PHASE_LAST,
00217     };
00218
00219     enum iammeter_id {
00220         IA_VOLTAGE,
00221         IA_CURRENT,
00222         IA_POWER,
00223         IA_IMPORT,
00224         IA_EXPORT,
00225         IA_FREQ,
00226         IA_PF,
00227         IA_LAST,
00228     };
00229
00230     enum mqtt_vars {
00231         V_FCCM,
00232         V_FBEKW,
00233         V_FRUNT,
00234         V_FBAMPS,
00235         V_FBV,
```

```
00236         V_FLO,
00237         V_FSO,
00238         V_FACE,
00239         V_BEN,
00240         V_PWA,
00241         V_PAMPS,
00242         V_PVOLTS,
00243         V_FLAST,
00244         V_HDCSW,
00245         V_HACSW,
00246         V_HSHUT,
00247         V_HMODE,
00248         V_HCON0,
00249         V_HCON1,
00250         V_HCON2,
00251         V_HCON3,
00252         V_HCON4,
00253         V_HCON5,
00254         V_HCON6,
00255         V_HCON7,
00256         // add other data ranges here
00257         V_DVPV,
00258         V_DPPV,
00259         V_DPBAT,
00260         V_DVBAT,
00261         V_DCMPPT,
00262         V_DPMPPT,
00263         V_DAHBAT,
00264         V_DCCMODE,
00265         V_DGTI,
00266         V_DLAST,
00267     };
00268
00269     enum sane_vars {
00270         S_FCCM,
00271         S_FBEKW,
00272         S_FRUNT,
00273         S_FBAMPS,
00274         S_FBV,
00275         S_FLO,
00276         S_FSO,
00277         S_FACE,
00278         S_BEN,
00279         S_PWA,
00280         S_PAMPS,
00281         S_PVOLTS,
00282         S_FLAST,
00283         S_HDCSW,
00284         S_HACSW,
00285         S_HSHUT,
00286         S_HMODE,
00287         // add other data ranges here
00288         S_DVPV,
00289         S_DPPV,
00290         S_DPBAT,
00291         S_DVBAT,
00292         S_DCMPPT,
00293         S_DPMPPT,
00294         S_DAHBAT,
00295         S_DCCMODE,
00296         S_DGTI,
00297         S_DLAST,
00298     };
00299
00300 #define MAX_IM_VAR  IA_LAST*PHASE_LAST
00301
00302 #define L1_P    IA_POWER
00303 #define L2_P    L1_P+IA_LAST
00304 #define L3_P    L2_P+IA_LAST
00305
00306     struct link_type {
00307         volatile uint32_t iammeter_error, iammeter_count;
00308         volatile uint32_t mqtt_error, mqtt_count;
00309         volatile uint32_t shutdown;
00310     };
00311
00312     struct mode_type {
00313         volatile double error, target, total_system, gti_dumpload, pv_bias, dl_mqtt_max, off_grid,
      sequence;
00314         volatile bool mode, in_pid_control, con0, con1, con2, con3, con4, con5, con6, con7, no_float,
      data_error, bat_crit;
00315         volatile uint32_t mode_tmr;
00316         volatile struct SPid pid;
00317         volatile enum energy_state E;
00318         volatile enum running_state R;
00319     };
00320
```

```
00321     struct energy_type {
00322         volatile double print_vars[MAX_IM_VAR];
00323         volatile double im_vars[IA_LAST][PHASE_LAST];
00324         volatile double mvar[V_DLAST + 1];
00325         volatile bool once_gti, once_ac, iammeter, fm80, dumpload, homeassistant, once_gti_zero;
00326         volatile double gti_low_adj, ac_low_adj, dl_excess_adj;
00327         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;
00328         volatile uint32_t speed_go, im_delay, im_display, gti_delay;
00329         volatile int32_t rc, sane;
00330         volatile uint32_t ten_sec_clock, log_spam, log_time_reset;
00331         pthread_mutex_t ha_lock;
00332         struct mode_type mode;
00333         struct link_type link;
00334         MQTTClient client_p, client_sd, client_ha;
00335     };
00336
00337     extern struct energy_type E;
00338     extern struct ha_flag_type ha_flag_vars_ss;
00339     extern FILE* fout;
00340
00341     void timer_callback(int32_t);
00342     void connlost(void *, char *);
00343
00344     void ramp_up_gti(MQTTClient, bool, bool);
00345     void ramp_up_ac(MQTTClient, bool);
00346     void ramp_down_gti(MQTTClient, bool);
00347     void ramp_down_ac(MQTTClient, bool);
00348     void ha_ac_off(void);
00349     void ha_ac_on(void);
00350     void ha_dc_off(void);
00351     void ha_dc_on(void);
00352
00353     size_t iammeter_write_callback(char *, size_t, size_t, void *);
00354     void iammeter_read(void);
00355     void print_im_vars(void);
00356     void print_mvar_vars(void);
00357
00358     bool sanity_check(void);
00359     char * log_time(bool);
00360     bool sync_ha(void);
00361     bool log_timer(void);
00362
00363 #ifdef __cplusplus
00364 }
00365 #endif
00366
00367 #endif /* BMC_H */
00368
```

4.20 ha_energy/http_vars.c File Reference

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

Include dependency graph for http_vars.c:

Functions

- static void iammeter_get_data (const double, const uint32_t, const uint32_t)
- bool mqtt_gti_time (MQTTClient, const char *, char *)
- size_t iammeter_write_callback (char *buffer, size_t size, size_t nitems, void *stream)
- void iammeter_read (void)
- void print_im_vars (void)

Variables

- static CURL * curl
- static CURLcode res

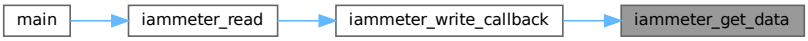
4.20.1 Function Documentation

4.20.1.1 iammeter_get_data()

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

```
00103 {
00104     E.im_vars[i][j] = valuedouble;
00105 }
```

Here is the caller graph for this function:

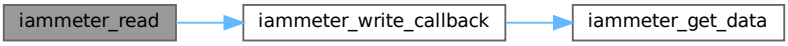


4.20.1.2 iammeter_read()

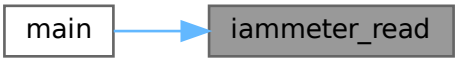
```
void iammeter_read (
            void )
```

```
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 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.20.1.3 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 }
```

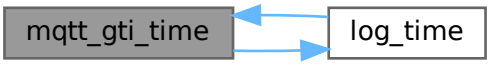
Here is the call graph for this function:

Here is the caller graph for this function:

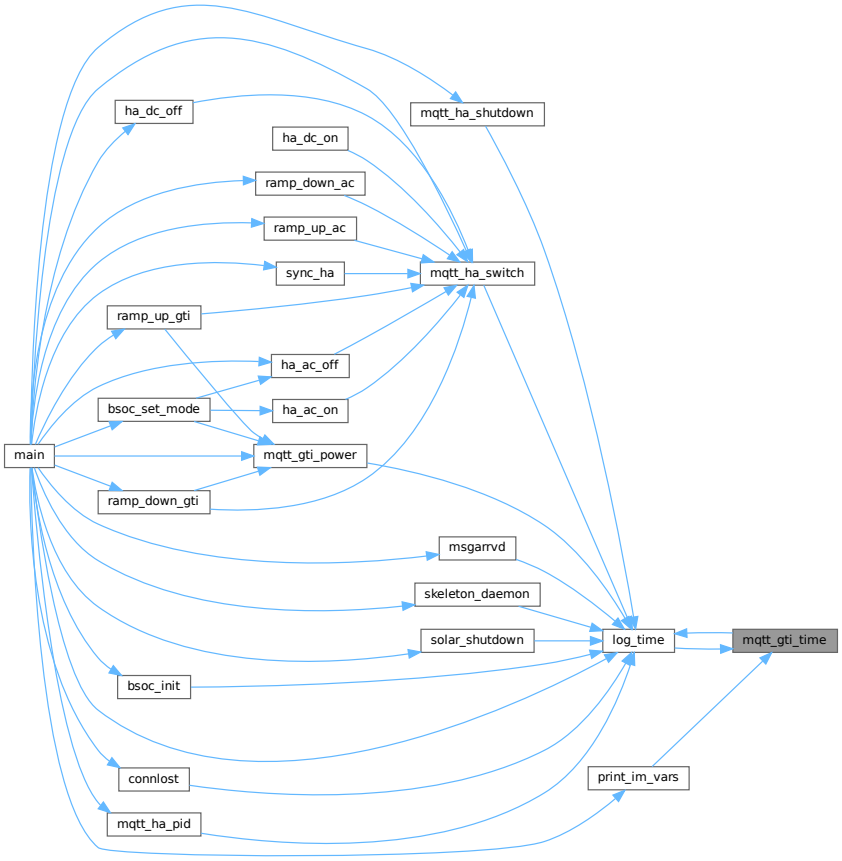
4.20.1.4 mqtt_gti_time()

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

Here is the call graph for this function:



Here is the caller graph for this function:



4.20.1.5 print_im_vars()

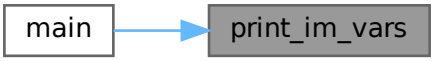
```
void print_im_vars (
            void )
```

