HA Energy

Generated by Doxygen 1.9.4

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:

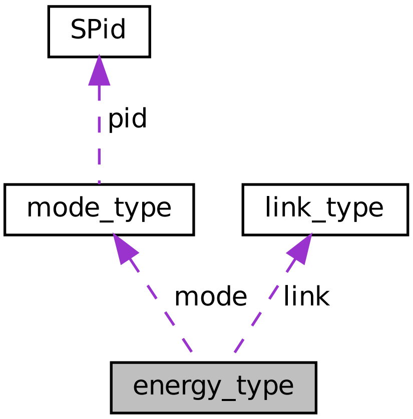
Chapter 3

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
```

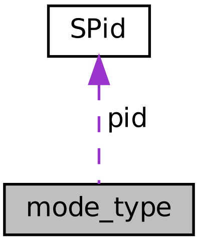
The documentation for this struct was generated from the following file:

- 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

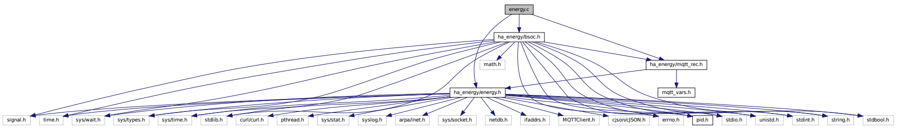
Chapter 4

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 )
{
    struct ha_flag_type *ha_flag = context;
    int32_t id_num;

    // bug-out if no context variables passed to callback
    if (context == NULL) {
        id_num = -1;
    } else {
        id_num = ha_flag->ha_id;
    }
    fprintf(fout, "\n%s Connection lost, exit ha_energy program\n", log_time(false));
    fprintf(fout, "%s     cause:  %s, %d\n", log_time(false), cause, id_num);
    fprintf(fout, "%sDAEMON failure  LOG Version %s :  MQTT Version %s\n", log_time(false), LOG_VERSION,
  MQTT_VERSION);
    fflush(fout);
    exit(EXIT_FAILURE);
}
```

4.1.2.2 ha_ac_off()

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

4.1.2.3 ha_ac_on()

```
void ha_ac_on (
            void )
```

```
{
    mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
    E.ac_sw_status = true;
}
```

4.1.2.4 ha_dc_off()

```
void ha_dc_off (
            void )
```

```
{
    mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
    E.gti_sw_status = false;
}
```

4.1.2.5 ha_dc_on()

```
void ha_dc_on (
            void )
```

```
{
    mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
    E.gti_sw_status = true;
}
```

4.1.2.6 log_time()

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

```
{
    static char time_log[RBUF_SIZ] = {0};
    static uint32_t len = 0, sync_time = TIME_SYNC_SEC - 1;
    time_t rawtime_log;

    tzset();
    timezone = 0;
    daylight = 0;
    time(&rawtime_log);
    if (sync_time++ > TIME_SYNC_SEC) {
        sync_time = 0;
        snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti time commands
        mqtt_gti_time(E.client_p, TOPIC_P, time_log);
    }

    sprintf(time_log, "%s", ctime(&rawtime_log));
    len = strlen(time_log);
    time_log[len - 1] = 0; // munge out the return character
    if (log) {
        fprintf(fout, "%s ", time_log);
        fflush(fout);
    }

    return time_log;
}
```

4.1.2.7 log_timer()

```
bool log_timer (
            void )
```

```
{
    bool itstime = false;

    if (E.log_spam < LOW_LOG_SPAM) {
        E.log_time_reset = 0;
        itstime = true;
    }
    if (E.log_time_reset > RESET_LOG_SPAM) {
        E.log_spam = 0;
        itstime = true;
    }
    return itstime;
}
```

4.1.2.8 main()

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

```
{
    struct itimerval new_timer = {
        .it_value.tv_sec = CMD_SEC,
        .it_value.tv_usec = 0,
        .it_interval.tv_sec = CMD_SEC,
        .it_interval.tv_usec = 0,
    };
    struct itimerval old_timer;
    time_t rawtime;
    MQTTClient_connectOptions conn_opts_p = MQTTClient_connectOptions_initializer,
        conn_opts_sd = MQTTClient_connectOptions_initializer,
        conn_opts_ha = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    char hname[256], *hname_ptr = hname;
    size_t hname_len = 12;

    gethostname(hname, hname_len);
    hname[12] = 0;
    printf("\r\n  LOG Version %s :  MQTT Version %s :  Host Name %s\r\n", LOG_VERSION, MQTT_VERSION, hname);
    showIP();
    skeleton_daemon();

    while (true) {
        switch (E.mode.E) {
        case E_INIT:

#ifdef LOG_TO_FILE
            fout = fopen(LOG_TO_FILE, "a");
            if (fout == NULL) {
                fout = fopen(LOG_TO_FILE_ALT, "a");
                if (fout == NULL) {
                    fout = stdout;
                    printf("\r\n%s Unable to open LOG file %s \r\n", log_time(false), LOG_TO_FILE_ALT);
                }
            }
#else
            fout = stdout;
#endif
            fprintf(fout, "\r\n%s LOG Version %s :  MQTT Version %s\r\n", log_time(false), LOG_VERSION, MQTT_VERSION);
            fflush(fout);

            if (!bsoc_init()) {
                fprintf(fout, "\r\n%s bsoc_init failure \r\n", log_time(false));
                fflush(fout);
                exit(EXIT_FAILURE);
            }
            /*
* set the timer for MQTT publishing sample speed
* CMD_SEC         10
*/
```

```
            setitimer(ITIMER_REAL, &new_timer, &old_timer);
            signal(SIGALRM, timer_callback);

            if (strncmp(hname, TNAME, 6) == 0) {
                MQTTClient_create(&E.client_p, LADDRESS, CLIENTID1,
                    MQTTCLIENT_PERSISTENCE_NONE, NULL);
                conn_opts_p.keepAliveInterval = 20;
                conn_opts_p.cleansession = 1;
                hname_ptr = LADDRESS;
            } else {
                MQTTClient_create(&E.client_p, ADDRESS, CLIENTID1,
                    MQTTCLIENT_PERSISTENCE_NONE, NULL);
                conn_opts_p.keepAliveInterval = 20;
                conn_opts_p.cleansession = 1;
                hname_ptr = ADDRESS;
            }

            fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID1);
            fflush(fout);
            MQTTClient_setCallbacks(E.client_p, &ha_flag_vars_ss, connlost, msgarrvd, delivered);
            if ((E.rc = MQTTClient_connect(E.client_p, &conn_opts_p)) != MQTTCLIENT_SUCCESS) {
                fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
  log_time(false), E.rc, hname_ptr, CLIENTID1);
                fflush(fout);
                pthread_mutex_destroy(&E.ha_lock);
                exit(EXIT_FAILURE);
            }

            if (strncmp(hname, TNAME, 6) == 0) {
                MQTTClient_create(&E.client_sd, LADDRESS, CLIENTID2,
                    MQTTCLIENT_PERSISTENCE_NONE, NULL);
                conn_opts_sd.keepAliveInterval = 20;
                conn_opts_sd.cleansession = 1;
                hname_ptr = LADDRESS;
            } else {
                MQTTClient_create(&E.client_sd, ADDRESS, CLIENTID2,
                    MQTTCLIENT_PERSISTENCE_NONE, NULL);
                conn_opts_sd.keepAliveInterval = 20;
                conn_opts_sd.cleansession = 1;
                hname_ptr = ADDRESS;
            }

            fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID2);
            fflush(fout);
            MQTTClient_setCallbacks(E.client_sd, &ha_flag_vars_sd, connlost, msgarrvd, delivered);
            if ((E.rc = MQTTClient_connect(E.client_sd, &conn_opts_sd)) != MQTTCLIENT_SUCCESS) {
                fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
  log_time(false), E.rc, hname_ptr, CLIENTID2);
                fflush(fout);
                pthread_mutex_destroy(&E.ha_lock);
                exit(EXIT_FAILURE);
            }

            /*
* Home Assistant MQTT receive messages
*/
            if (strncmp(hname, TNAME, 6) == 0) {
                MQTTClient_create(&E.client_ha, LADDRESS, CLIENTID3,
                    MQTTCLIENT_PERSISTENCE_NONE, NULL);
                conn_opts_ha.keepAliveInterval = 20;
                conn_opts_ha.cleansession = 1;
                hname_ptr = LADDRESS;
            } else {
                MQTTClient_create(&E.client_ha, ADDRESS, CLIENTID3,
                    MQTTCLIENT_PERSISTENCE_NONE, NULL);
                conn_opts_ha.keepAliveInterval = 20;
                conn_opts_ha.cleansession = 1;
                hname_ptr = ADDRESS;
            }

            fprintf(fout, "%s Connect MQTT server %s, %s\n", log_time(false), hname_ptr, CLIENTID3);
            fflush(fout);
            MQTTClient_setCallbacks(E.client_ha, &ha_flag_vars_ha, connlost, msgarrvd, delivered);
            if ((E.rc = MQTTClient_connect(E.client_ha, &conn_opts_ha)) != MQTTCLIENT_SUCCESS) {
                fprintf(fout, "%s Failed to connect MQTT server, return code %d %s, %s\n",
  log_time(false), E.rc, hname_ptr, CLIENTID3);
                fflush(fout);
                pthread_mutex_destroy(&E.ha_lock);
                exit(EXIT_FAILURE);
            }

            /*
* on topic received data will trigger the msgarrvd function
*/
            MQTTClient_subscribe(E.client_p, TOPIC_SS, QOS); // FM80 Q84
            MQTTClient_subscribe(E.client_sd, TOPIC_SD, QOS); // DUMPLOAD K42
            MQTTClient_subscribe(E.client_ha, TOPIC_HA, QOS); // Home Assistant Linux AMD64  and ARM64

            pubmsg.payload = "online";
            pubmsg.payloadlen = strlen("online");
            pubmsg.qos = QOS;
            pubmsg.retained = 0;
            ha_flag_vars_ss.deliveredtoken = 0;
            // notify HA we are running and controlling AC power plugs
            MQTTClient_publishMessage(E.client_p, TOPIC_PACA, &pubmsg, &token);
            MQTTClient_publishMessage(E.client_p, TOPIC_PDCA, &pubmsg, &token);

            // sync HA power switches
            mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
            mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
            mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
            mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
            mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
            mqtt_ha_switch(E.client_p, TOPIC_PACC, false);

            E.ac_sw_on = true; // can be switched on once
            E.gti_sw_on = true; // can be switched on once

            /*
* use libcurl to read AC power meter HTTP data
* iammeter connected for split single phase monitoring and one leg GTI power exporting
*/
            iammeter_read();

            /*
* start the main energy monitoring loop
*/
            fprintf(fout, "\r\n%s Solar Energy AC power controller\r\n", log_time(false));

#ifdef FAKE_VPV
            fprintf(fout, "\r\n Faking dumpload PV voltage\r\n");
#endif
            ha_flag_vars_ss.energy_mode = NORM_MODE;
            E.mode.E = E_WAIT;
            break;
        case E_WAIT:
            if (ha_flag_vars_ss.runner || E.speed_go++ > 1500000) {
                E.speed_go = 0;
                ha_flag_vars_ss.runner = false;
                E.mode.E = E_RUN;
            }

            usleep(100);
            /*
* main state-machine update sequence
*/
            bsoc_data_collect();
            if (!sanity_check()) {
                fprintf(fout, "\r\n%s Sanity Check error %d %s \r\n", log_time(false), E.sane, mqtt_name[E.sane]);
                fflush(fout);
            }

            /*
* stop and restart the energy control processing
* from inside the program or from a remote Home Assistant command
*/
            if (solar_shutdown()) {
                if (!E.startup) {
                    fprintf(fout, "%s SHUTDOWN Solar Energy Control ---> \r\n", log_time(false));
                }
                fflush(fout);
                ramp_down_gti(E.client_p, true);
                usleep(100000); // wait
                ramp_down_ac(E.client_p, true);
                usleep(100000); // wait
                ramp_down_gti(E.client_p, true);
                usleep(100000); // wait
                ramp_down_ac(E.client_p, true);
                usleep(100000); // wait
                if (!E.startup) {
                    fprintf(fout, "%s Completed SHUTDOWN, Press again to RESTART.\r\n", log_time(false));
                    fflush(fout);
                }
                fflush(fout);

                uint8_t iam_delay = 0;
                while (solar_shutdown()) {
                    mqtt_ha_shutdown(E.client_p, TOPIC_SHUTDOWN);
                    usleep(USEC_SEC); // wait
                    if ((int32_t) E.mvar[V_HACSW]) {
                        ha_ac_off();
                    }
                    if ((int32_t) E.mvar[V_HDCSW]) {
                        ha_dc_off();
                    }
                    if ((iam_delay++ > IAM_DELAY) && E.link.shutdown) {
                        E.fm80 = true;
                        E.dumpload = true;
                        E.iammeter = true;
                        E.homeassistant = true;
                    }
                }
                E.link.shutdown = 0;
                fprintf(fout, "%s RESTART Solar Energy Control\r\n", log_time(false));
                fflush(fout);
                bsoc_set_mode(E.mode.pv_bias, true, true);
                E.dl_excess = true;
                mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 1); // zero power at startup
                E.dl_excess = false;
#ifdef AUTO_CHARGE
                mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
#endif
                usleep(100000); // wait
                E.gti_sw_status = true;
                ResetPI(&E.mode.pid);
                ha_flag_vars_ss.runner = true;
                E.fm80 = true;
                E.dumpload = true;
                E.iammeter = true;
                E.homeassistant = true;
                E.mode.in_pid_control = false; // shutdown auto energy control
                E.mode.R = R_INIT;
            }
            if (ha_flag_vars_ss.receivedtoken) {
                ha_flag_vars_ss.receivedtoken = false;
            }
            if (ha_flag_vars_sd.receivedtoken) {
                ha_flag_vars_sd.receivedtoken = false;
            }
            break;
        case E_RUN:
            usleep(100);
            switch (E.mode.R) {
            case R_INIT:
                E.once_ac = true;
                E.once_gti = true;
                E.ac_sw_on = true;
                E.gti_sw_on = true;
                E.mode.R = R_RUN;
                E.mode.no_float = true;
                break;
            case R_FLOAT:
                if (E.mode.no_float) {
                    E.once_ac = true;
                    E.once_gti = true;
                    E.ac_sw_on = true;
                    E.gti_sw_on = true;
                    E.gti_sw_status = false;
                    E.ac_sw_status = false;
                    E.mode.no_float = false;
                }
                if (!E.gti_sw_status) {
                    if (gti_test() > MIN_BAT_KW_GTI_HI) {
                        mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
                        E.gti_sw_status = true;
                        fprintf(fout, "%s R_FLOAT DC switch true \r\n", log_time(false));
                    }
                }
                usleep(100000); // wait
                if (!E.ac_sw_status) {
                    if (ac_test() > MIN_BAT_KW_AC_HI) {
                        mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
                        E.ac_sw_status = true;
                        fprintf(fout, "%s R_FLOAT AC switch true \r\n", log_time(false));
                    }
                }
                E.mode.pv_bias = PV_BIAS;
                fm80_float(true);
                break;
            case R_RUN:
            default:
                E.mode.R = R_RUN;
                E.mode.no_float = true;
                break;
            }
            /*
* main state-machine update sequence and control logic
*/
            /*
```

```
* check for idle/data errors flags from sensors and HA
*/
            if (!E.mode.data_error) {
                bsoc_set_mode(E.mode.pv_bias, true, false);
                if (E.gti_delay++ >= GTI_DELAY) {
                    char gti_str[SBUF_SIZ];
                    int32_t error_drive;

                    /*
* reset the control mode from simple switched power to PID control
*/
                    if (!E.mode.in_pid_control) {
                        mqtt_ha_switch(E.client_p, TOPIC_PDCC, true);
                        E.gti_sw_status = true;
                        usleep(100000); // wait
                        mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
                        E.ac_sw_status = true;
                        E.mode.pv_bias = PV_BIAS;
                        fprintf(fout, "%s in_pid_mode AC/DC switch true \r\n", log_time(false));
                        fm80_float(true);
                    } else {
                        if (!fm80_float(true)) {
                            E.mode.pv_bias = (int32_t) E.mode.error - PV_BIAS;
                        }
                    }
                    /*
* use PID style set-point error correction
*/
                    E.mode.in_pid_control = true;
                    E.gti_delay = 0;
                    /*
* adjust power balance if battery charging energy is low
*/
                    if (E.mvar[V_DPBAT] > PV_DL_BIAS_RATE) {
                        error_drive = (int32_t) E.mode.error - E.mode.pv_bias; // PI feedback control signal
                    } else {
                        error_drive = (int32_t) E.mode.error - PV_BIAS_RATE;
                    }
                    /*
* when main battery is in float, crank-up the power draw from the solar panels
*/
                    if (fm80_float(true)) {
                        error_drive = (int32_t) (E.mode.error + PV_BIAS);
                    }
                    /*
* don't drive to zero power
*/
                    if (error_drive < 0) {
                        error_drive = PV_BIAS_LOW; // control wide power swings
                        if (!fm80_sleep()) { // check for using sleep bias
                            if ((E.mvar[V_FBEKW] > MIN_BAT_KW_BSOC_SLP) && (E.mvar[V_PWA] > PWA_SLEEP)) {
                                error_drive = PV_BIAS_SLEEP; // use higher power when we still have sun for better inverter efficiency
                            }
                        }
                    }

                    /*
* reduce charging/diversion power to safe PS limits
*/
                    if (E.mode.dl_mqtt_max > PV_DL_MPTT_MAX) {
                        if (!E.dl_excess) {
                            error_drive = PV_DL_MPTT_IDLE;
                        } else {
                            if (E.mode.dl_mqtt_max > PV_DL_MPTT_EXCESS) {
                                error_drive = PV_DL_MPTT_IDLE;
                            }
                        }
                    } else {
                        if (E.dl_excess) {
                            error_drive = PV_DL_EXCESS + E.dl_excess_adj;
                        }
                    }

                    /*
* shutdown GTI power at low DL battery Ah or Voltage
*/
                    if ((E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) || (E.mvar[V_DVBAT] < PV_DL_B_V_LOW)) {
                        error_drive = PV_BIAS_ZERO;
                    }

                    snprintf(gti_str, SBUF_SIZ - 1, "V%04dX", error_drive); // format for dumpload controller gti power commands
                    mqtt_gti_power(E.client_p, TOPIC_P, gti_str, 2);
                }
```

```

#ifndef  FAKE_VPV
                if (fm80_float(true) || ((ac1_filter(E.mvar[V_BEN]) > BAL_MAX_ENERGY_AC) && (ac_test() > MIN_BAT_KW_AC_HI))) {
                    ramp_up_ac(E.client_p, E.ac_sw_on); // use once control
#ifdef PSW_DEBUG
                    fprintf(fout, "%s MIN_BAT_KW_AC_HI AC switch %d \r\n", log_time(false), E.ac_sw_on);
#endif
                    E.ac_sw_on = false; // once flag
                }
#endif
                if (((ac2_filter(E.mvar[V_BEN]) < BAL_MIN_ENERGY_AC) || ((ac_test() < (MIN_BAT_KW_AC_LO + E.ac_low_adj))))) {
                    if (!fm80_float(true)) {
                        ramp_down_ac(E.client_p, E.ac_sw_on);
                        if (log_timer()) {
                            fprintf(fout, "%s RAMP DOWN AC, MIN_BAT_KW_AC_LO AC switch %d \r\n", log_time(false), E.ac_sw_on);
                        }
                    }
                    E.ac_sw_on = true;
                }


                /*
* Dump Load Excess testing
* send excess power into the home power grid taking care not to export energy to the utility grid
*/
                if (((dc1_filter(E.mvar[V_BEN]) > BAL_MAX_ENERGY_GTI) && (gti_test() > MIN_BAT_KW_GTI_HI)) || E.dl_excess) {
#ifndef  FAKE_VPV
#ifdef B_DLE_DEBUG
                    if (E.dl_excess) {
                        fprintf(fout, "%s DL excess ramp_up_gti, DC switch %d\r\n", log_time(false), E.gti_sw_on);
                    }
#endif
                    ramp_up_gti(E.client_p, E.gti_sw_on, E.dl_excess);
                    if (log_timer()) {
                        fprintf(fout, "%s RAMP DOWN DC, MIN_BAT_KW_GTI_HI DC switch %d \r\n", log_time(false), E.gti_sw_on);
                    }
                    E.gti_sw_on = false; // once flag
#endif
                } else {
                    if ((dc2_filter(E.mvar[V_BEN]) < BAL_MIN_ENERGY_GTI) || (gti_test() < (MIN_BAT_KW_GTI_LO + E.gti_low_adj))) {
                        if (!E.dl_excess) {
                            if (log_timer()) {
                                ramp_down_gti(E.client_p, true);
#ifdef PSW_DEBUG
                                fprintf(fout, "%s MIN_BAT_KW_GTI_LO DC switch %d \r\n", log_time(false), E.gti_sw_on);
#endif
                            }
                            E.gti_sw_on = true;
                        }
                    }
                }
            };

#ifdef B_ADJ_DEBUG
            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);
#endif
#ifdef B_DLE_DEBUG
            if (E.dl_excess) {
                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);
            }
#endif

            time(&rawtime);

            if (E.im_delay++ >= IM_DELAY) {
                E.im_delay = 0;
                iammeter_read();
            }
            if (E.im_display++ >= IM_DISPLAY) {
                char buffer[SYSLOG_SIZ];
                uint32_t len;

                E.im_display = 0;
                mqtt_ha_pid(E.client_p, TOPIC_PPID);
                if (!(E.fm80 && E.dumpload && E.iammeter)) {
                    if (!E.iammeter) {
                        E.link.iammeter_error++;
                    } else {
                        E.link.mqtt_error++;
                    }
                    E.link.shutdown++;
                    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,
                        E.link.mqtt_count, E.link.mqtt_error, E.link.iammeter_count, E.link.iammeter_error);
                    fflush(fout);
                    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,
                        E.link.mqtt_count, E.link.mqtt_error, E.link.iammeter_count, E.link.iammeter_error);
                    syslog(LOG_NOTICE, buffer);
                    mqtt_ha_shutdown(E.client_p, TOPIC_SHUTDOWN);
                    E.mode.data_error = true;
                } else {
                    E.mode.data_error = false;
                    E.link.shutdown = 0;
                }
                snprintf(buffer, RBUF_SIZ - 1, "%s", ctime(&rawtime));
                len = strlen(buffer);
                buffer[len - 1] = 0; // munge out the return character
                fprintf(fout, "%s ", buffer);
                fflush(fout);
                E.fm80 = false;
                E.dumpload = false;
                E.homeassistant = false;
                E.iammeter = false;
                sync_ha();
                print_im_vars();
                print_mvar_vars();
                fprintf(fout, "%s\r", ctime(&rawtime));
            }
            E.mode.E = E_WAIT;
            fflush(fout);
            if (E.mode.con6) {
                E.mode.R = R_IDLE;
            }
            if (E.mode.con7) {
                E.mode.E = E_STOP;
            }
            break;
        case E_STOP:
        default:
            fflush(fout);
            fprintf(fout, "\r\n%s HA Energy stopped and exited.\r\n", log_time(false));
            fflush(fout);
            return 0;
            break;
        }
    }
}
```

4.1.2.9 ramp_down_ac()

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

```
{
    if (sw_off) {
        mqtt_ha_switch(client_p, TOPIC_PACC, false);
        E.ac_sw_status = false;
        usleep(500000);
    }
    E.once_ac = true;
}
```

4.1.2.10 ramp_down_gti()

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

```
{
    if (sw_off) {
        mqtt_ha_switch(client_p, TOPIC_PDCC, false);
        E.once_gti_zero = true;
        E.gti_sw_status = false;
    }
    E.once_gti = true;

    if (E.once_gti_zero) {
        mqtt_gti_power(client_p, TOPIC_P, "Z#", 7); // zero power
        E.once_gti_zero = false;
    }
}
```

4.1.2.11 ramp_up_ac()

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

```
{

    if (start) {
        E.once_ac = true;
    }

    if (E.once_ac) {
        E.once_ac = false;
        mqtt_ha_switch(client_p, TOPIC_PACC, true);
        E.ac_sw_status = true;
        usleep(500000); // wait for voltage to ramp
    }
}
```

4.1.2.12 ramp_up_gti()

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

```
{
    static uint32_t sequence = 0;

    if (start) {
        E.once_gti = true;
    }

    if (E.once_gti) {
        E.once_gti = false;
        sequence = 0;
        if (!excess) {
            mqtt_ha_switch(client_p, TOPIC_PDCC, true);
            E.gti_sw_status = true;
            usleep(500000); // wait for voltage to ramp
        } else {
            sequence = 1;
        }
    }

    switch (sequence) {
    case 4:
        E.once_gti_zero = true;
        break;
    case 3:
    case 2:
    case 1:
        E.once_gti_zero = true;
        if (bat_current_stable() || E.dl_excess) { // check battery current std dev, stop 'motorboating'
            sequence++;
            if (!mqtt_gti_power(client_p, TOPIC_P, "+#", 3)) {
                sequence = 0;
            }; // +100W power
        } else {
            usleep(500000); // wait a bit more for power to be stable
            sequence = 1; // do power ramps when ready
            if (!mqtt_gti_power(client_p, TOPIC_P, "-#", 4)) {
                sequence = 0;
            }; // - 100W power
        }
        break;
    case 0:
        sequence++;
        if (E.once_gti_zero) {
            mqtt_gti_power(client_p, TOPIC_P, "Z#", 5); // zero power
            E.once_gti_zero = false;
        }
        break;
    default:
        if (E.once_gti_zero) {
            mqtt_gti_power(client_p, TOPIC_P, "Z#", 6); // zero power
            E.once_gti_zero = false;
        }
        sequence = 0;
        break;
    }
}
```

4.1.2.13 sanity_check()

```
bool sanity_check (
               void   )
```

```
{
    if (E.mvar[V_PWA] > PWA_SANE) {
        E.sane = S_PWA;
        return false;
    }
    if (E.mvar[V_PAMPS] > PAMPS_SANE) {
        E.sane = S_PAMPS;
        return false;
    }
    if (E.mvar[V_PVOLTS] > PVOLTS_SANE) {
        E.sane = S_PVOLTS;
        return false;
    }
    if (E.mvar[V_FBAMPS] > BAMPS_SANE) {
        E.sane = S_FBAMPS;
        return false;
    }
    return true;
}
```