```
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 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.20.2 Variable Documentation

4.20.2.1 curl

```
CURL* curl  [static]
```

4.20.2.2 res

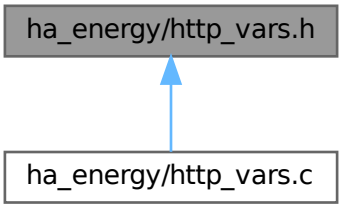
```
CURLcode res  [static]
```

4.21 ha_energy/http_vars.h File Reference

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

Include dependency graph for http_vars.h:

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



Variables

- FILE * fout

4.21.1 Variable Documentation

4.21.1.1 fout

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

4.22 http_vars.h

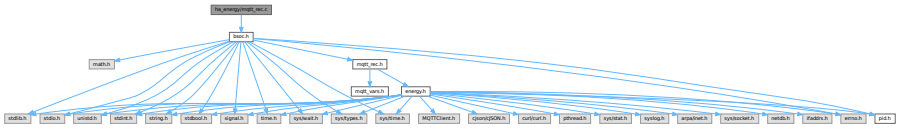
Go to the documentation of this file.

```
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
```

4.23 ha_energy/mqtt_rec.c File Reference

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

Include dependency graph for mqtt_rec.c:



Functions

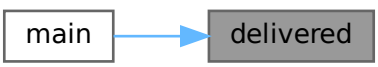
- int32_t msgarrvd (void *context, char *topicName, int topicLen, MQTTClient_message *message)
- void delivered (void *context, MQTTClient_deliveryToken dt)
- bool json_get_data (cJSON *json_src, const char *data_id, cJSON *name, uint32_t i)
- void print_mvar_vars (void)
- bool fm80_float (const bool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

4.23.1 Function Documentation

4.23.1.1 delivered()

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

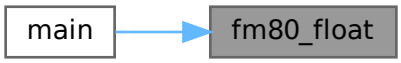
Here is the caller graph for this function:



4.23.1.2 fm80_float()

```
bool fm80_float (
              const bool set_bias)
00247 {
00248     if ((uint32_t) E.mvar[V_FCCM] == FLOAT_CODE) {
00249         if (set_bias) {
00250             E.mode.pv_bias = PV_BIAS_FLOAT;
00251         }
00252         if (E.mode.R != R_IDLE) {
00253             E.mode.R = R_FLOAT;
00254         }
00255         return true;
00256     } else {
00257         if (E.mode.R == R_FLOAT) {
00258             E.mode.R = R_RUN;
00259         }
00260     }
00261     return false;
00262 }
```

Here is the caller graph for this function:



4.23.1.3 fm80_sleep()

```
bool fm80_sleep (
              void )
00269 {
00270     if ((uint32_t) E.mvar[V_FCCM] == SLEEP_CODE) {
00271         return true;
00272     }
00273     return false;
00274 }
```

Here is the caller graph for this function:

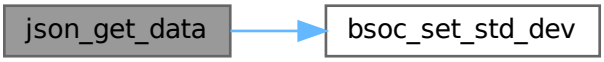
4.23.1.4 json_get_data()

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

```
00138 {
00139     bool ret = false;
00140     static uint32_t j = 0;
00141 
00142     // access the JSON data using the lookup string passed in data_id
00143     name = cJSON_GetObjectItemCaseSensitive(json_src, data_id);
00144 
00145     /*
00146      * process string values
00147      */
00148     if (cJSON_IsString(name) && (name->valuestring != NULL)) {
00149 #ifdef GET_DEBUG
00150         fprintf(fout, "%s Name: %s\n", data_id, name->valuestring);
00151 #endif
00152         ret = true;
00153     }
00154 
00155     /*
00156      * process numeric values
00157      */
00158     if (cJSON_IsNumber(name)) {
00159 #ifdef GET_DEBUG
00160         fprintf(fout, "%s Value: %f\n", data_id, name->valuedouble);
00161 #endif
00162         if (i > V_DLAST) { // check for out-of-range index
00163             i = V_DLAST;
00164         }
00165 
00166         // lock the main value array during updates
00167         pthread_mutex_lock(&E.ha_lock);
00168         E.mvar[i] = name->valuedouble;
00169         pthread_mutex_unlock(&E.ha_lock);
00170 
00171         /*
00172          * special processing for variable data received
00173          */
00174         if (i == V_DCMPPT) {
00175             /*
00176              * load battery current standard deviation array bat_c_std_dev with data
00177              */
00178             bsoc_set_std_dev(E.mvar[i], j++);
00179             if (j >= RDEV_SIZE) {
00180                 j = 0;
00181             }
00182         }
00183         /*
00184          * update local MATTER switch status from HA
00185          */
00186         if (i == V_HDCSW) {
00187             E.gti_sw_status = (bool) ((int32_t) E.mvar[i]);
00188             E.dc_mismatch = false;
00189         }
00190 
00191         if (i == V_HACSW) {
00192             E.ac_sw_status = (bool) ((int32_t) E.mvar[i]);
00193             E.ac_mismatch = false;
00194         }
00195 
00196         // command HA_ENERGY to shutdown mode
00197         if (i == V_HSHUT) {
00198             E.solar_shutdown = (bool) ((int32_t) E.mvar[i]);
00199         }
00200         // set HA_ENERGY energy processing mode
00201         if (i == V_HMODE) {
00202             ha_flag_vars_ss.energy_mode = (bool) ((int32_t) E.mvar[i]);
00203         }
00204         if (i == V_HCON0) {
00205             E.mode.con0 = (bool) ((int32_t) E.mvar[i]);
00206         }
00207         if (i == V_HCON1) {
00208             E.mode.con1 = (bool) ((int32_t) E.mvar[i]);
00209         }
00210         if (i == V_HCON2) {
00211             E.mode.con2 = (bool) ((int32_t) E.mvar[i]);
00212         }
00213         if (i == V_HCON3) {
```

```
00214             E.mode.con3 = (bool) ((int32_t) E.mvar[i]);
00215         }
00216         if (i == V_HCON4) { // set DL GTI excess load MODE
00217             E.mode.con4 = (bool) ((int32_t) E.mvar[i]);
00218         }
00219         if (i == V_HCON5) { // clear DL GTI excess load MODE
00220             E.mode.con5 = (bool) ((int32_t) E.mvar[i]);
00221         }
00222         if (i == V_HCON6) { // HA Energy program idle
00223             E.mode.con6 = (bool) ((int32_t) E.mvar[i]);
00224         }
00225         if (i == V_HCON7) { // HA Energy program exit
00226             E.mode.con7 = (bool) ((int32_t) E.mvar[i]);
00227         }
00228         ret = true;
00229     }
00230     return ret;
00231 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

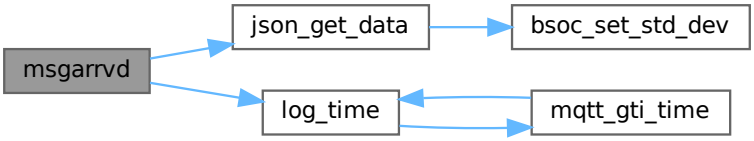
4.23.1.5 msgarrvd()

```
int32_t msgarrvd (
            void * context,
            char * topicName,
            int topicLen,
            MQTTClient_message * message)
00008 {
00009     int32_t i, ret = 1;
00010     const char* payloadptr;
00011     char buffer[MBMQTT];
00012     struct ha_flag_type *ha_flag = context;
00013
00014     E.link.mqtt_count++;
00015     // bug-out if no context variables passed to callback
00016     if (context == NULL) {
00017         ret = -1;
00018         goto null_exit;
00019     }
00020
00021 #ifdef DEBUG_REC
```

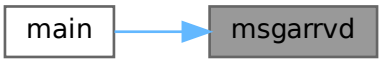
```
00022     fprintf(fout, "Message arrived\n");
00023 #endif
00024     /*
00025      * move the received message into a processing holding buffer
00026      */
00027     payloadptr = message->payload;
00028     for (i = 0; i < message->payloadlen; i++) {
00029         buffer[i] = *payloadptr++;
00030     }
00031     buffer[i] = 0; // make a null terminated C string
00032
00033     // parse the JSON data in the holding buffer
00034     cJSON *json = cJSON_ParseWithLength(buffer, message->payloadlen);
00035     if (json == NULL) {
00036         const char *error_ptr = cJSON_GetErrorPtr();
00037         if (error_ptr != NULL) {
00038             fprintf(fout, "%s Error: %s NULL cJSON pointer\n", log_time(false), error_ptr);
00039         }
00040         ret = -1;
00041         ha_flag->rec_ok = false;
00042         E.fm80 = false;
00043         E.dumpload = false;
00044         E.homeassistant = false;
00045         E.link.mqtt_error++;
00046         goto error_exit;
00047     }
00048
00049     /*
00050      * MQTT messages from the FM80 Q84 interface
00051      */
00052     if (ha_flag->ha_id == FM80_ID) {
00053 #ifdef DEBUG_REC
00054         fprintf(fout, "FM80 MQTT data\r\n");
00055 #endif
00056         cJSON *data_result = json;
00057
00058         for (uint32_t ii = V_FCCM; ii < V_FLAST; ii++) {
00059             if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
00060                 ha_flag->var_update++;
00061             }
00062         }
00063         E.fm80 = true;
00064     }
00065
00066     /*
00067      * MQTT messages from the K42 dumpload/gti interface
00068      */
00069     if (ha_flag->ha_id == DUMPLOAD_ID) {
00070 #ifdef DEBUG_REC
00071         fprintf(fout, "DUMPLOAD MQTT data\r\n");
00072 #endif
00073         cJSON *data_result = json;
00074
00075         for (uint32_t ii = V_HDCSW; ii < V_DLAST; ii++) {
00076             if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
00077                 ha_flag->var_update++;
00078             }
00079         }
00080         E.dumpload = true;
00081     }
00082
00083     /*
00084      * MQTT messages from the Linux HA_ENERGY interface
00085      */
00086     if (ha_flag->ha_id == HA_ID) {
00087 #ifdef DEBUG_REC
00088         fprintf(fout, "Home Assistant MQTT data\r\n");
00089 #endif
00090         cJSON *data_result = json;
00091
00092         if (json_get_data(json, mqtt_name[V_HACSW], data_result, V_HACSW)) {
00093             ha_flag->var_update++;
00094         }
00095         data_result = json;
00096         if (json_get_data(json, mqtt_name[V_HDCSW], data_result, V_HDCSW)) {
00097             ha_flag->var_update++;
00098         }
00099
00100         E.homeassistant = true;
00101     }
00102
00103     // done with processing MQTT async message, set state flags
00104     ha_flag->receivedtoken = true;
00105     ha_flag->rec_ok = true;
00106     /*
00107      * exit and delete/free resources. In steps depending of possible error conditions
00108      */
```

```
00109 error_exit:
00110     // delete the JSON object
00111     cJSON_Delete(json);
00112 null_exit:
00113     // free the MQTT objects
00114     MQTTClient_freeMessage(&message);
00115     MQTTClient_free(topicName);
00116     return ret;
00117 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.23.1.6 print_mvar_vars()

```
void print_mvar_vars (
            void )
00237 {
00238     fprintf(fout, ", AC Inverter %7.2fW, BAT Energy %7.2fWh, Solar E %7.2fWh, AC E %7.2fWh, PERR
      %7.2fW, PBAL %7.2fW, ST %7.2f, GDL %7.2f, %d,%d,%d %d,%d,%d\r",
00239         E.mvar[V_FLO], E.mvar[V_FBEKW], E.mvar[V_FSO], E.mvar[V_FACE], E.mode.error, E.mvar[V_BEN],
      E.mode.total_system, E.mode.gti_dumpload, (int32_t) E.mvar[V_HDCSW], (int32_t) E.mvar[V_HACSW],
      (int32_t) E.mvar[V_HMODE],
00240         (int32_t) E.dc_mismatch, (int32_t) E.ac_mismatch, (int32_t) E.mode_mismatch);
00241 }
```

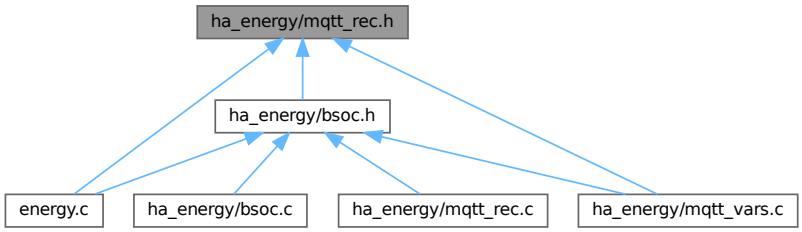
Here is the caller graph for this function:

4.24 ha_energy/mqtt_rec.h File Reference

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

Include dependency graph for mqtt_rec.h:

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



Data Structures

- struct ha_flag_type

Macros

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

Enumerations

- enum mqtt_id {
  P8055_ID , FM80_ID , DUMPLOAD_ID , HA_ID ,
  LAST_MQTT_ID }

Functions

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

Variables

- FILE * fout

4.24.1 Macro Definition Documentation

4.24.1.1 FLOAT_CODE

```
#define FLOAT_CODE 1
```

4.24.1.2 MBMQTT

```
#define MBMQTT 1024
```

4.24.1.3 RDEV_SIZE

```
#define RDEV_SIZE 10
```

4.24.1.4 SLEEP_CODE

```
#define SLEEP_CODE 0
```

4.24.2 Enumeration Type Documentation

4.24.2.1 mqtt_id

```
enum mqtt_id
```

Enumerator

P8055_ID	
FM80_ID	
DUMPLOAD_ID	
HA_ID	
LAST_MQTT_ID	

```
00027                 {
00028         P8055_ID,
00029         FM80_ID,
00030         DUMPLOAD_ID,
00031     HA_ID,
00032         LAST_MQTT_ID,
00033     };
```

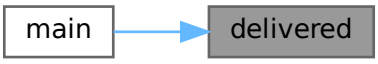
4.24.3 Function Documentation

4.24.3.1 delivered()

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

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

Here is the caller graph for this function:

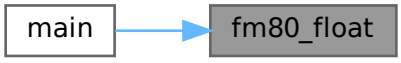


4.24.3.2 fm80_float()

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

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

Here is the caller graph for this function:

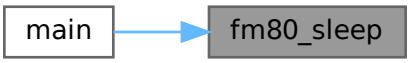


4.24.3.3 fm80_sleep()

```
bool fm80_sleep (
            void )
```

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

Here is the caller graph for this function:



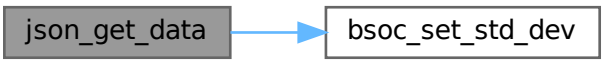
4.24.3.4 json_get_data()