4.1.2.14 showIP()

```
void showIP (
               void   )
```

```
{
    struct ifaddrs *ifaddr, *ifa;
    int s;
    char host[NI_MAXHOST];

    if (getifaddrs(&ifaddr) == -1) {
        perror("getifaddrs");
        exit(EXIT_FAILURE);
    }

    for (ifa = ifaddr; ifa != NULL; ifa = ifa->ifa_next) {
        if (ifa->ifa_addr == NULL)
            continue;

        s = getnameinfo(ifa->ifa_addr, sizeof(struct sockaddr_in), host, NI_MAXHOST, NULL, 0,
  NI_NUMERICHOST);

        if (ifa->ifa_addr->sa_family == AF_INET) {
            if (s != 0) {
                exit(EXIT_FAILURE);
            }
            printf("\tInterface :  <%s>\n", ifa->ifa_name);
            printf("\t  Address :  <%s>\n", host);
        }
    }

    freeifaddrs(ifaddr);
}
```

4.1.2.15 skeleton_daemon()

```
static void skeleton_daemon ( )  [static]
{
    pid_t pid;

    /* Fork off the parent process */
    pid = fork();

    /* An error occurred */
    if (pid < 0) {
        printf("\r\n%sDAEMON failure  LOG Version %s :  MQTT Version %s\r\n", log_time(false),
  LOG_VERSION, MQTT_VERSION);
        exit(EXIT_FAILURE);
    }

    /* Success:  Let the parent terminate */
    if (pid > 0) {
        exit(EXIT_SUCCESS);
    }

    /* On success:  The child process becomes session leader */
    if (setsid() < 0) {
        exit(EXIT_FAILURE);
    }

    /* Catch, ignore and handle signals */
    /*TODO: Implement a working signal handler */
    //    signal(SIGCHLD, SIG_IGN);
    //    signal(SIGHUP, SIG_IGN);

    /* Fork off for the second time*/
    pid = fork();

    /* An error occurred */
    if (pid < 0) {
        exit(EXIT_FAILURE);
    }

    /* Success:  Let the parent terminate */
    if (pid > 0) {
        exit(EXIT_SUCCESS);
    }

    /* Set new file permissions */
    umask(0);

    /* Change the working directory to the root directory */
    /* or another appropriated directory */
    chdir("/");

    /* Close all open file descriptors */
    int x;
    for (x = sysconf(_SC_OPEN_MAX); x >= 0; x--) {
        close(x);
    }

}
```

4.1.2.16 solar_shutdown()

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

```
{
    static bool ret = false;

    if (E.startup) {
        ret = true;
        E.startup = false;
        return ret;
    } else {
        ret = false;

        /*
* FIXME
*
*/
    }

    if (E.solar_shutdown) {
        ret = true;
    } else {
        ret = false;
    }

    if ((E.mvar[V_FBEKW] < BAT_CRITICAL) && !E.startup) { // special case for low battery
        if (!E.mode.bat_crit) {
            ret = true;
#ifdef CRITIAL_SHUTDOWN_LOG
            fprintf(fout, "%s Solar BATTERY CRITICAL shutdown comms check ret = %d \r\n", log_time(false), ret);
            fflush(fout);
#endif
            E.mode.bat_crit = true;
            return ret;
        }
    } else {
        E.mode.bat_crit = false;
    }

    if (E.link.shutdown >= MAX_ERROR) {
        ret = true;
        if (E.fm80 && E.dumpload && E.iammeter) {
            ret = false;
            E.link.shutdown = 0;
        }

#ifdef DEBUG_SHUTDOWN
        fprintf(fout, "%s Solar shutdown comms check ret = %d \r\n", log_time(false), ret);
        fflush(fout);
#endif
    }
    return ret;
}
```

4.1.2.17 sync_ha()

```
bool sync_ha (
            void )
```

```
{
    bool sync = false;
    if (E.gti_sw_status != (bool) ((int32_t) E.mvar[V_HDCSW])) {
        fprintf(fout, "DC_MM %d %d ", (bool) E.gti_sw_status, (bool) ((int32_t) E.mvar[V_HDCSW]));
        mqtt_ha_switch(E.client_p, TOPIC_PDCC, !E.gti_sw_status);
        E.dc_mismatch = true;
        fflush(fout);
        sync = true;
    } else {
        E.dc_mismatch = false;
    }

    E.ac_sw_status = (bool) ((int32_t) E.mvar[V_HACSW]); // TEMP FIX for MISmatch errors
    if (E.ac_sw_status != (bool) ((int32_t) E.mvar[V_HACSW])) {
        fprintf(fout, "AC_MM %d %d ", (bool) E.ac_sw_status, (bool) ((int32_t) E.mvar[V_HACSW]));
        mqtt_ha_switch(E.client_p, TOPIC_PACC, !E.ac_sw_status);
        E.ac_mismatch = true;
        fflush(fout);
        sync = true;
    } else {
        E.ac_mismatch = false;
    }
    return sync;
}
```

4.1.2.18 timer_callback()

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

```
{
    signal(signum, timer_callback);
    ha_flag_vars_ss.runner = true;
    E.ten_sec_clock++;
    E.log_spam++;
    E.log_time_reset++;
    if (E.log_spam > MAX_LOG_SPAM) {
        E.log_spam = 0;
    }
}
```

4.1.3 Variable Documentation

4.1.3.1 board_name

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

4.1.3.2 driver_name

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

4.1.3.3 E

```
struct energy_type E
```

4.1.3.4 fout

```
FILE* fout
```

4.1.3.5 ha_flag_vars_ha

```
struct ha_flag_type ha_flag_vars_ha
```

Initial value:

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

4.1.3.6 ha_flag_vars_pc

```
struct ha_flag_type ha_flag_vars_pc
```

Initial value:

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

4.1.3.7 ha_flag_vars_sd

```
struct ha_flag_type ha_flag_vars_sd
```

Initial value:

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

4.1.3.8 ha_flag_vars_ss

```
struct ha_flag_type ha_flag_vars_ss
```

Initial value:

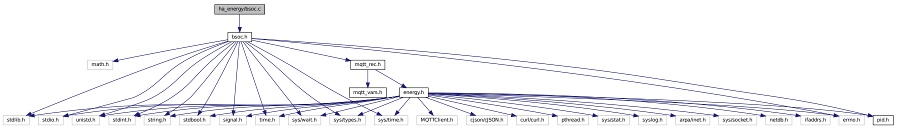
```
= {
    .runner = false,
    .receivedtoken = false,
    .deliveredtoken = false,
    .rec_ok = false,
    .ha_id = FM80_ID,
    .var_update = 0,
    .energy_mode = NORM_MODE,
}
```

4.2 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 )
```

```
{
    static double accum = 0.0f;
    static double coef = COEFF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.3.1.2 ac1_filter()

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

```
{
    static double accum = 0.0f;
    static double coef = COEF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.3.1.3 ac2_filter()

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

```
{
    static double accum = 0.0f;
    static double coef = COEF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.3.1.4 ac_test()

```
double ac_test (
                void  )
```

```
{
    return ac0_filter(L.ac_weight);
}
```

4.3.1.5 bat_current_stable()

```
bool bat_current_stable (
                void  )
```

```
{
    static double gap = 0.0f;

    if (L.batc_std_dev <= (MAX_BATC_DEV + gap)) {
        gap = MAX_BATC_DEV;
        if (L.bat_c_std_dev[0] < BAT_C_DRAW) {
            return true;
        } else {
            gap = 0.0f;
            return false;
        }
    } else {
        gap = 0.0f;
        return false;
    }
}
```

4.3.1.6 bsoc_ac()

```
double bsoc_ac (
               void )
```

```
{

    return ac0_filter(L.ac_weight);
};
```

4.3.1.7 bsoc_data_collect()

```
bool bsoc_data_collect (
               void )
```

```
{
    bool ret = false;
    static uint32_t i = 0;
    // lockout threaded updates
    pthread_mutex_lock(&E.ha_lock); // lockout MQTT var updates

    L.ac_weight = E.mvar[V_FBEKW];
    L.gti_weight = E.mvar[V_FBEKW];
#ifdef FAKE_VPV // no DUMPLOAD AC charger
    if (E.gti_sw_on) {
        pv_voltage = PV_V_NOM;
    } else {
        pv_voltage = PV_V_FAKE;
    }
    E.mvar[V_DVPV] = pv_voltage;
#else
    L.pv_voltage = E.mvar[V_DVPV];
#endif
    L.bat_voltage = E.mvar[V_DVBAT];
    L.bat_current = E.mvar[V_DCMPPT];
    E.ac_low_adj = E.mvar[V_FSO]* -0.5f;
    E.gti_low_adj = E.mvar[V_FACE] * -0.5f;
    E.mode.dl_mqtt_max = E.mvar[V_DPMPPT];

    pthread_mutex_unlock(&E.ha_lock); // resume remote MQTT var updates

    if (E.ac_low_adj < -2000.0f) {
        E.ac_low_adj = -2000.0f;
    }
    if (E.gti_low_adj < -2000.0f) {
        E.gti_low_adj = -2000.0f;
    }

    L.bat_c_std_dev[i++] = L.bat_current;
    if (i >= DEV_SIZE) {
        i = 0;
    }

    calculateStandardDeviation(DEV_SIZE, L.bat_c_std_dev);

#ifdef BSOC_DEBUG
    fprintf(fout, "\r\nmqtt var bsoc update\r\n");
#endif
    return ret;
}
```

4.3.1.8 bsoc_gti()

```
double bsoc_gti (
               void )
```

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

```
    // check for 48VDC AC charger powered from the Solar battery bank AC inverter unless E.dl_excess is TRUE
    if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
        L.gti_weight = 0.0f; // reduce power to zero
    } else {
        if (E.dl_excess) {
            if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
                L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
            } else {
                L.gti_weight = 0.0f; // reduce power to zero
            }
        }
    }


    return dc0_filter(L.gti_weight);
};
```

4.3.1.9 bsoc_init()

```
bool bsoc_init (
               void  )
```

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

4.3.1.10 bsoc_set_mode()

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

```
{
    static bool bsoc_mode = false;
    static bool bsoc_high = false, ha_ac_mode = true;
    static double accum = 0.0f, vpwa = 0.0f;

    if (init) {
        bsoc_mode = false;
        bsoc_high = false;
        ha_ac_mode = true;
        accum = 0.0f;
        vpwa = 0.0f;
        return true;
    }
    /*
* running avg filter
*/
    accum = accum - accum / COEFN + E.mvar[V_PWA];
    vpwa = accum / COEFN;

    if ((vpwa >= PV_FULL_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
        if (!bsoc_mode) {
            ResetPI(&E.mode.pid);
        }
        bsoc_mode = true;
        bsoc_high = true;
        if (!ha_ac_mode) {
            ha_ac_on();
            ha_ac_mode = true;
        }

    } else {
        if (bsoc_high) { // turn off at min limit power
            if ((vpwa >= PV_MIN_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
                bsoc_mode = true;
                if (ha_ac_mode) {
                    ha_ac_off();
                    ha_ac_mode = false;
                }
            } else {
                bsoc_high = false;
                ha_ac_mode = false;
            }
        }
    }

    E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) + E.mvar[V_DPPV]; // use as a temp variable
    E.mode.total_system = (E.mvar[V_FLO] - E.mode.gti_dumpload) + E.mvar[V_DPPV] +(E.print_vars[L3_P]*
  -1.0f);
    E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) - E.mvar[V_DPPV]; // use this value

    /*
* look at system energy balance for power control drive
*/
    if (mode) { // add GTI power from dumpload
        E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + E.mode.gti_dumpload +
  PBAL_OFFSET);
    } else {
        E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + PBAL_OFFSET);
    }

    if (E.mode.error > 0.0f) {
        L.coef = COEF;
    } else {
        L.coef = COEFN;
    }
    E.mode.target = target;
    E.mode.error = round(error_filter(E.mode.error));
    /*
* check for idle flag from HA
*/
    if (E.mode.con6) {
        ha_ac_mode = true;
        bsoc_mode = false;
    }

    /*
* HA start excess button pressed
*/
    if (E.mode.con4) {
        E.dl_excess = true;
        E.mode.con4 = false;
    }

    /*
* HA stop excess button pressed
*/
    if (E.mode.con5) {
        mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 9); // zero power at excess shutdown
        E.dl_excess = false;
        E.mode.con5 = false;
    }

    /*
* DL buffer battery low set-point excess load shutdown
*/
    if (E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) {
        mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 10); // zero power at excess shutdown
        E.dl_excess = false;
        E.mode.con4 = false;
        E.mode.con5 = false;
    }

    return bsoc_mode;
}
```

4.3.1.11 bsoc_set_std_dev()

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

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

4.3.1.12 calculateStandardDeviation()

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

```
{
    // variable to store sum of the given data
    double sum = 0;

    for (int i = 0; i < N; i++) {
        sum += data[i];
    }

    // calculating mean
    double mean = sum / N;

    // temporary variable to store the summation of square
    // of difference between individual data items and mean
    double values = 0;

    for (int i = 0; i < N; i++) {
        values += pow(data[i] - mean, 2);
    }

    // variance is the square of standard deviation
    double variance = values / N;

    // calculating standard deviation by finding square root
    // of variance
    double standardDeviation = sqrt(variance);
    L.batc_std_dev = standardDeviation;

#ifdef BSOC_DEBUG
    // printing standard deviation
    fprintf(fout, "STD DEV of Current %.2f\r\n", standardDeviation);
#endif
    return standardDeviation;
}
```