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

```
00138 {
00139     bool ret = false;
00140     static uint32_t j = 0;
00141 
00142     // access the JSON data using the lookup string passed in data_id
00143     name = cJSON_GetObjectItemCaseSensitive(json_src, data_id);
00144 
00145     /*
00146      * process string values
00147      */
00148     if (cJSON_IsString(name) && (name->valuestring != NULL)) {
00149 #ifdef GET_DEBUG
00150         fprintf(fout, "%s Name: %s\n", data_id, name->valuestring);
00151 #endif
00152         ret = true;
00153     }
00154 
00155     /*
00156      * process numeric values
00157      */
00158     if (cJSON_IsNumber(name)) {
00159 #ifdef GET_DEBUG
00160         fprintf(fout, "%s Value: %f\n", data_id, name->valuedouble);
00161 #endif
00162         if (i > V_DLAST) { // check for out-of-range index
00163             i = V_DLAST;
00164         }
00165 
00166         // lock the main value array during updates
00167         pthread_mutex_lock(&E.ha_lock);
00168         E.mvar[i] = name->valuedouble;
00169         pthread_mutex_unlock(&E.ha_lock);
00170 
00171         /*
00172          * special processing for variable data received
00173          */
00174         if (i == V_DCMPPT) {
00175             /*
00176              * load battery current standard deviation array bat_c_std_dev with data
```

```
00177                */
00178               bsoc_set_std_dev(E.mvar[i], j++);
00179               if (j >= RDEV_SIZE) {
00180                   j = 0;
00181               }
00182           }
00183           /*
00184            * update local MATTER switch status from HA
00185            */
00186           if (i == V_HDCSW) {
00187               E.gti_sw_status = (bool) ((int32_t) E.mvar[i]);
00188               E.dc_mismatch = false;
00189           }
00190
00191           if (i == V_HACSW) {
00192               E.ac_sw_status = (bool) ((int32_t) E.mvar[i]);
00193               E.ac_mismatch = false;
00194           }
00195
00196           // command HA_ENERGY to shutdown mode
00197           if (i == V_HSHUT) {
00198               E.solar_shutdown = (bool) ((int32_t) E.mvar[i]);
00199           }
00200           // set HA_ENERGY energy processing mode
00201           if (i == V_HMODE) {
00202               ha_flag_vars_ss.energy_mode = (bool) ((int32_t) E.mvar[i]);
00203           }
00204           if (i == V_HCON0) {
00205               E.mode.con0 = (bool) ((int32_t) E.mvar[i]);
00206           }
00207           if (i == V_HCON1) {
00208               E.mode.con1 = (bool) ((int32_t) E.mvar[i]);
00209           }
00210           if (i == V_HCON2) {
00211               E.mode.con2 = (bool) ((int32_t) E.mvar[i]);
00212           }
00213           if (i == V_HCON3) {
00214               E.mode.con3 = (bool) ((int32_t) E.mvar[i]);
00215           }
00216           if (i == V_HCON4) { // set DL GTI excess load MODE
00217               E.mode.con4 = (bool) ((int32_t) E.mvar[i]);
00218           }
00219           if (i == V_HCON5) { // clear DL GTI excess load MODE
00220               E.mode.con5 = (bool) ((int32_t) E.mvar[i]);
00221           }
00222           if (i == V_HCON6) { // HA Energy program idle
00223               E.mode.con6 = (bool) ((int32_t) E.mvar[i]);
00224           }
00225           if (i == V_HCON7) { // HA Energy program exit
00226               E.mode.con7 = (bool) ((int32_t) E.mvar[i]);
00227           }
00228           ret = true;
00229       }
00230       return ret;
00231 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

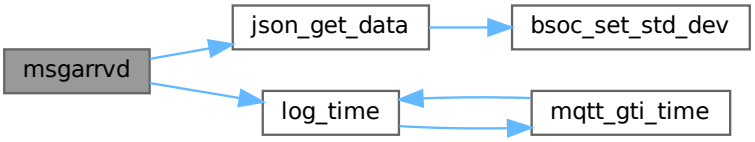
4.24.3.5 msgarrvd()

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

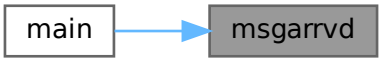
```
00008 {
00009     int32_t i, ret = 1;
00010     const char* payloadptr;
00011     char buffer[MBMQTT];
00012     struct ha_flag_type *ha_flag = context;
00013 
00014     E.link.mqtt_count++;
00015     // bug-out if no context variables passed to callback
00016     if (context == NULL) {
00017   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

```
00062         }
00063         E.fm80 = true;
00064     }
00065
00066     /*
00067      * MQTT messages from the K42 dumpload/gti interface
00068      */
00069     if (ha_flag->ha_id == DUMPLOAD_ID) {
00070 #ifdef DEBUG_REC
00071         fprintf(fout, "DUMPLOAD MQTT data\r\n");
00072 #endif
00073         cJSON *data_result = json;
00074
00075         for (uint32_t ii = V_HDCSW; ii < V_DLAST; ii++) {
00076             if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
00077                 ha_flag->var_update++;
00078             }
00079         }
00080         E.dumpload = true;
00081     }
00082
00083     /*
00084      * MQTT messages from the Linux HA_ENERGY interface
00085      */
00086     if (ha_flag->ha_id == HA_ID) {
00087 #ifdef DEBUG_REC
00088         fprintf(fout, "Home Assistant MQTT data\r\n");
00089 #endif
00090         cJSON *data_result = json;
00091
00092         if (json_get_data(json, mqtt_name[V_HACSW], data_result, V_HACSW)) {
00093             ha_flag->var_update++;
00094         }
00095         data_result = json;
00096         if (json_get_data(json, mqtt_name[V_HDCSW], data_result, V_HDCSW)) {
00097             ha_flag->var_update++;
00098         }
00099
00100         E.homeassistant = true;
00101     }
00102
00103     // done with processing MQTT async message, set state flags
00104     ha_flag->receivedtoken = true;
00105     ha_flag->rec_ok = true;
00106     /*
00107      * exit and delete/free resources. In steps depending of possible error conditions
00108      */
00109 error_exit:
00110     // delete the JSON object
00111     cJSON_Delete(json);
00112 null_exit:
00113     // free the MQTT objects
00114     MQTTClient_freeMessage(&message);
00115     MQTTClient_free(topicName);
00116     return ret;
00117 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.24.4 Variable Documentation

4.24.4.1 fout

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

4.25 mqtt_rec.h

Go to the documentation of this file.

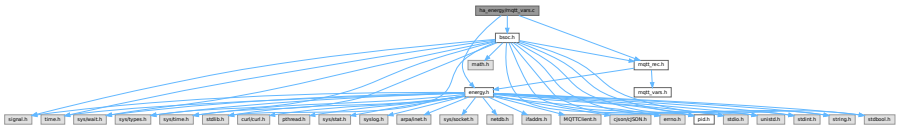
```
00001 /*
00002  * File:   mqtt_rec.h
00003  * Author: root
00004  *
00005  * Created on February 5, 2024, 2:54 PM
00006  */
00007
00008 #ifndef MQTT_REC_H
00009 #define MQTT_REC_H
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014
00015 #include "energy.h"
00016 #include "mqtt_vars.h"
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     };
00041
00042     extern FILE* fout;
00043
00044     int32_t msgarrvd(void *, char *, int, MQTTClient_message *);
00045     void delivered(void *, MQTTClient_deliveryToken);
00046
00047     bool json_get_data(cJSON *, const char *, cJSON *, uint32_t);
```

```
00048     bool fm80_float(const bool set_bias);
00049     bool fm80_sleep(void);
00050
00051 #ifdef __cplusplus
00052 }
00053 #endif
00054
00055 #endif /* MQTT_REC_H */
00056
```

4.26 ha_energy/mqtt_vars.c File Reference

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

Include dependency graph for mqtt_vars.c:



Macros

- #define _DEFAULT_SOURCE

Functions

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

Variables

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

4.26.1 Macro Definition Documentation

4.26.1.1 _DEFAULT_SOURCE