4.3.1.13 dc0_filter()

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

```
{
    static double accum = 0.0f;
    static double coef = COEFF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.3.1.14 dc1_filter()

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

```
{
    static double accum = 0.0f;
    static double coef = COEF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.3.1.15 dc2_filter()

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

```
{
    static double accum = 0.0f;
    static double coef = COEF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.3.1.16 drive0_filter()

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

```
{
    static double accum = 0.0f;
    static double coef = COEF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.3.1.17 drive1_filter()

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

```
{
    static double accum = 0.0f;
    static double coef = COEFF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.3.1.18 error_filter()

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

```
{
    static double accum = 0.0f;
    accum = accum - accum / L.coef + raw;
    return accum / L.coef;
}
```

4.3.1.19 get_batc_dev()

```
double get_batc_dev (
            void  )
```

```
{
    return L.batc_std_dev;
}
```

4.3.1.20 gti_test()

```
double gti_test (
              void )
```

```
{
    // check for 48VDC AC charger powered from the Solar battery bank AC inverter
    if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
        L.gti_weight = 0.0f; // reduce power to zero
#ifdef BSOC_DEBUG
        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);
#endif
    } else {
        if (E.dl_excess) {
            if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
                L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
            } else {
                L.gti_weight = 0.0f; // reduce power to zero
            }
        }
    }
    return dc0_filter(L.gti_weight);
}
```

4.3.2 Variable Documentation

4.3.2.1 L

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

Initial value:

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

4.3.2.2 mqtt_name

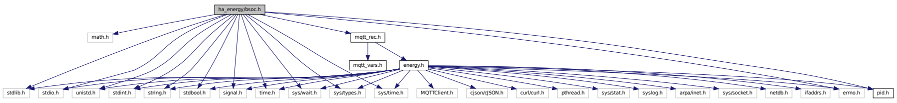
```
const char* mqtt_name[V_DLAST]
```

4.4 ha_energy/bsoc.h File Reference

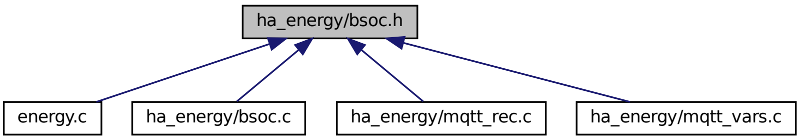
```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#include <signal.h>
```

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

Include dependency graph for bsoc.h:



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



Macros

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

Functions

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

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

4.4.1 Macro Definition Documentation

4.4.1.1 BAT_C_DRAW

```
#define BAT_C_DRAW 3.0f
```

4.4.1.2 COEF

```
#define COEF 8.0f
```

4.4.1.3 COEFF

```
#define COEFF 2.0f
```

4.4.1.4 COEFN

```
#define COEFN 4.0f
```

4.4.1.5 DEV_SIZE

```
#define DEV_SIZE 10
```

4.4.1.6 MAX_BATC_DEV

```
#define MAX_BATC_DEV 1.5f
```

4.4.1.7 MIN_BAT_KW

```
#define MIN_BAT_KW 4100.0f
```

4.4.1.8 MIN_BAT_VOLTS

```
#define MIN_BAT_VOLTS 23.0f
```

4.4.1.9 MIN_PV_VOLTS

```
#define MIN_PV_VOLTS 5.0f
```

4.4.1.10 PBAL_OFFSET

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

4.4.1.11 PV_FULL_PWR

```
#define PV_FULL_PWR 300.0f
```

4.4.1.12 PV_MIN_PWR

```
#define PV_MIN_PWR 160.0f
```

4.4.1.13 PV_V_FAKE

```
#define PV_V_FAKE 0.336699f
```

4.4.1.14 PV_V_NOM

```
#define PV_V_NOM 60.0f
```

4.4.2 Function Documentation

4.4.2.1 ac0_filter()

```
double ac0_filter (
              const double raw )
{
    static double accum = 0.0f;
    static double coef = COEFF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.4.2.2 ac1_filter()

```
double ac1_filter (
              const double raw )
{
    static double accum = 0.0f;
    static double coef = COEF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.4.2.3 ac2_filter()

```
double ac2_filter (
              const double raw )
{
    static double accum = 0.0f;
    static double coef = COEF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.4.2.4 ac_test()

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

4.4.2.5 bat_current_stable()

```
bool bat_current_stable (
               void  )
```

```
{
    static double gap = 0.0f;

    if (L.batc_std_dev <= (MAX_BATC_DEV + gap)) {
        gap = MAX_BATC_DEV;
        if (L.bat_c_std_dev[0] < BAT_C_DRAW) {
            return true;
        } else {
            gap = 0.0f;
            return false;
        }
    } else {
        gap = 0.0f;
        return false;
    }
}
```

4.4.2.6 bsoc_ac()

```
double bsoc_ac (
               void  )
```

```
{

    return ac0_filter(L.ac_weight);
};
```

4.4.2.7 bsoc_data_collect()

```
bool bsoc_data_collect (
               void  )
```

```
{
    bool ret = false;
    static uint32_t i = 0;
    // lockout threaded updates
    pthread_mutex_lock(&E.ha_lock); // lockout MQTT var updates

    L.ac_weight = E.mvar[V_FBEKW];
    L.gti_weight = E.mvar[V_FBEKW];
#ifdef FAKE_VPV // no DUMPLOAD AC charger
    if (E.gti_sw_on) {
        pv_voltage = PV_V_NOM;
    } else {
        pv_voltage = PV_V_FAKE;
    }
    E.mvar[V_DVPV] = pv_voltage;
#else
    L.pv_voltage = E.mvar[V_DVPV];
#endif
    L.bat_voltage = E.mvar[V_DVBAT];
    L.bat_current = E.mvar[V_DCMPPT];
    E.ac_low_adj = E.mvar[V_FSO]* -0.5f;
    E.gti_low_adj = E.mvar[V_FACE] * -0.5f;
    E.mode.dl_mqtt_max = E.mvar[V_DPMPPT];

    pthread_mutex_unlock(&E.ha_lock); // resume remote MQTT var updates

    if (E.ac_low_adj < -2000.0f) {
        E.ac_low_adj = -2000.0f;
    }
    if (E.gti_low_adj < -2000.0f) {
        E.gti_low_adj = -2000.0f;
    }

    L.bat_c_std_dev[i++] = L.bat_current;
    if (i >= DEV_SIZE) {
        i = 0;
    }

    calculateStandardDeviation(DEV_SIZE, L.bat_c_std_dev);

#ifdef BSOC_DEBUG
    fprintf(fout, "\r\nmqtt var bsoc update\r\n");
#endif
    return ret;
}
```

4.4.2.8 bsoc_gti()

```
double bsoc_gti (
              void  )
```

```
{
#ifdef BSOC_DEBUG
    fprintf(fout, "pvp %f, gweight %f, aweight %f, batv %f, batc %f\r\n", pv_voltage, gti_weight, ac_weight, bat_voltage, bat_current);
#endif
    // check for 48VDC AC charger powered from the Solar battery bank AC inverter unless E.dl_excess is TRUE
    if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
        L.gti_weight = 0.0f; // reduce power to zero
    } else {
        if (E.dl_excess) {
            if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
                L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
            } else {
                L.gti_weight = 0.0f; // reduce power to zero
            }
        }
    }


    return dc0_filter(L.gti_weight);
};
```

4.4.2.9 bsoc_init()

```
bool bsoc_init (
              void  )
```

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

4.4.2.10 bsoc_set_mode()

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

```
{
    static bool bsoc_mode = false;
    static bool bsoc_high = false, ha_ac_mode = true;
    static double accum = 0.0f, vpwa = 0.0f;

    if (init) {
        bsoc_mode = false;
        bsoc_high = false;
        ha_ac_mode = true;
        accum = 0.0f;
        vpwa = 0.0f;
        return true;
    }
    /*
* running avg filter
*/
    accum = accum - accum / COEFN + E.mvar[V_PWA];
    vpwa = accum / COEFN;

    if ((vpwa >= PV_FULL_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
        if (!bsoc_mode) {
            ResetPI(&E.mode.pid);
        }
        bsoc_mode = true;
        bsoc_high = true;
        if (!ha_ac_mode) {
            ha_ac_on();
            ha_ac_mode = true;
        }

    } else {
        if (bsoc_high) { // turn off at min limit power
            if ((vpwa >= PV_MIN_PWR) && (E.mvar[V_FBEKW] >= MIN_BAT_KW_BSOC_HI)) {
                bsoc_mode = true;
                if (ha_ac_mode) {
                    ha_ac_off();
                    ha_ac_mode = false;
                }
            } else {
                bsoc_high = false;
                ha_ac_mode = false;
            }
        }
    }

    E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) + E.mvar[V_DPPV]; // use as a temp variable
    E.mode.total_system = (E.mvar[V_FLO] - E.mode.gti_dumpload) + E.mvar[V_DPPV] +(E.print_vars[L3_P]*
  -1.0f);
    E.mode.gti_dumpload = (E.print_vars[L3_P]* -1.0f) - E.mvar[V_DPPV]; // use this value

    /*
* look at system energy balance for power control drive
*/
    if (mode) { // add GTI power from dumpload
        E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + E.mode.gti_dumpload +
  PBAL_OFFSET);
    } else {
        E.mode.error = (int32_t) UpdatePI(&E.mode.pid, E.mvar[V_BEN] + PBAL_OFFSET);
    }

    if (E.mode.error > 0.0f) {
        L.coef = COEF;
    } else {
        L.coef = COEFN;
    }
    E.mode.target = target;
    E.mode.error = round(error_filter(E.mode.error));
    /*
* check for idle flag from HA
*/
    if (E.mode.con6) {
        ha_ac_mode = true;
        bsoc_mode = false;
    }

    /*
* HA start excess button pressed
*/
    if (E.mode.con4) {
        E.dl_excess = true;
        E.mode.con4 = false;
    }

    /*
* HA stop excess button pressed
*/
    if (E.mode.con5) {
        mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 9); // zero power at excess shutdown
        E.dl_excess = false;
        E.mode.con5 = false;
    }

    /*
* DL buffer battery low set-point excess load shutdown
*/
    if (E.mvar[V_DAHBAT] < PV_DL_B_AH_LOW) {
        mqtt_gti_power(E.client_p, TOPIC_P, "Z#", 10); // zero power at excess shutdown
        E.dl_excess = false;
        E.mode.con4 = false;
        E.mode.con5 = false;
    }

    return bsoc_mode;
}
```

4.4.2.11 bsoc_set_std_dev()

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

4.4.2.12 calculateStandardDeviation()

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

4.4.2.13 dc0_filter()

```
double dc0_filter (
        const double raw )
{
    static double accum = 0.0f;
    static double coef = COEFF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.4.2.14 dc1_filter()

```
double dc1_filter (
        const double raw )
{
    static double accum = 0.0f;
    static double coef = COEF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.4.2.15 dc2_filter()

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

```
{
    static double accum = 0.0f;
    static double coef = COEF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.4.2.16 drive0_filter()

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

```
{
    static double accum = 0.0f;
    static double coef = COEF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.4.2.17 drive1_filter()

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

```
{
    static double accum = 0.0f;
    static double coef = COEFF;
    accum = accum - accum / coef + raw;
    return accum / coef;
}
```

4.4.2.18 get_batc_dev()

```
double get_batc_dev (
              void  )
```

```
{
    return L.batc_std_dev;
}
```

4.4.2.19 gti_test()

```
double gti_test (
              void  )
```

```
{
    // check for 48VDC AC charger powered from the Solar battery bank AC inverter
    if (((L.pv_voltage < MIN_PV_VOLTS) && (!E.dl_excess)) || (L.bat_voltage < MIN_BAT_VOLTS)) {
        L.gti_weight = 0.0f; // reduce power to zero
#ifdef BSOC_DEBUG
        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);
#endif
    } else {
        if (E.dl_excess) {
            if (E.mvar[V_DAHBAT] > PV_DL_B_AH_MIN) {
                L.gti_weight = PV_DL_EXCESS + E.dl_excess_adj;
            } else {
                L.gti_weight = 0.0f; // reduce power to zero
            }
        }
    }
    return dc0_filter(L.gti_weight);
}
```

4.5 bsoc.h

Go to the documentation of this file.

```c
/*
 * File:      bsoc.h
 * Author:  root
 *
 * Created on February 10, 2024, 6:24 PM
 */

#ifndef BSOC_H
#define BSOC_H

#ifdef __cplusplus
extern "C" {
#endif
#include <math.h>
    //#define BSOC_DEBUG

#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 // postive bias for control point
#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

#include <stdlib.h>
#include <stdio.h> /* for printf() */
#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 <math.h>
#include "pid.h"
#include "mqtt_rec.h"

    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);

#ifdef __cplusplus
}
#endif

#endif /* BSOC_H */

```

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

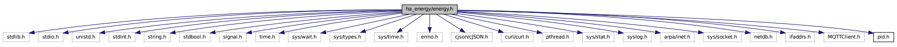
4.17 ha_energy/build/Release/GNU-Linux/pid.o.d File Reference

4.18 ha_energy/energy.h File Reference

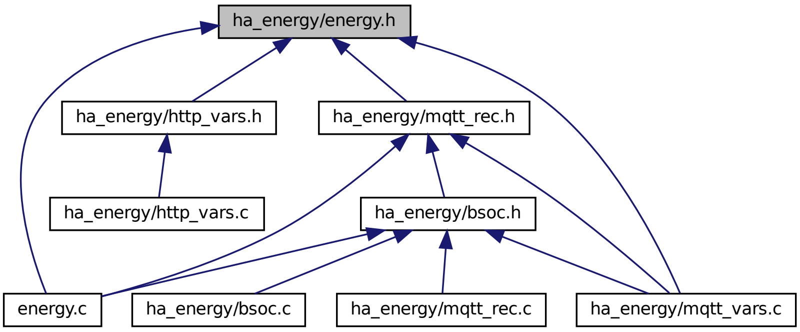
```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <stdbool.h>
#include <signal.h>
#include <time.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/time.h>
#include <errno.h>
```

```
#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	

```
                      {
        E_INIT,
        E_RUN,
        E_WAIT,
        E_IDLE,
        E_STOP,
        E_LAST,
    };
```

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	

```
                     {
        IA_VOLTAGE,
        IA_CURRENT,
        IA_POWER,
        IA_IMPORT,
        IA_EXPORT,
        IA_FREQ,
        IA_PF,
        IA_LAST,
    };
```

4.18.2.3 iammeter_phase

```
enum iammeter_phase
```

Enumerator

PHASE_A	
PHASE_B	
PHASE_C	
PHASE_LAST	

```
                        {
        PHASE_A,
        PHASE_B,
        PHASE_C,
        PHASE_LAST,
    };
```

4.18.2.4 mqtt_vars

```
enum mqtt_vars
```

Enumerator

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

```
                   {
        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,
    // add other data ranges here
    V_DVPV,
    V_DPPV,
    V_DPBAT,
    V_DVBAT,
    V_DCMPPT,
    V_DPMPPT,
    V_DAHBAT,
    V_DCCMODE,
    V_DGTI,
    V_DLAST,
};
```

4.18.2.5 running_state

```
enum running_state
```

Enumerator

R_INIT	
R_FLOAT	
R_SLEEP	
R_RUN	
R_IDLE	
R_LAST	

```
                   {
    R_INIT,
    R_FLOAT,
    R_SLEEP,
    R_RUN,
    R_IDLE,
    R_LAST,
};
```

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	

```
                    {
        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,
        // add other data ranges here
        S_DVPV,
        S_DPPV,
        S_DPBAT,
        S_DVBAT,
        S_DCMPPT,
        S_DPMPPT,
        S_DAHBAT,
        S_DCCMODE,
        S_DGTI,
        S_DLAST,
    };
```

4.18.3 Function Documentation

4.18.3.1 connlost()

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

```
{
    struct ha_flag_type *ha_flag = context;
    int32_t id_num;

    // bug-out if no context variables passed to callback
    if (context == NULL) {
        id_num = -1;
    } else {
        id_num = ha_flag->ha_id;
    }
    fprintf(fout, "\n%s Connection lost, exit ha_energy program\n", log_time(false));
    fprintf(fout, "%s     cause:  %s, %d\n", log_time(false), cause, id_num);
    fprintf(fout, "%sDAEMON failure  LOG Version %s :  MQTT Version %s\n", log_time(false), LOG_VERSION, MQTT_VERSION);
    fflush(fout);
    exit(EXIT_FAILURE);
}
```

4.18.3.2 ha_ac_off()

```
void ha_ac_off (
             void   )
```

```
{
    mqtt_ha_switch(E.client_p, TOPIC_PACC, false);
    E.ac_sw_status = false;
}
```

4.18.3.3 ha_ac_on()

```
void ha_ac_on (
             void   )
```

```
{
    mqtt_ha_switch(E.client_p, TOPIC_PACC, true);
    E.ac_sw_status = true;
}
```

4.18.3.4 ha_dc_off()

```
void ha_dc_off (
             void   )
```

```
{
    mqtt_ha_switch(E.client_p, TOPIC_PDCC, false);
    E.gti_sw_status = false;
}
```

4.18.3.5 ha_dc_on()

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

4.18.3.6 iammeter_read()

```
void iammeter_read (
            void )
{

    curl = curl_easy_init();
    if (curl) {
        E.link.iammeter_count++;
        curl_easy_setopt(curl, CURLOPT_URL, "http://10.1.1.101/monitorjson");
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, iammeter_write_callback);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, E.print_vars); // external data array for iammeter values

        res = curl_easy_perform(curl);
        /* Check for errors */
        if (res != CURLE_OK) {
            fprintf(fout, "curl_easy_perform() failed in iammeter_read:  %s\n",
                curl_easy_strerror(res));
            E.iammeter = false;
            E.link.iammeter_error++;
        } else {
            E.iammeter = true;
        }
        curl_easy_cleanup(curl);
    }
}
```

4.18.3.7 iammeter_write_callback()

```
size_t iammeter_write_callback (
            char * buffer,
            size_t size,
            size_t nitems,
            void * stream )
{
    cJSON *json = cJSON_ParseWithLength(buffer, strlen(buffer));
    struct energy_type * e = stream;
    uint32_t next_var = 0;

    E.link.iammeter_count++;

    if (json == NULL) {
        const char *error_ptr = cJSON_GetErrorPtr();
        E.link.iammeter_error++;
        if (error_ptr != NULL) {
            fprintf(fout, "Error in iammeter_write_callback %u:  %s\n", E.link.iammeter_error, error_ptr);
        }
        goto iammeter_exit;
    }
#ifdef IM_DEBUG
    fprintf(fout, "\n iammeter_read_callback %s \n", buffer);
#endif

    cJSON *data_result = cJSON_GetObjectItemCaseSensitive(json, "Datas");

    if (!data_result) {
        size = 0;
        nitems = 0;
        goto iammeter_exit;
    }

    cJSON *jname;
    uint32_t phase = PHASE_A;

    cJSON_ArrayForEach(jname, data_result)
    {
        cJSON *ianame;
#ifdef IM_DEBUG
        fprintf(fout, "\n iammeter variables ");
#endif

        cJSON_ArrayForEach(ianame, jname)
        {
            uint32_t phase_var = IA_VOLTAGE;
            iammeter_get_data(ianame->valuedouble, phase_var, phase);
            e->print_vars[next_var++] = ianame->valuedouble;
#ifdef IM_DEBUG
            fprintf(fout, "%8.2f ", im_vars[phase_var][phase]);
#endif
            phase_var++;
        }
        phase++;
    }
#ifdef IM_DEBUG
    fprintf(fout, "\n");
#endif

iammeter_exit:
    cJSON_Delete(json);
    return size * nitems;
}
```

4.18.3.8 log_time()

```
char * log_time (
               bool log )
{
    static char time_log[RBUF_SIZ] = {0};
    static uint32_t len = 0, sync_time = TIME_SYNC_SEC - 1;
    time_t rawtime_log;

    tzset();
    timezone = 0;
    daylight = 0;
    time(&rawtime_log);
    if (sync_time++ > TIME_SYNC_SEC) {
        sync_time = 0;
        snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti time commands
        mqtt_gti_time(E.client_p, TOPIC_P, time_log);
    }

    sprintf(time_log, "%s", ctime(&rawtime_log));
    len = strlen(time_log);
    time_log[len - 1] = 0; // munge out the return character
    if (log) {
        fprintf(fout, "%s ", time_log);
        fflush(fout);
    }

    return time_log;
}
```

4.18.3.9 log_timer()

```
bool log_timer (
               void   )
```

```
{
    bool itstime = false;

    if (E.log_spam < LOW_LOG_SPAM) {
        E.log_time_reset = 0;
        itstime = true;
    }
    if (E.log_time_reset > RESET_LOG_SPAM) {
        E.log_spam = 0;
        itstime = true;
    }
    return itstime;
}
```

4.18.3.10 print_im_vars()

```
void print_im_vars (
              void   )
```

```
{
    static char time_log[RBUF_SIZ] = {0};
    static uint32_t sync_time = TIME_SYNC_SEC - 1;
    time_t rawtime_log;
    char imvars[SYSLOG_SIZ];

    fflush(fout);
    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]);
    fprintf(fout, "%s", imvars);
    fflush(fout);
    time(&rawtime_log);
    if (sync_time++ > TIME_SYNC_SEC) {
        sync_time = 0;
        snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti time commands
        mqtt_gti_time(E.client_p, TOPIC_P, time_log);
    }
}
```

4.18.3.11 print_mvar_vars()

```
void print_mvar_vars (
              void   )
```

```
{
    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",
        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],
        (int32_t) E.dc_mismatch, (int32_t) E.ac_mismatch, (int32_t) E.mode_mismatch);
}
```

4.18.3.12 ramp_down_ac()

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

```
{
    if (sw_off) {
        mqtt_ha_switch(client_p, TOPIC_PACC, false);
        E.ac_sw_status = false;
        usleep(500000);
    }
    E.once_ac = true;
}
```

4.18.3.13 ramp_down_gti()

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

```
{
    if (sw_off) {
        mqtt_ha_switch(client_p, TOPIC_PDCC, false);
        E.once_gti_zero = true;
        E.gti_sw_status = false;
    }
    E.once_gti = true;

    if (E.once_gti_zero) {
        mqtt_gti_power(client_p, TOPIC_P, "Z#", 7); // zero power
        E.once_gti_zero = false;
    }
}
```

4.18.3.14 ramp_up_ac()

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

```
{

    if (start) {
        E.once_ac = true;
    }

    if (E.once_ac) {
        E.once_ac = false;
        mqtt_ha_switch(client_p, TOPIC_PACC, true);
        E.ac_sw_status = true;
        usleep(500000); // wait for voltage to ramp
    }
}
```

4.18.3.15 ramp_up_gti()

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

```
{
    static uint32_t sequence = 0;

    if (start) {
        E.once_gti = true;
    }

    if (E.once_gti) {
        E.once_gti = false;
        sequence = 0;
        if (!excess) {
            mqtt_ha_switch(client_p, TOPIC_PDCC, true);
            E.gti_sw_status = true;
            usleep(500000); // wait for voltage to ramp
        } else {
            sequence = 1;
        }
    }

    switch (sequence) {
    case 4:
        E.once_gti_zero = true;
        break;
    case 3:
    case 2:
    case 1:
        E.once_gti_zero = true;
        if (bat_current_stable() || E.dl_excess) { // check battery current std dev, stop ’motorboating’
            sequence++;
            if (!mqtt_gti_power(client_p, TOPIC_P, "+#", 3)) {
                sequence = 0;
            }; // +100W power
        } else {
            usleep(500000); // wait a bit more for power to be stable
            sequence = 1; // do power ramps when ready
            if (!mqtt_gti_power(client_p, TOPIC_P, "-#", 4)) {
                sequence = 0;
            }; // - 100W power
        }
        break;
    case 0:
        sequence++;
        if (E.once_gti_zero) {
            mqtt_gti_power(client_p, TOPIC_P, "Z#", 5); // zero power
            E.once_gti_zero = false;
        }
        break;
    default:
        if (E.once_gti_zero) {
            mqtt_gti_power(client_p, TOPIC_P, "Z#", 6); // zero power
            E.once_gti_zero = false;
        }
        sequence = 0;
        break;
    }
}
```

4.18.3.16 sanity_check()

```
bool sanity_check (
            void  )
```

```
{
    if (E.mvar[V_PWA] > PWA_SANE) {
        E.sane = S_PWA;
        return false;
    }
    if (E.mvar[V_PAMPS] > PAMPS_SANE) {
        E.sane = S_PAMPS;
        return false;
    }
    if (E.mvar[V_PVOLTS] > PVOLTS_SANE) {
        E.sane = S_PVOLTS;
        return false;
    }
    if (E.mvar[V_FBAMPS] > BAMPS_SANE) {
        E.sane = S_FBAMPS;
        return false;
    }
    return true;
}
```

4.18.3.17 sync_ha()

```
bool sync_ha (
            void  )
```

```
{
    bool sync = false;
    if (E.gti_sw_status != (bool) ((int32_t) E.mvar[V_HDCSW])) {
        fprintf(fout, "DC_MM %d %d ", (bool) E.gti_sw_status, (bool) ((int32_t) E.mvar[V_HDCSW]));
        mqtt_ha_switch(E.client_p, TOPIC_PDCC, !E.gti_sw_status);
        E.dc_mismatch = true;
        fflush(fout);
        sync = true;
    } else {
        E.dc_mismatch = false;
    }

    E.ac_sw_status = (bool) ((int32_t) E.mvar[V_HACSW]); // TEMP FIX for MISmatch errors
    if (E.ac_sw_status != (bool) ((int32_t) E.mvar[V_HACSW])) {
        fprintf(fout, "AC_MM %d %d ", (bool) E.ac_sw_status, (bool) ((int32_t) E.mvar[V_HACSW]));
        mqtt_ha_switch(E.client_p, TOPIC_PACC, !E.ac_sw_status);
        E.ac_mismatch = true;
        fflush(fout);
        sync = true;
    } else {
        E.ac_mismatch = false;
    }
    return sync;
}
```

4.18.3.18 timer_callback()

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

```
{
    signal(signum, timer_callback);
    ha_flag_vars_ss.runner = true;
    E.ten_sec_clock++;
    E.log_spam++;
    E.log_time_reset++;
    if (E.log_spam > MAX_LOG_SPAM) {
        E.log_spam = 0;
    }
}
```

4.18.4 Variable Documentation

4.18.4.1 E

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

4.18.4.2 fout

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

4.18.4.3 ha_flag_vars_ss

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

4.19 energy.h

Go to the documentation of this file.