```
#define _DEFAULT_SOURCE
```

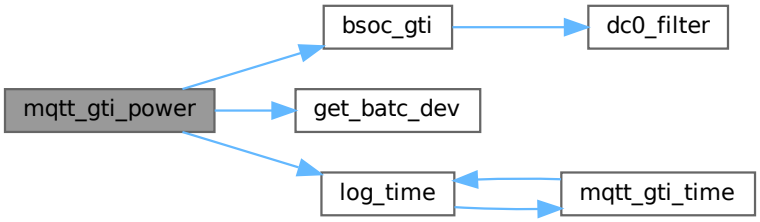
4.26.2 Function Documentation

4.26.2.1 mqtt_gti_power()

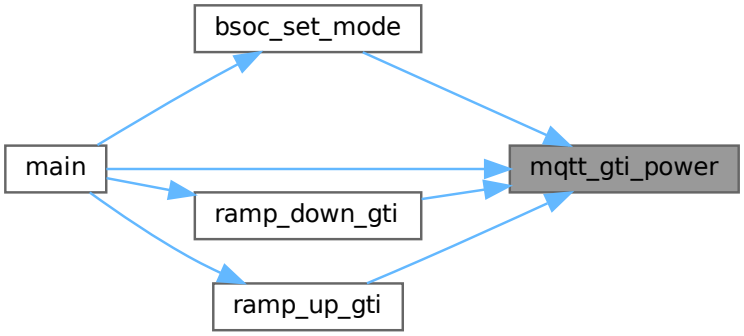
```
bool mqtt_gti_power (
            MQTTClient client_p,
            const char * topic_p,
            char * msg,
            uint32_t trace)
```

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

Here is the call graph for this function:



Here is the caller graph for this function:



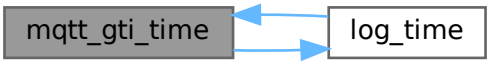
4.26.2.2 mqtt_gti_time()

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

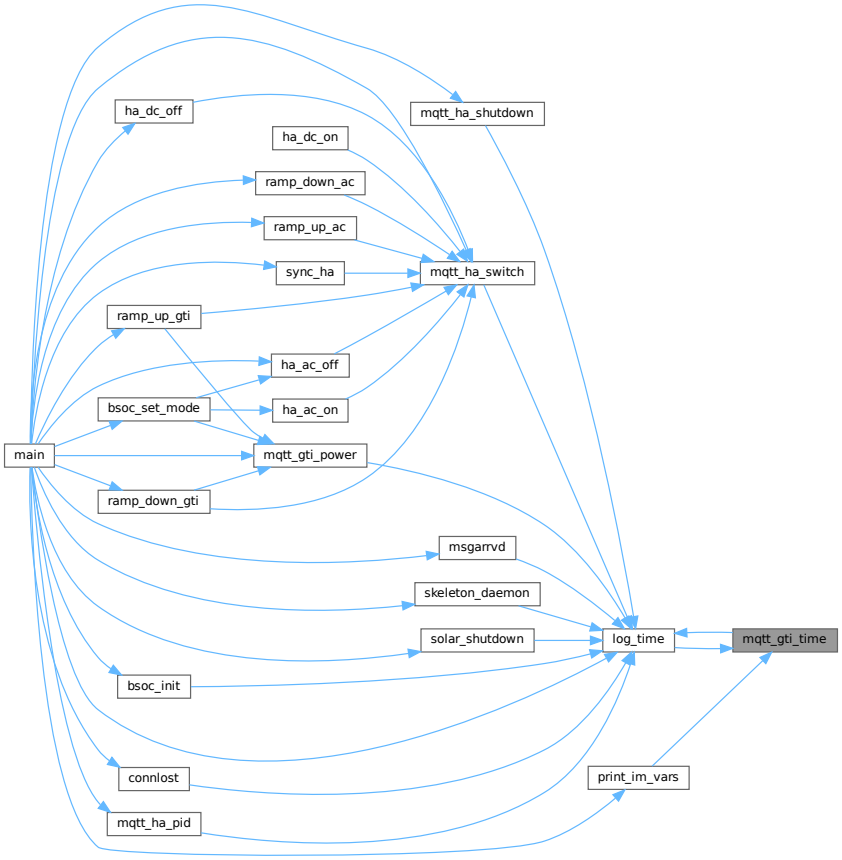
```
00249 {
00250     bool ret = true;
00251     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00252     MQTTClient_deliveryToken token;
00253     ha_flag_vars_ss.deliveredtoken = 0;
00254 
00255     E.link.mqtt_count++;
00256     pubmsg.payload = msg;
00257     pubmsg.payloadlen = strlen(msg);
00258     pubmsg.qos = QOS;
00259     pubmsg.retained = 0;
00260 
00261     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run time commands
00262 
00263     // a busy, wait loop for the async delivery thread to complete
00264     {
00265         uint32_t waiting = 0;
00266         while (ha_flag_vars_ss.deliveredtoken != token) {
```

```
00267             usleep(GTI_TOKEN_DELAY);
00268             if (waiting++ > MQTT_TIMEOUT) {
00269                 fprintf(fout, "\r\n%s GTI Time Still Waiting, timeout\r\n", log_time(false));
00270                 break;
00271             }
00272         };
00273     }
00274     usleep(HA_SW_DELAY);
00275     return ret;
00276 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

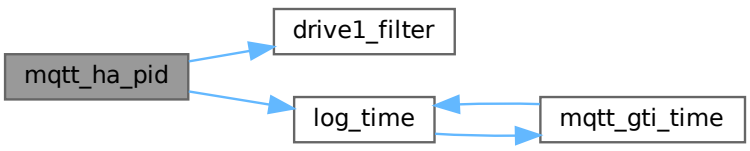


4.26.2.3 mqtt_ha_pid()

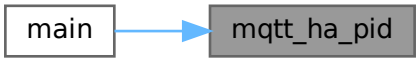
```
void mqtt_ha_pid (
            MQTTClient client_p,
            const char * topic_p)
```

```
00046 {
00047     cJSON *json;
00048     time_t rawtime;
00049
00050     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00051     MQTTClient_deliveryToken token;
00052     ha_flag_vars_ss.deliveredtoken = 0;
00053
00054     E.link.mqtt_count++;
00055     E.mode.sequence++;
00056     json = cJSON_CreateObject();
00057     cJSON_AddStringToObject(json, "name", CLIENTID1);
00058     cJSON_AddNumberToObject(json, "sequence", E.mode.sequence);
00059     cJSON_AddNumberToObject(json, "mqtt_count", (double) E.link.mqtt_count);
00060     cJSON_AddNumberToObject(json, "http_count", (double) E.link.iammeter_count);
00061     cJSON_AddNumberToObject(json, "piderror", E.mode.error);
00062     cJSON_AddNumberToObject(json, "totalsystem", E.mode.total_system);
00063     cJSON_AddNumberToObject(json, "gtinet", E.mode.gti_dumpload);
00064     cJSON_AddNumberToObject(json, "energy_state", (double) E.mode.E);
00065     cJSON_AddNumberToObject(json, "run_state", (double) E.mode.R);
00066     // correct for power sensed by GTI metering
00067     E.mode.off_grid = (E.mvar[V_FLO] - (E.mvar[V_DPPV] * DL_AC_DC_EFF));
00068     E.mode.off_grid = drive1_filter(E.mode.off_grid);
00069     if (E.mode.off_grid < 0.0f) { // only see power removed from grid usage
00070         E.mode.off_grid = 0.0f;
00071     }
00072     cJSON_AddNumberToObject(json, "off_grid", E.mode.off_grid);
00073     cJSON_AddNumberToObject(json, "excess_mode", (double) E.dl_excess);
00074     cJSON_AddStringToObject(json, "build_date", FW_Date);
00075     cJSON_AddStringToObject(json, "build_time", FW_Time);
00076     time(&rawtime);
00077     cJSON_AddNumberToObject(json, "sequence_time", (double) rawtime);
00078     // convert the cJSON object to a JSON string
00079     char *json_str = cJSON_Print(json);
00080
00081     pubmsg.payload = json_str;
00082     pubmsg.payloadlen = strlen(json_str);
00083     pubmsg.qos = QOS;
00084     pubmsg.retained = 0;
00085
00086     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
00087     // a busy, wait loop for the async delivery thread to complete
00088     {
00089         uint32_t waiting = 0;
00090         while (ha_flag_vars_ss.deliveredtoken != token) {
00091             usleep(TOKEN_DELAY);
00092             if (waiting++ > MQTT_TIMEOUT) {
00093                 fprintf(fout, "\r\n%s SW Still Waiting, timeout\r\n", log_time(false));
00094                 break;
00095             }
00096         };
00097     }
00098
00099     cJSON_free(json_str);
00100     cJSON_Delete(json);
00101 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

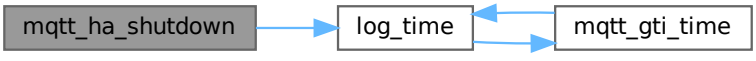


4.26.2.4 mqtt_ha_shutdown()

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

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

Here is the call graph for this function:



Here is the caller graph for this function:

4.26.2.5 mqtt_ha_switch()

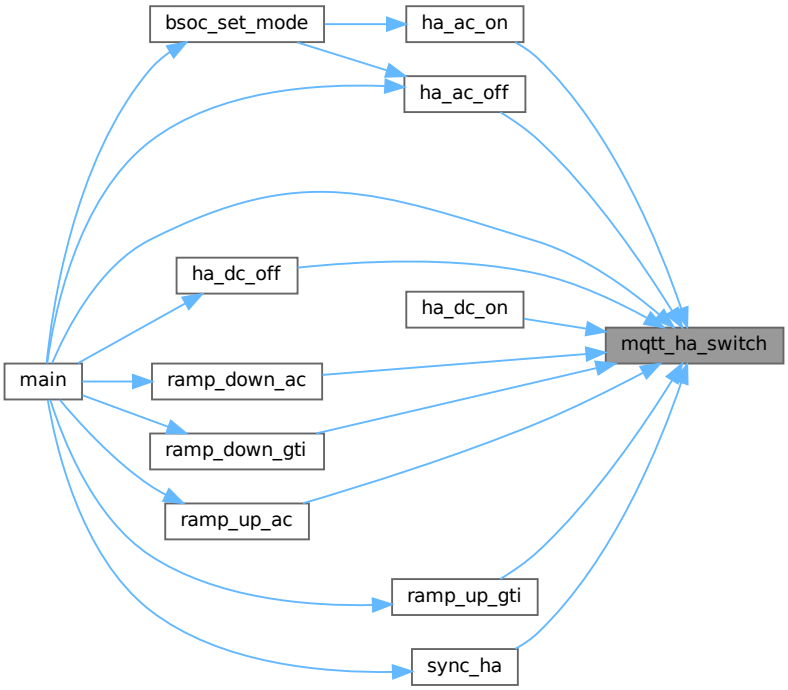
```
void mqtt_ha_switch (
            MQTTClient client_p,
            const char * topic_p,
            const bool sw_state)
```