```
/*
* File:      bmc.h
* Author:  root
*
* Created on September 21, 2012, 12:54 PM
*/

#ifndef BMC_H
#define BMC_H

#ifdef __cplusplus
extern "C" {
#endif
#include <stdlib.h>
#include <stdio.h> /* for printf() */
#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"


#define LOG_VERSION     "V0.73"
#define MQTT_VERSION    "V3.11"
#define TNAME  "maint9"
#define LADDRESS        "tcp://127.0.0.1:1883"
#ifdef __amd64
#define ADDRESS         "tcp://10.1.1.172:1883"
#else
#define ADDRESS         "tcp://10.1.1.30:1883"
#endif
#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  // short buffer string size
#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
```

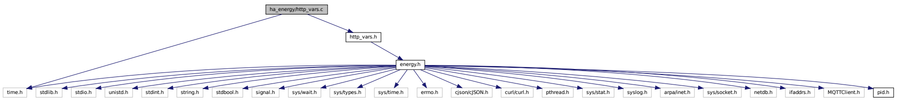
```
#define MAX_ERROR       5
#define IAM_DELAY       120

#define CMD_SEC         10
#define TIME_SYNC_SEC   30

    /*
* Battery SoC cycle limits parameters
*/
#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

    /*
* PV panel cycle limits parameters
*/
#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 // DL battery should be at least 175Ah
#define PV_DL_B_V_LOW       23.8f // Battery low-voltqage cutoff
#define PWA_SLEEP          200.0f
#define DL_AC_DC_EFF        1.24f

    /*
* Energy control loop parameters
*/
#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_DEBUG                      // WEM3080T LOGGING
    //#define B_ADJ_DEBUG                   // debug printing
    //#define FAKE_VPV                      // NO AC CHARGER for DUMPLOAD, batteries are cross-connected to a parallel bank
    //#define PSW_DEBUG
    //#define DEBUG_SHUTDOWN

    //#define AUTO_CHARGE                   // turn on dumpload charger during restarts
    //#define B_DLE_DEBUG    // Dump Load debugging
    //#define BSOC_DEGUB
    //#define DEBUG_HA_CMD

#define IM_DELAY            1   // tens of second updates
#define IM_DISPLAY          1
#define GTI_DELAY           1

    /*
* sane limits for system data elements
*/
#define PWA_SANE            1700.0f
#define PAMPS_SANE          16.0f
#define PVOLTS_SANE         150.0f
#define BAMPS_SANE          70.0f

    /*
```

```
Three Phase WiFi Energy Meter (WEM3080T)
name    Unit    Description
wem3080t_voltage_a  V   A phase voltage
wem3080t_current_a  A   A phase current
wem3080t_power_a    W   A phase active power
wem3080t_importenergy_a kWh A phase import energy
wem3080t_exportgrid_a   kWh A phase export energy
wem3080t_frequency_a    kWh A phase frequency
wem3080t_pf_a   kWh A phase power factor
wem3080t_voltage_b  V   B phase voltage
wem3080t_current_b  A   B phase current
wem3080t_power_b    W   B phase active power
wem3080t_importenergy_b kWh B phase import energy
wem3080t_exportgrid_b   kWh B phase export energy
wem3080t_frequency_b    kWh B phase frequency
wem3080t_pf_b   kWh B phase power factor
wem3080t_voltage_c  V   C phase voltage
wem3080t_current_c  A   C phase current
wem3080t_power_c    W   C phase active power
wem3080t_importenergy_c kWh C phase import energy
wem3080t_exportgrid_c   kWh C phase export energy
wem3080t_frequency_c    kWh C phase frequency
wem3080t_pf_c   kWh C phase power factor
*/

    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,
        // add other data ranges here
        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,
        // add other data ranges here
        S_DVPV,
        S_DPPV,
        S_DPBAT,
        S_DVBAT,
        S_DCMPPT,
        S_DPMPPT,
        S_DAHBAT,
        S_DCCMODE,
        S_DGTI,
        S_DLAST,
    };

#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

    struct link_type {
        volatile uint32_t iammeter_error, iammeter_count;
        volatile uint32_t mqtt_error, mqtt_count;
        volatile uint32_t shutdown;
    };

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

    struct energy_type {
        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, once_ac, iammeter, fm80, dumpload, homeassistant, once_gti_zero;
        volatile double gti_low_adj, ac_low_adj, dl_excess_adj;
        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;
        volatile uint32_t speed_go, im_delay, im_display, gti_delay;
        volatile int32_t rc, sane;
        volatile uint32_t ten_sec_clock, log_spam, log_time_reset;
        pthread_mutex_t ha_lock;
        struct mode_type mode;
        struct link_type link;
        MQTTClient client_p, client_sd, client_ha;
    };

    extern struct energy_type E;
    extern struct ha_flag_type ha_flag_vars_ss;
    extern FILE* fout;

    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);

#ifdef __cplusplus
}
#endif

#endif /* BMC_H */

```

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()

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

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

4.20.1.2 iammeter_read()

```
void iammeter_read (
            void  )
```

```
{

    curl = curl_easy_init();
    if (curl) {
        E.link.iammeter_count++;
        curl_easy_setopt(curl, CURLOPT_URL, "http://10.1.1.101/monitorjson");
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, iammeter_write_callback);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, E.print_vars); // external data array for iammeter values

        res = curl_easy_perform(curl);
        /* Check for errors */
        if (res != CURLE_OK) {
            fprintf(fout, "curl_easy_perform() failed in iammeter_read:  %s\n",
                curl_easy_strerror(res));
            E.iammeter = false;
            E.link.iammeter_error++;
        } else {
            E.iammeter = true;
        }
        curl_easy_cleanup(curl);
    }
}
```

4.20.1.3 iammeter_write_callback()

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

```
{
    cJSON *json = cJSON_ParseWithLength(buffer, strlen(buffer));
    struct energy_type * e = stream;
    uint32_t next_var = 0;

    E.link.iammeter_count++;

    if (json == NULL) {
        const char *error_ptr = cJSON_GetErrorPtr();
        E.link.iammeter_error++;
        if (error_ptr != NULL) {
            fprintf(fout, "Error in iammeter_write_callback %u:  %s\n", E.link.iammeter_error, error_ptr);
        }
        goto iammeter_exit;
    }
#ifdef IM_DEBUG
    fprintf(fout, "\n iammeter_read_callback %s \n", buffer);
#endif

    cJSON *data_result = cJSON_GetObjectItemCaseSensitive(json, "Datas");

    if (!data_result) {
        size = 0;
        nitems = 0;
        goto iammeter_exit;
    }

    cJSON *jname;
    uint32_t phase = PHASE_A;

    cJSON_ArrayForEach(jname, data_result)
    {
        cJSON *ianame;
#ifdef IM_DEBUG
        fprintf(fout, "\n iammeter variables ");
#endif

        cJSON_ArrayForEach(ianame, jname)
        {
            uint32_t phase_var = IA_VOLTAGE;
            iammeter_get_data(ianame->valuedouble, phase_var, phase);
            e->print_vars[next_var++] = ianame->valuedouble;
#ifdef IM_DEBUG
            fprintf(fout, "%8.2f ", im_vars[phase_var][phase]);
#endif
            phase_var++;
        }
        phase++;
    }
#ifdef IM_DEBUG
    fprintf(fout, "\n");
#endif

iammeter_exit:
    cJSON_Delete(json);
    return size * nitems;
}
```

4.20.1.4 mqtt_gti_time()

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

```
{
    bool ret = true;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    ha_flag_vars_ss.deliveredtoken = 0;

    E.link.mqtt_count++;
    pubmsg.payload = msg;
    pubmsg.payloadlen = strlen(msg);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;

    MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run time commands

    // a busy, wait loop for the async delivery thread to complete
    {
        uint32_t waiting = 0;
        while (ha_flag_vars_ss.deliveredtoken != token) {
            usleep(GTI_TOKEN_DELAY);
            if (waiting++ > MQTT_TIMEOUT) {
                fprintf(fout, "\r\n%s GTI Time Still Waiting, timeout\r\n", log_time(false));
                break;
            }
        };
    }
    usleep(HA_SW_DELAY);
    return ret;
}
```

4.20.1.5 print_im_vars()

```
void print_im_vars (
            void )
{
    static char time_log[RBUF_SIZ] = {0};
    static uint32_t sync_time = TIME_SYNC_SEC - 1;
    time_t rawtime_log;
    char imvars[SYSLOG_SIZ];

    fflush(fout);
    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]);
    fprintf(fout, "%s", imvars);
    fflush(fout);
    time(&rawtime_log);
    if (sync_time++ > TIME_SYNC_SEC) {
        sync_time = 0;
        snprintf(time_log, RBUF_SIZ - 1, "VT%lut", rawtime_log); // format for dumpload controller gti time commands
        mqtt_gti_time(E.client_p, TOPIC_P, time_log);
    }
}
```

4.20.2 Variable Documentation

4.20.2.1 curl

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

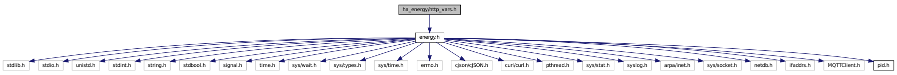
4.20.2.2 res

```
CURLcode res [static]
```

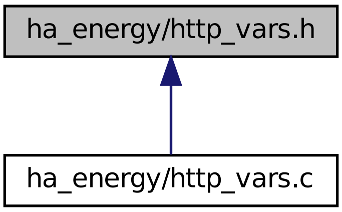
4.21 ha_energy/http_vars.h File Reference

```
#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

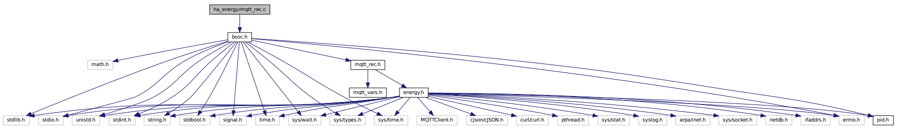
Go to the documentation of this file.

```
/*
* File:      http_vars.h
* Author:  root
*
* Created on February 16, 2024, 8:37 AM
*/

#ifndef HTTP_VARS_H
#define HTTP_VARS_H

#ifdef __cplusplus
extern "C" {
#endif

#include "energy.h"

    extern FILE* fout;

#ifdef __cplusplus
}
#endif

#endif /* HTTP_VARS_H */

```

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 )
```

```
{
    struct ha_flag_type *ha_flag = context;

    // bug-out if no context variables passed to callback
    if (context == NULL) {
        return;
    }
    ha_flag->deliveredtoken = dt;
}
```

4.23.1.2 fm80_float()

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

```
{
    if ((uint32_t) E.mvar[V_FCCM] == FLOAT_CODE) {
        if (set_bias) {
            E.mode.pv_bias = PV_BIAS_FLOAT;
        }
        if (E.mode.R != R_IDLE) {
            E.mode.R = R_FLOAT;
        }
        return true;
    } else {
        if (E.mode.R == R_FLOAT) {
            E.mode.R = R_RUN;
        }
    }
    return false;
}
```

4.23.1.3 fm80_sleep()

```
bool fm80_sleep (
               void )
```

```
{
    if ((uint32_t) E.mvar[V_FCCM] == SLEEP_CODE) {
        return true;
    }
    return false;
}
```

4.23.1.4 json_get_data()

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

```
{
    bool ret = false;
    static uint32_t j = 0;

    // access the JSON data using the lookup string passed in data_id
    name = cJSON_GetObjectItemCaseSensitive(json_src, data_id);

    /*
* process string values
*/
    if (cJSON_IsString(name) && (name->valuestring != NULL)) {
#ifdef GET_DEBUG
        fprintf(fout, "%s Name:  %s\n", data_id, name->valuestring);
#endif
        ret = true;
    }

    /*
* process numeric values
*/
    if (cJSON_IsNumber(name)) {
#ifdef GET_DEBUG
        fprintf(fout, "%s Value:  %f\n", data_id, name->valuedouble);
#endif
        if (i > V_DLAST) { // check for out-of-range index
            i = V_DLAST;
        }

        // lock the main value array during updates
        pthread_mutex_lock(&E.ha_lock);
        E.mvar[i] = name->valuedouble;
        pthread_mutex_unlock(&E.ha_lock);

        /*
* special processing for variable data received
*/
        if (i == V_DCMPPT) {
            /*
* load battery current standard deviation array bat_c_std_dev with data
*/
            bsoc_set_std_dev(E.mvar[i], j++);
            if (j >= RDEV_SIZE) {
                j = 0;
            }
        }
        /*
* update local MATTER switch status from HA
*/
        if (i == V_HDCSW) {
            E.gti_sw_status = (bool) ((int32_t) E.mvar[i]);
            E.dc_mismatch = false;
        }

        if (i == V_HACSW) {
            E.ac_sw_status = (bool) ((int32_t) E.mvar[i]);
            E.ac_mismatch = false;
        }