```
00107 {
00108     cJSON *json;
00109 #ifdef DEBUG_HA_CMD
00110     static bool spam = false;
00111     static uint32_t less_spam = 0;
00112 #endif
00113
00114     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00115     MQTTClient_deliveryToken token;
00116     ha_flag_vars_ss.deliveredtoken = 0;
00117
00118     E.link.mqtt_count++;
00119     json = cJSON_CreateObject();
00120     if (sw_state) {
00121         cJSON_AddStringToObject(json, "state", "ON");
00122 #ifdef DEBUG_HA_CMD
00123         spam = true;
00124         less_spam = 0;
00125 #endif
00126     } else {
00127         if ((uint32_t) E.mvar[V_FCCM] != FLOAT_CODE) { // use max power in FLOAT mode
00128             cJSON_AddStringToObject(json, "state", "OFF");
00129         } else {
00130             cJSON_AddStringToObject(json, "state", "ON");
00131 #ifdef DEBUG_HA_CMD
00132             spam = true;
00133             less_spam = 0;
00134 #endif
00135         }
00136     }
00137     // convert the cJSON object to a JSON string
00138     char *json_str = cJSON_Print(json);
00139
00140     pubmsg.payload = json_str;
00141     pubmsg.payloadlen = strlen(json_str);
00142     pubmsg.qos = QOS;
00143     pubmsg.retained = 0;
00144
00145 #ifdef DEBUG_HA_CMD
00146     if (spam) {
00147         log_time(true);
00148         fflush(fout);
00149         fprintf(fout, "HA switch command %s, %d\r\n", topic_p, sw_state);
00150         fflush(fout);
00151         if (!sw_state) {
00152             if (less_spam++ > 3) {
00153                 spam = false;
00154                 less_spam = 0;
00155             }
00156         }
00157     }
00158 #endif
00159
00160     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
00161     // a busy, wait loop for the async delivery thread to complete
00162     {
```

```
00163         uint32_t waiting = 0;
00164         while (ha_flag_vars_ss.deliveredtoken != token) {
00165             usleep(TOKEN_DELAY);
00166             if (waiting++ > MQTT_TIMEOUT) {
00167 #ifdef DEBUG_HA_CMD
00168                 fflush(fout);
00169                 fprintf(fout, "\r\nSW Still Waiting, timeout\r\n");
00170                 fflush(fout);
00171 #endif
00172                 break;
00173             }
00174         };
00175     }
00176 
00177     cJSON_free(json_str);
00178     cJSON_Delete(json);
00179     usleep(HA_SW_DELAY);
00180     fflush(fout);
00181 }
```

Here is the call graph for this function:

Here is the caller graph for this function:



4.26.3 Variable Documentation

4.26.3.1 FW_Date

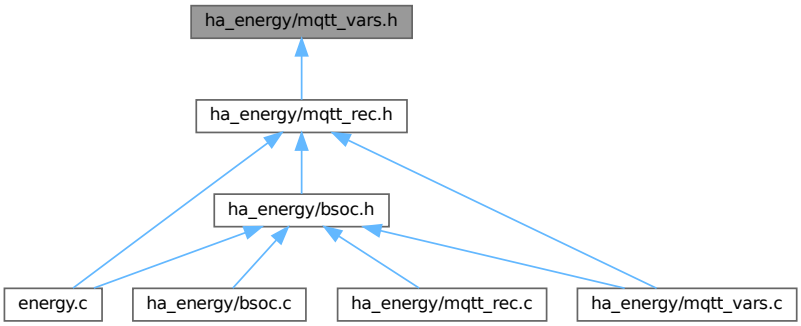
```
const char* const FW_Date = __DATE__  [static]
```

4.26.3.2 FW_Time

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

4.27 ha_energy/mqtt_vars.h File Reference

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



Macros

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

Functions

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

Variables

- const char * mqtt_name [V_DLAST]

4.27.1 Macro Definition Documentation

4.27.1.1 GTI_TOKEN_DELAY