        // command HA_ENERGY to shutdown mode
        if (i == V_HSHUT) {
            E.solar_shutdown = (bool) ((int32_t) E.mvar[i]);
        }
        // set HA_ENERGY energy processing mode
        if (i == V_HMODE) {
            ha_flag_vars_ss.energy_mode = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON0) {
            E.mode.con0 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON1) {
            E.mode.con1 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON2) {
            E.mode.con2 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON3) {
            E.mode.con3 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON4) { // set DL GTI excess load MODE
            E.mode.con4 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON5) { // clear DL GTI excess load MODE
            E.mode.con5 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON6) { // HA Energy program idle
            E.mode.con6 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON7) { // HA Energy program exit
            E.mode.con7 = (bool) ((int32_t) E.mvar[i]);
        }
        ret = true;
    }
    return ret;
}
```

4.23.1.5 msgarrvd()

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

```
{
    int32_t i, ret = 1;
    const char* payloadptr;
    char buffer[MBMQTT];
    struct ha_flag_type *ha_flag = context;

    E.link.mqtt_count++;
    // bug-out if no context variables passed to callback
    if (context == NULL) {
        ret = -1;
        goto null_exit;
    }

#ifdef DEBUG_REC
    fprintf(fout, "Message arrived\n");
#endif
    /*
* move the received message into a processing holding buffer
*/
    payloadptr = message->payload;
    for (i = 0; i < message->payloadlen; i++) {
        buffer[i] = *payloadptr++;
    }
    buffer[i] = 0; // make a null terminated C string

    // parse the JSON data in the holding buffer
    cJSON *json = cJSON_ParseWithLength(buffer, message->payloadlen);
    if (json == NULL) {
        const char *error_ptr = cJSON_GetErrorPtr();
        if (error_ptr != NULL) {
            fprintf(fout, "%s Error:  %s NULL cJSON pointer\n", log_time(false), error_ptr);
        }
        ret = -1;
        ha_flag->rec_ok = false;
        E.fm80 = false;
        E.dumpload = false;
        E.homeassistant = false;
        E.link.mqtt_error++;
        goto error_exit;
    }

    /*
* MQTT messages from the FM80 Q84 interface
*/
    if (ha_flag->ha_id == FM80_ID) {
#ifdef DEBUG_REC
        fprintf(fout, "FM80 MQTT data\r\n");
#endif
        cJSON *data_result = json;

        for (uint32_t ii = V_FCCM; ii < V_FLAST; ii++) {
            if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
                ha_flag->var_update++;
            }
        }
        E.fm80 = true;
    }

    /*
* MQTT messages from the K42 dumpload/gti interface
*/
    if (ha_flag->ha_id == DUMPLOAD_ID) {
#ifdef DEBUG_REC
        fprintf(fout, "DUMPLOAD MQTT data\r\n");
#endif
        cJSON *data_result = json;

        for (uint32_t ii = V_HDCSW; ii < V_DLAST; ii++) {
            if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
                ha_flag->var_update++;
            }
        }
        E.dumpload = true;
    }

    /*
* MQTT messages from the Linux HA_ENERGY interface
*/
    if (ha_flag->ha_id == HA_ID) {
#ifdef DEBUG_REC
        fprintf(fout, "Home Assistant MQTT data\r\n");
#endif
        cJSON *data_result = json;

        if (json_get_data(json, mqtt_name[V_HACSW], data_result, V_HACSW)) {
            ha_flag->var_update++;
        }
        data_result = json;
        if (json_get_data(json, mqtt_name[V_HDCSW], data_result, V_HDCSW)) {
            ha_flag->var_update++;
        }

        E.homeassistant = true;
    }

    // done with processing MQTT async message, set state flags
    ha_flag->receivedtoken = true;
    ha_flag->rec_ok = true;
    /*
* exit and delete/free resources.  In steps depending of possible error conditions
*/
error_exit:
    // delete the JSON object
    cJSON_Delete(json);
null_exit:
    // free the MQTT objects
    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
    return ret;
}
```

4.23.1.6 print_mvar_vars()

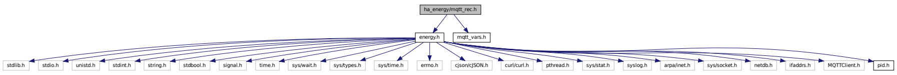
```
void print_mvar_vars (
            void  )
```

```
{
    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",
        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],
        (int32_t) E.dc_mismatch, (int32_t) E.ac_mismatch, (int32_t) E.mode_mismatch);
}
```

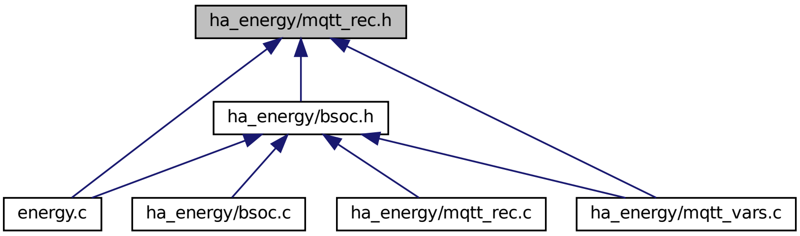
4.24 ha_energy/mqtt_rec.h File Reference

```
#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	

```
             {
    P8055_ID,
    FM80_ID,
    DUMPLOAD_ID,
HA_ID,
    LAST_MQTT_ID,
};
```

4.24.3 Function Documentation

4.24.3.1 delivered()

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

```
{
    struct ha_flag_type *ha_flag = context;

    // bug-out if no context variables passed to callback
    if (context == NULL) {
        return;
    }
    ha_flag->deliveredtoken = dt;
}
```

4.24.3.2 fm80_float()

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

```
{
    if ((uint32_t) E.mvar[V_FCCM] == FLOAT_CODE) {
        if (set_bias) {
            E.mode.pv_bias = PV_BIAS_FLOAT;
        }
        if (E.mode.R != R_IDLE) {
            E.mode.R = R_FLOAT;
        }
        return true;
    } else {
        if (E.mode.R == R_FLOAT) {
            E.mode.R = R_RUN;
        }
    }
    return false;
}
```

4.24.3.3 fm80_sleep()

```
bool fm80_sleep (
              void )
```

```
{
    if ((uint32_t) E.mvar[V_FCCM] == SLEEP_CODE) {
        return true;
    }
    return false;
}
```

4.24.3.4 json_get_data()

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

```
{
    bool ret = false;
    static uint32_t j = 0;

    // access the JSON data using the lookup string passed in data_id
    name = cJSON_GetObjectItemCaseSensitive(json_src, data_id);

    /*
* process string values
*/
    if (cJSON_IsString(name) && (name->valuestring != NULL)) {
#ifdef GET_DEBUG
        fprintf(fout, "%s Name:  %s\n", data_id, name->valuestring);
#endif
        ret = true;
    }

    /*
* process numeric values
*/
    if (cJSON_IsNumber(name)) {
#ifdef GET_DEBUG
        fprintf(fout, "%s Value:  %f\n", data_id, name->valuedouble);
#endif
        if (i > V_DLAST) { // check for out-of-range index
            i = V_DLAST;
        }

        // lock the main value array during updates
        pthread_mutex_lock(&E.ha_lock);
        E.mvar[i] = name->valuedouble;
        pthread_mutex_unlock(&E.ha_lock);

        /*
* special processing for variable data received
*/
        if (i == V_DCMPPT) {
            /*
* load battery current standard deviation array bat_c_std_dev with data
*/
            bsoc_set_std_dev(E.mvar[i], j++);
            if (j >= RDEV_SIZE) {
                j = 0;
            }
        }
        /*
* update local MATTER switch status from HA
*/
        if (i == V_HDCSW) {
            E.gti_sw_status = (bool) ((int32_t) E.mvar[i]);
            E.dc_mismatch = false;
        }

        if (i == V_HACSW) {
            E.ac_sw_status = (bool) ((int32_t) E.mvar[i]);
            E.ac_mismatch = false;
        }

        // command HA_ENERGY to shutdown mode
        if (i == V_HSHUT) {
            E.solar_shutdown = (bool) ((int32_t) E.mvar[i]);
        }
        // set HA_ENERGY energy processing mode
        if (i == V_HMODE) {
            ha_flag_vars_ss.energy_mode = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON0) {
            E.mode.con0 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON1) {
            E.mode.con1 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON2) {
            E.mode.con2 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON3) {
            E.mode.con3 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON4) { // set DL GTI excess load MODE
            E.mode.con4 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON5) { // clear DL GTI excess load MODE
            E.mode.con5 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON6) { // HA Energy program idle
            E.mode.con6 = (bool) ((int32_t) E.mvar[i]);
        }
        if (i == V_HCON7) { // HA Energy program exit
            E.mode.con7 = (bool) ((int32_t) E.mvar[i]);
        }
        ret = true;
    }
    return ret;
}
```

4.24.3.5 msgarrvd()

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

```
{
    int32_t i, ret = 1;
    const char* payloadptr;
    char buffer[MBMQTT];
    struct ha_flag_type *ha_flag = context;

    E.link.mqtt_count++;
    // bug-out if no context variables passed to callback
    if (context == NULL) {
        ret = -1;
        goto null_exit;
    }

#ifdef DEBUG_REC
    fprintf(fout, "Message arrived\n");
#endif
    /*
* move the received message into a processing holding buffer
*/
    payloadptr = message->payload;
    for (i = 0; i < message->payloadlen; i++) {
        buffer[i] = *payloadptr++;
    }
    buffer[i] = 0; // make a null terminated C string

    // parse the JSON data in the holding buffer
    cJSON *json = cJSON_ParseWithLength(buffer, message->payloadlen);
    if (json == NULL) {
        const char *error_ptr = cJSON_GetErrorPtr();
        if (error_ptr != NULL) {
            fprintf(fout, "%s Error:  %s NULL cJSON pointer\n", log_time(false), error_ptr);
        }
        ret = -1;
        ha_flag->rec_ok = false;
        E.fm80 = false;
        E.dumpload = false;
        E.homeassistant = false;
        E.link.mqtt_error++;
        goto error_exit;
    }

    /*
* MQTT messages from the FM80 Q84 interface
*/
    if (ha_flag->ha_id == FM80_ID) {
#ifdef DEBUG_REC
        fprintf(fout, "FM80 MQTT data\r\n");
#endif
        cJSON *data_result = json;

        for (uint32_t ii = V_FCCM; ii < V_FLAST; ii++) {
            if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
                ha_flag->var_update++;
            }
        }
        E.fm80 = true;
    }

    /*
* MQTT messages from the K42 dumpload/gti interface
*/
    if (ha_flag->ha_id == DUMPLOAD_ID) {
#ifdef DEBUG_REC
        fprintf(fout, "DUMPLOAD MQTT data\r\n");
#endif
        cJSON *data_result = json;

        for (uint32_t ii = V_HDCSW; ii < V_DLAST; ii++) {
            if (json_get_data(json, mqtt_name[ii], data_result, ii)) {
                ha_flag->var_update++;
            }
        }
        E.dumpload = true;
    }

    /*
* MQTT messages from the Linux HA_ENERGY interface
*/
    if (ha_flag->ha_id == HA_ID) {
#ifdef DEBUG_REC
        fprintf(fout, "Home Assistant MQTT data\r\n");
#endif
        cJSON *data_result = json;

        if (json_get_data(json, mqtt_name[V_HACSW], data_result, V_HACSW)) {
            ha_flag->var_update++;
        }
        data_result = json;
        if (json_get_data(json, mqtt_name[V_HDCSW], data_result, V_HDCSW)) {
            ha_flag->var_update++;
        }

        E.homeassistant = true;
    }

    // done with processing MQTT async message, set state flags
    ha_flag->receivedtoken = true;
    ha_flag->rec_ok = true;
    /*
* exit and delete/free resources.  In steps depending of possible error conditions
*/
error_exit:
    // delete the JSON object
    cJSON_Delete(json);
null_exit:
    // free the MQTT objects
    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);
    return ret;
}
```

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.