```
#define GTI_TOKEN_DELAY 300
```

4.27.1.2 HA_SW_DELAY

```
#define HA_SW_DELAY 400000
```

4.27.1.3 QOS

```
#define QOS 1
```

4.27.1.4 TOKEN_DELAY

```
#define TOKEN_DELAY 250
```

4.27.2 Function Documentation

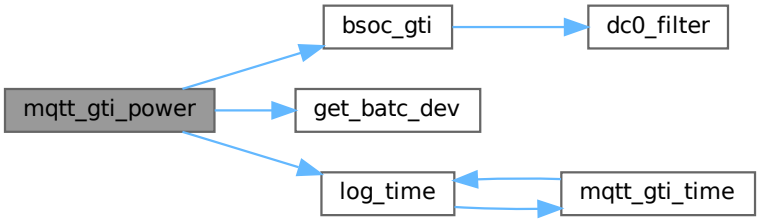
4.27.2.1 mqtt_gti_power()

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

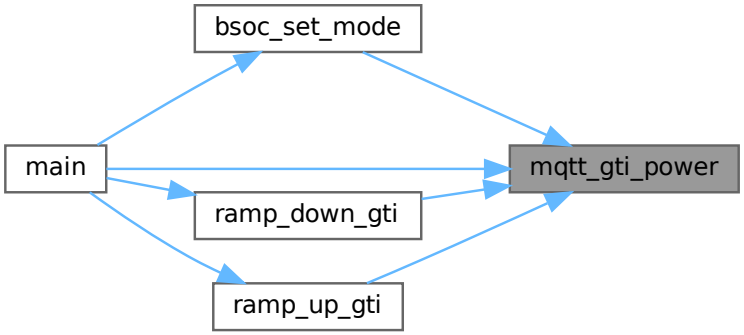
```
00187 {
00188     bool ret = true;
00189     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00190     MQTTClient_deliveryToken token;
00191     ha_flag_vars_ss.deliveredtoken = 0;
00192     static bool spam = false;
00193 
00194     E.link.mqtt_count++;
00195     pubmsg.payload = msg;
00196     pubmsg.payloadlen = strlen(msg);
00197     pubmsg.qos = QOS;
00198     pubmsg.retained = 0;
00199 
00200     if (E.dl_excess) { // always run excess commands
00201         spam = false;
00202     }
00203 #ifdef GTI_NO_POWER
00204     MQTTClient_publishMessage(client_p, "mateq84/data/gticmd_nopower", &pubmsg, &token);
00205 #else
00206     if (bsoc_gti() > MIN_BAT_KW || E.dl_excess) {
00207 #ifdef DEBUG_HA_CMD
00208         log_time(true);
00209         fprintf(fout, "HA GTI power command %s, SDEV %5.2f trace %u\r\n", msg, get_batc_dev(), trace);
00210         fflush(fout);
00211         spam = true;
00212 #endif
00213         MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run power commands
00214     } else {
00215         ret = false;
00216         pubmsg.payload = "Z#";
00217         pubmsg.payloadlen = strlen("Z#");
00218         if (!spam) {
00219             MQTTClient_publishMessage(client_p, TOPIC_SPAM, &pubmsg, &token);
```

```
00220         } else {
00221             MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // only shutdown GTI power
00222         }
00223 #ifdef DEBUG_HA_CMD
00224         if (spam) {
00225             log_time(true);
00226             fprintf(fout, "HA GTI power set to zero, trace %u\r\n", trace);
00227             fflush(fout);
00228             spam = false;
00229         }
00230 #endif
00231     }
00232 #endif
00233     // a busy, wait loop for the async delivery thread to complete
00234     {
00235         uint32_t waiting = 0;
00236         while (ha_flag_vars_ss.deliveredtoken != token) {
00237             usleep(TOKEN_DELAY);
00238             if (waiting++ > MQTT_TIMEOUT) {
00239                 fprintf(fout, "\r\n%s GTI Power Still Waiting, timeout\r\n", log_time(false));
00240                 break;
00241             }
00242         };
00243     }
00244     usleep(HA_SW_DELAY);
00245     return ret;
00246 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.27.2.2 mqtt_gti_time()

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

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

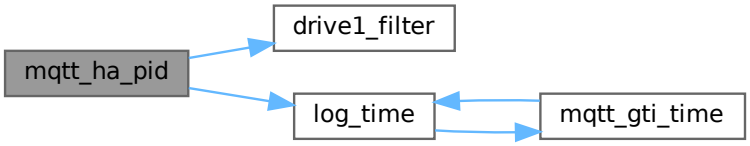
4.27.2.3 mqtt_ha_pid()

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

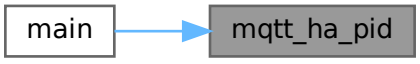
```
00046 {
00047     cJSON *json;
00048     time_t rawtime;
00049
00050     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00051     MQTTClient_deliveryToken token;
00052     ha_flag_vars_ss.deliveredtoken = 0;
00053
00054     E.link.mqtt_count++;
00055     E.mode.sequence++;
00056    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

```
00083     pubmsg.qos = QOS;
00084     pubmsg.retained = 0;
00085
00086     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
00087     // a busy, wait loop for the async delivery thread to complete
00088     {
00089         uint32_t waiting = 0;
00090         while (ha_flag_vars_ss.deliveredtoken != token) {
00091             usleep(TOKEN_DELAY);
00092             if (waiting++ > MQTT_TIMEOUT) {
00093                 fprintf(fout, "\r\n%s SW Still Waiting, timeout\r\n", log_time(false));
00094                 break;
00095             }
00096         };
00097     }
00098
00099     cJSON_free(json_str);
00100     cJSON_Delete(json);
00101 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

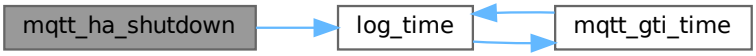


4.27.2.4 mqtt_ha_shutdown()

```
void mqtt_ha_shutdown (
            MQTTClient client_p,
            const char * topic_p)
00013 {
00014     cJSON *json;
00015     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00016     MQTTClient_deliveryToken token;
00017     ha_flag_vars_ss.deliveredtoken = 0;
00018
00019     json = cJSON_CreateObject();
00020     cJSON_AddStringToObject(json, "shutdown", CLIENTID1);
00021     char *json_str = cJSON_Print(json);
00022
00023     pubmsg.payload = json_str;
00024     pubmsg.payloadlen = strlen(json_str);
00025     pubmsg.qos = QOS;
```

```
00026     pubmsg.retained = 0;
00027
00028     MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
00029     // a busy, wait loop for the async delivery thread to complete
00030     {
00031         uint32_t waiting = 0;
00032         while (ha_flag_vars_ss.deliveredtoken != token) {
00033             usleep(TOKEN_DELAY);
00034             if (waiting++ > MQTT_TIMEOUT) {
00035                 fprintf(fout, "\r\n%s SW Still Waiting, timeout\r\n", log_time(false));
00036                 break;
00037             }
00038         };
00039     }
00040 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.27.2.5 mqtt_ha_switch()

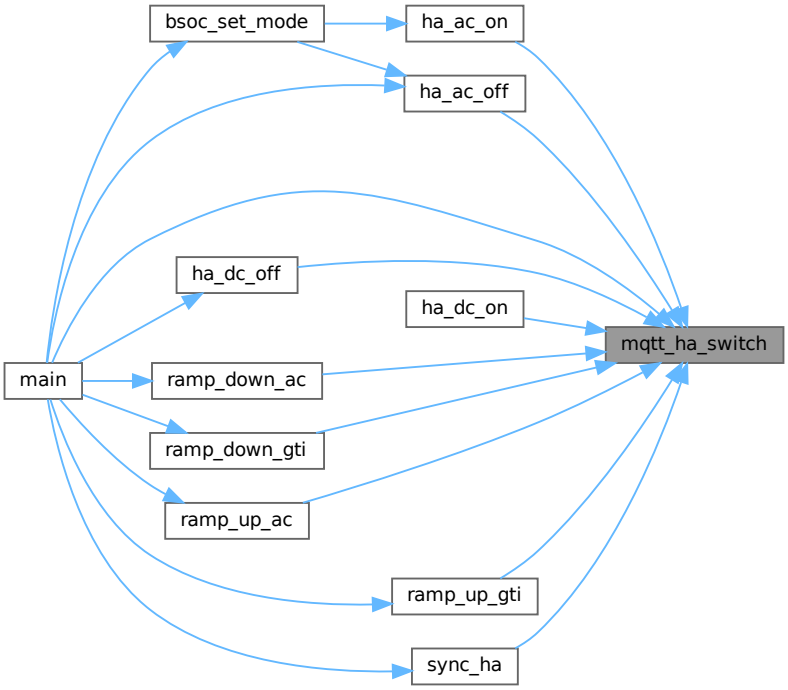
```
void mqtt_ha_switch (
             MQTTClient client_p,
             const char * topic_p,
             const bool sw_state)
```

```
00107 {
00108     cJSON *json;
00109 #ifdef DEBUG_HA_CMD
00110     static bool spam = false;
00111     static uint32_t less_spam = 0;
00112 #endif
00113
00114     MQTTClient_message pubmsg = MQTTClient_message_initializer;
00115     MQTTClient_deliveryToken token;
00116     ha_flag_vars_ss.deliveredtoken = 0;
00117
00118     E.link.mqtt_count++;
00119     json = cJSON_CreateObject();
00120     if (sw_state) {
00121         cJSON_AddStringToObject(json, "state", "ON");
00122 #ifdef DEBUG_HA_CMD
00123         spam = true;
00124         less_spam = 0;
00125 #endif
00126     } else {
00127         if ((uint32_t) E.mvar[V_FCCM] != FLOAT_CODE) { // use max power in FLOAT mode
```

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

Here is the call graph for this function:

Here is the caller graph for this function:



4.27.3 Variable Documentation

4.27.3.1 mqtt_name

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

```
00003                                   {
00004     "pccmode",
00005     "batenergykw",
00006     "runtime",
00007     "bamps",
00008     "bvolts",
00009     "load",
00010     "solar",
00011     "acenergy",
00012     "benergy",
00013     "pwatts",
00014     "pamps",
00015     "pvolts",
00016     "flast",
00017     "HAdcsw",
00018     "HAacsw",
00019     "HAshut",
00020     "HAmode",
00021     "HAcon0",
00022     "HAcon1",
00023     "HAcon2",
00024     "HAcon3",
00025     "HAcon4",
00026     "HAcon5",
00027     "HAcon6",
00028     "HAcon7",
00029     "DLv_pv",
00030     "DLp_pv",
00031     "DLp_bat",
00032     "DLv_bat",
00033     "DLc_mppt",
```

```
00034     "DLp_mppt",
00035     "DLah_bat",
00036     "DLccmode",
00037     "DLgti",
00038 };
```

4.28 mqtt_vars.h

Go to the documentation of this file.

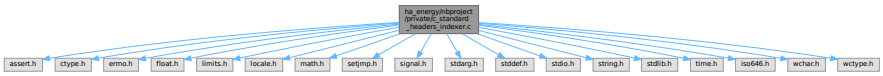
```
00001 /*
00002  * File:   mqtt_vars.h
00003  * Author: root
00004  *
00005  * Created on February 9, 2024, 6:50 AM
00006  */
00007
00008 #ifndef MQTT_VARS_H
00009 #define MQTT_VARS_H
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014
00015     //#define GTI_NO_POWER      // do we 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
```

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

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

```
#include <wchar.h>
#include <wctype.h>
```

Include dependency graph for c_standard_headers_indexer.c:



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

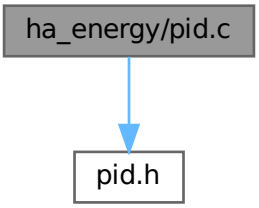
```
#include <cstdlib>
#include <csignal>
#include <csetjmp>
#include <cstdarg>
#include <typeinfo>
#include <bitset>
#include <functional>
#include <utility>
#include <ctime>
#include <cstddef>
#include <new>
#include <memory>
#include <climits>
#include <cfloat>
#include <limits>
#include <exception>
#include <stdexcept>
#include <cassert>
#include <cerrno>
#include <cctype>
#include <cwctype>
#include <cstring>
#include <cwchar>
#include <string>
#include <vector>
#include <deque>
#include <list>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <algorithm>
#include <iterator>
#include <cmath>
#include <complex>
#include <valarray>
#include <numeric>
#include <iosfwd>
#include <ios>
#include <istream>
#include <ostream>
#include <iostream>
```

```
#include <fstream>
#include <sstream>
#include <strstream>
#include <iomanip>
#include <streambuf>
#include <cstdio>
#include <locale>
#include <clocale>
#include <ciso646>
```

4.31 ha_energy/pid.c File Reference

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

Include dependency graph for pid.c:



Functions

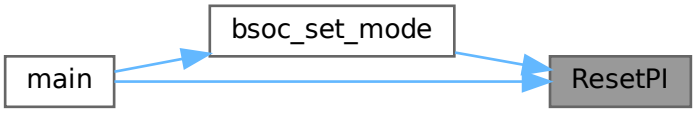
- double UpdatePI (volatile struct SPid *const pid, double const error)
- void ResetPI (volatile struct SPid *const pid)

4.31.1 Function Documentation

4.31.1.1 ResetPI()

```
void ResetPI (
            volatile struct SPid *const pid)
00030                                           {
00031     pid->dState = 0.0; // not used but cleared
00032     pid->iState = 0.0;
00033 }
```

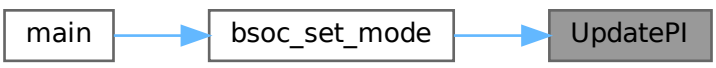
Here is the caller graph for this function:



4.31.1.2 UpdatePI()

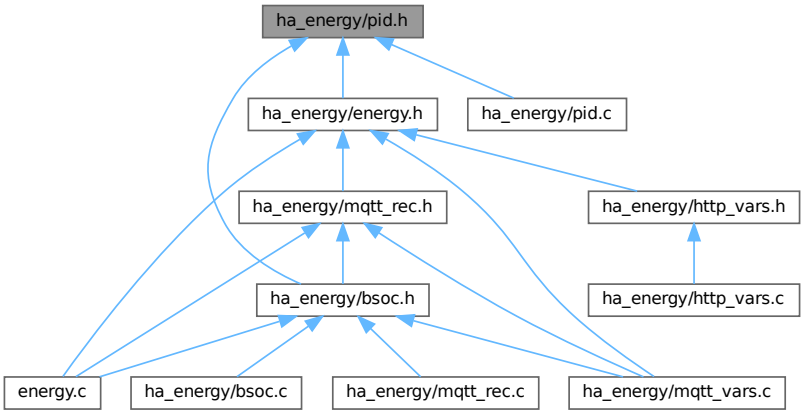
```
double UpdatePI (
            volatile struct SPid *const pid,
            double const error)
00006                                                                   {
00007     double pTerm, iTerm;
00008
00009     pTerm = pid->pGain * error; // calculate the proportional term
00010     // calculate the integral state with appropriate limiting
00011     pid->iState += error;
00012
00013     if (pid->iState > pid->iMax) {
00014         pid->iState = pid->iMax;
00015     } else if (pid->iState < pid->iMin) {
00016         pid->iState = pid->iMin;
00017     } else {
00018
00019     }
00020
00021     iTerm = (pid->iGain * pid->iState); // calculate the integral term
00022
00023     if ((pTerm + iTerm) > pid->iMax) {
00024         iTerm = 0.0f;
00025         pTerm = pid->iMax;
00026     }
00027     return pTerm + iTerm;
00028 }
```

Here is the caller graph for this function:



4.32 ha_energy/pid.h File Reference

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



Data Structures

- struct SPid

Functions

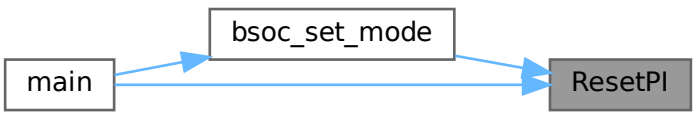
- double UpdatePI (volatile struct SPid *const, const double)
- void ResetPI (volatile struct SPid *const)

4.32.1 Function Documentation

4.32.1.1 ResetPI()

```
void ResetPI (
            volatile struct SPid * const pid)
00030                                                {
00031     pid->dState = 0.0; // not used but cleared
00032     pid->iState = 0.0;
00033 }
```

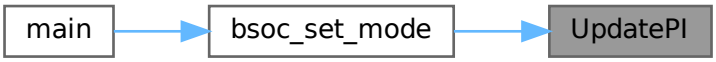
Here is the caller graph for this function:



4.32.1.2 UpdatePI()

```
double UpdatePI (
            volatile struct SPid * const pid,
            const double error)
00006                                                                   {
00007     double pTerm, iTerm;
00008
00009     pTerm = pid->pGain * error; // calculate the proportional term
00010     // calculate the integral state with appropriate limiting
00011     pid->iState += error;
00012
00013     if (pid->iState > pid->iMax) {
00014         pid->iState = pid->iMax;
00015     } else if (pid->iState < pid->iMin) {
00016         pid->iState = pid->iMin;
00017     } else {
00018
00019     }
00020
00021     iTerm = (pid->iGain * pid->iState); // calculate the integral term
00022
00023     if ((pTerm + iTerm) > pid->iMax) {
00024         iTerm = 0.0f;
00025         pTerm = pid->iMax;
00026     }
00027     return pTerm + iTerm;
00028 }
```

Here is the caller graph for this function:



4.33 pid.h

Go to the documentation of this file.

```
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