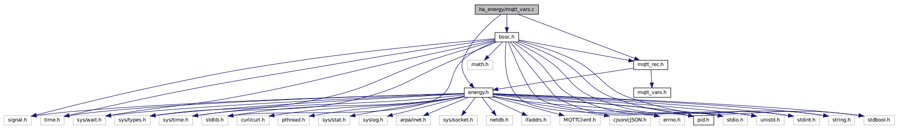
```
/*
* File:      mqtt_rec.h
* Author:  root
*
* Created on February 5, 2024, 2:54 PM
*/

#ifndef MQTT_REC_H
#define MQTT_REC_H

#ifdef __cplusplus
extern "C" {
#endif

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

#define RDEV_SIZE        10

#define SLEEP_CODE      0
#define FLOAT_CODE      1
    //#define DEBUG_REC
    //#define GET_DEBUG

#define MBMQTT      1024

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

    struct ha_flag_type {
        volatile MQTTClient_deliveryToken deliveredtoken, receivedtoken;
        volatile bool runner, rec_ok;
        int32_t ha_id;
        volatile int32_t var_update, energy_mode;
    };

    extern FILE* fout;

    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);

#ifdef __cplusplus
}
#endif

#endif /* MQTT_REC_H */

```

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 )
```

```
{
    bool ret = true;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    ha_flag_vars_ss.deliveredtoken = 0;
    static bool spam = false;

    E.link.mqtt_count++;
    pubmsg.payload = msg;
    pubmsg.payloadlen = strlen(msg);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;

    if (E.dl_excess) { // always run excess commands
        spam = false;
    }
#ifdef GTI_NO_POWER
    MQTTClient_publishMessage(client_p, "mateq84/data/gticmd_nopower", &pubmsg, &token);
#else
    if (bsoc_gti() > MIN_BAT_KW || E.dl_excess) {
#ifdef DEBUG_HA_CMD
        log_time(true);
        fprintf(fout, "HA GTI power command %s, SDEV %5.2f trace %u\r\n", msg, get_batc_dev(), trace);
        fflush(fout);
        spam = true;
#endif
        MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run power commands
    } else {
        ret = false;
        pubmsg.payload = "Z#";
        pubmsg.payloadlen = strlen("Z#");
        if (!spam) {
            MQTTClient_publishMessage(client_p, TOPIC_SPAM, &pubmsg, &token);
        } else {
            MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // only shutdown GTI power
        }
#ifdef DEBUG_HA_CMD
        if (spam) {
            log_time(true);
            fprintf(fout, "HA GTI power set to zero, trace %u\r\n", trace);
            fflush(fout);
            spam = false;
        }
#endif
    }
#endif
    // a busy, wait loop for the async delivery thread to complete
    {
        uint32_t waiting = 0;
        while (ha_flag_vars_ss.deliveredtoken != token) {
            usleep(TOKEN_DELAY);
            if (waiting++ > MQTT_TIMEOUT) {
                fprintf(fout, "\r\n%s GTI Power Still Waiting, timeout\r\n", log_time(false));
                break;
            }
        };
    }
    usleep(HA_SW_DELAY);
    return ret;
}
```

4.26.2.2 mqtt_gti_time()

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

```
{
    bool ret = true;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    ha_flag_vars_ss.deliveredtoken = 0;

    E.link.mqtt_count++;
    pubmsg.payload = msg;
    pubmsg.payloadlen = strlen(msg);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;

    MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run time commands

    // a busy, wait loop for the async delivery thread to complete
    {
        uint32_t waiting = 0;
        while (ha_flag_vars_ss.deliveredtoken != token) {
            usleep(GTI_TOKEN_DELAY);
            if (waiting++ > MQTT_TIMEOUT) {
                fprintf(fout, "\r\n%s GTI Time Still Waiting, timeout\r\n", log_time(false));
                break;
            }
        };
    }
    usleep(HA_SW_DELAY);
    return ret;
}
```

4.26.2.3 mqtt_ha_pid()

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

```
{
    cJSON *json;
    time_t rawtime;

    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    ha_flag_vars_ss.deliveredtoken = 0;

    E.link.mqtt_count++;
    E.mode.sequence++;
    json = cJSON_CreateObject();
    cJSON_AddStringToObject(json, "name", CLIENTID1);
    cJSON_AddNumberToObject(json, "sequence", E.mode.sequence);
    cJSON_AddNumberToObject(json, "mqtt_count", (double) E.link.mqtt_count);
    cJSON_AddNumberToObject(json, "http_count", (double) E.link.iammeter_count);
    cJSON_AddNumberToObject(json, "piderror", E.mode.error);
    cJSON_AddNumberToObject(json, "totalsystem", E.mode.total_system);
    cJSON_AddNumberToObject(json, "gtinet", E.mode.gti_dumpload);
    cJSON_AddNumberToObject(json, "energy_state", (double) E.mode.E);
    cJSON_AddNumberToObject(json, "run_state", (double) E.mode.R);
    // correct for power sensed by GTI metering
    E.mode.off_grid = (E.mvar[V_FLO] - (E.mvar[V_DPPV] * DL_AC_DC_EFF));
    E.mode.off_grid = drive1_filter(E.mode.off_grid);
    if (E.mode.off_grid < 0.0f) { // only see power removed from grid usage
        E.mode.off_grid = 0.0f;
    }
    cJSON_AddNumberToObject(json, "off_grid", E.mode.off_grid);
    cJSON_AddNumberToObject(json, "excess_mode", (double) E.dl_excess);
    cJSON_AddStringToObject(json, "build_date", FW_Date);
    cJSON_AddStringToObject(json, "build_time", FW_Time);
    time(&rawtime);
    cJSON_AddNumberToObject(json, "sequence_time", (double) rawtime);
    // convert the cJSON object to a JSON string
    char *json_str = cJSON_Print(json);

    pubmsg.payload = json_str;
    pubmsg.payloadlen = strlen(json_str);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;

    MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
    // a busy, wait loop for the async delivery thread to complete
    {
        uint32_t waiting = 0;
        while (ha_flag_vars_ss.deliveredtoken != token) {
            usleep(TOKEN_DELAY);
            if (waiting++ > MQTT_TIMEOUT) {
                fprintf(fout, "\r\n%s SW Still Waiting, timeout\r\n", log_time(false));
                break;
            }
        };
    }

    cJSON_free(json_str);
    cJSON_Delete(json);
}
```

4.26.2.4 mqtt_ha_shutdown()

```
void mqtt_ha_shutdown (
              MQTTClient client_p,
              const char * topic_p )
{
    cJSON *json;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    ha_flag_vars_ss.deliveredtoken = 0;

    json = cJSON_CreateObject();
    cJSON_AddStringToObject(json, "shutdown", CLIENTID1);
    char *json_str = cJSON_Print(json);

    pubmsg.payload = json_str;
    pubmsg.payloadlen = strlen(json_str);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;

    MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
    // a busy, wait loop for the async delivery thread to complete
    {
        uint32_t waiting = 0;
        while (ha_flag_vars_ss.deliveredtoken != token) {
            usleep(TOKEN_DELAY);
            if (waiting++ > MQTT_TIMEOUT) {
                fprintf(fout, "\r\n%s SW Still Waiting, timeout\r\n", log_time(false));
                break;
            }
        };
    }
}
```

4.26.2.5 mqtt_ha_switch()

```
void mqtt_ha_switch (
              MQTTClient client_p,
              const char * topic_p,
              const bool sw_state )
{
    cJSON *json;
#ifdef DEBUG_HA_CMD
    static bool spam = false;
    static uint32_t less_spam = 0;
#endif

    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    ha_flag_vars_ss.deliveredtoken = 0;

    E.link.mqtt_count++;
    json = cJSON_CreateObject();
    if (sw_state) {
        cJSON_AddStringToObject(json, "state", "ON");
#ifdef DEBUG_HA_CMD
        spam = true;
        less_spam = 0;
#endif
    } else {
        if ((uint32_t) E.mvar[V_FCCM] != FLOAT_CODE) { // use max power in FLOAT mode
            cJSON_AddStringToObject(json, "state", "OFF");
        } else {
            cJSON_AddStringToObject(json, "state", "ON");
#ifdef DEBUG_HA_CMD
            spam = true;
            less_spam = 0;
#endif
        }
    }
    // convert the cJSON object to a JSON string
    char *json_str = cJSON_Print(json);

    pubmsg.payload = json_str;
    pubmsg.payloadlen = strlen(json_str);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;

#ifdef DEBUG_HA_CMD
    if (spam) {
        log_time(true);
        fflush(fout);
        fprintf(fout, "HA switch command %s, %d\r\n", topic_p, sw_state);
        fflush(fout);
        if (!sw_state) {
            if (less_spam++ > 3) {
                spam = false;
                less_spam = 0;
            }
        }
    }
#endif

    MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
    // a busy, wait loop for the async delivery thread to complete
    {
        uint32_t waiting = 0;
        while (ha_flag_vars_ss.deliveredtoken != token) {
            usleep(TOKEN_DELAY);
            if (waiting++ > MQTT_TIMEOUT) {
#ifdef DEBUG_HA_CMD
                fflush(fout);
                fprintf(fout, "\r\nSW Still Waiting, timeout\r\n");
                fflush(fout);
#endif
                break;
            }
        };
    }

    cJSON_free(json_str);
    cJSON_Delete(json);
    usleep(HA_SW_DELAY);
    fflush(fout);
}
```

4.26.3 Variable Documentation

4.26.3.1 FW_Date

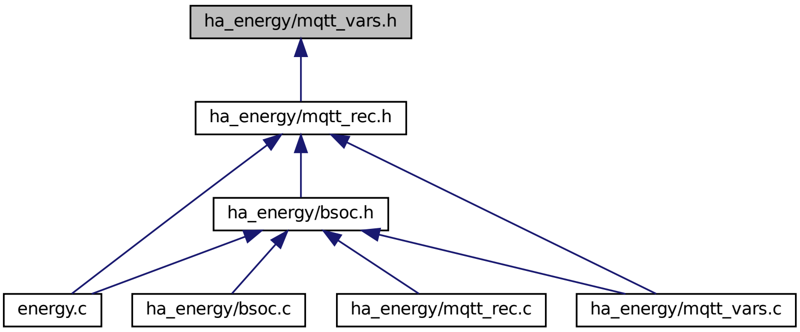
```
const char* const FW_Date = __DATE__  [static]
```

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 )
```

```
{
    bool ret = true;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    ha_flag_vars_ss.deliveredtoken = 0;
    static bool spam = false;

    E.link.mqtt_count++;
    pubmsg.payload = msg;
    pubmsg.payloadlen = strlen(msg);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;

    if (E.dl_excess) { // always run excess commands
        spam = false;
    }
#ifdef GTI_NO_POWER
    MQTTClient_publishMessage(client_p, "mateq84/data/gticmd_nopower", &pubmsg, &token);
#else
    if (bsoc_gti() > MIN_BAT_KW || E.dl_excess) {
#ifdef DEBUG_HA_CMD
        log_time(true);
        fprintf(fout, "HA GTI power command %s, SDEV %5.2f trace %u\r\n", msg, get_batc_dev(), trace);
        fflush(fout);
        spam = true;
#endif
        MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run power commands
    } else {
        ret = false;
        pubmsg.payload = "Z#";
        pubmsg.payloadlen = strlen("Z#");
        if (!spam) {
            MQTTClient_publishMessage(client_p, TOPIC_SPAM, &pubmsg, &token);
        } else {
            MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // only shutdown GTI power
        }
#ifdef DEBUG_HA_CMD
        if (spam) {
            log_time(true);
            fprintf(fout, "HA GTI power set to zero, trace %u\r\n", trace);
            fflush(fout);
            spam = false;
        }
#endif
    }
#endif
    // a busy, wait loop for the async delivery thread to complete
    {
        uint32_t waiting = 0;
        while (ha_flag_vars_ss.deliveredtoken != token) {
            usleep(TOKEN_DELAY);
            if (waiting++ > MQTT_TIMEOUT) {
                fprintf(fout, "\r\n%s GTI Power Still Waiting, timeout\r\n", log_time(false));
                break;
            }
        };
    }
    usleep(HA_SW_DELAY);
    return ret;
}
```

4.27.2.2 mqtt_gti_time()

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

```
{
    bool ret = true;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    ha_flag_vars_ss.deliveredtoken = 0;

    E.link.mqtt_count++;
    pubmsg.payload = msg;
    pubmsg.payloadlen = strlen(msg);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;

    MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token); // run time commands

    // a busy, wait loop for the async delivery thread to complete
    {
        uint32_t waiting = 0;
        while (ha_flag_vars_ss.deliveredtoken != token) {
            usleep(GTI_TOKEN_DELAY);
            if (waiting++ > MQTT_TIMEOUT) {
                fprintf(fout, "\r\n%s GTI Time Still Waiting, timeout\r\n", log_time(false));
                break;
            }
        };
    }
    usleep(HA_SW_DELAY);
    return ret;
}
```

4.27.2.3 mqtt_ha_pid()

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

```
{
    cJSON *json;
    time_t rawtime;

    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    ha_flag_vars_ss.deliveredtoken = 0;

    E.link.mqtt_count++;
    E.mode.sequence++;
    json = cJSON_CreateObject();
    cJSON_AddStringToObject(json, "name", CLIENTID1);
    cJSON_AddNumberToObject(json, "sequence", E.mode.sequence);
    cJSON_AddNumberToObject(json, "mqtt_count", (double) E.link.mqtt_count);
    cJSON_AddNumberToObject(json, "http_count", (double) E.link.iammeter_count);
    cJSON_AddNumberToObject(json, "piderror", E.mode.error);
    cJSON_AddNumberToObject(json, "totalsystem", E.mode.total_system);
    cJSON_AddNumberToObject(json, "gtinet", E.mode.gti_dumpload);
    cJSON_AddNumberToObject(json, "energy_state", (double) E.mode.E);
    cJSON_AddNumberToObject(json, "run_state", (double) E.mode.R);
    // correct for power sensed by GTI metering
    E.mode.off_grid = (E.mvar[V_FLO] - (E.mvar[V_DPPV] * DL_AC_DC_EFF));
    E.mode.off_grid = drive1_filter(E.mode.off_grid);
    if (E.mode.off_grid < 0.0f) { // only see power removed from grid usage
        E.mode.off_grid = 0.0f;
    }
    cJSON_AddNumberToObject(json, "off_grid", E.mode.off_grid);
    cJSON_AddNumberToObject(json, "excess_mode", (double) E.dl_excess);
    cJSON_AddStringToObject(json, "build_date", FW_Date);
    cJSON_AddStringToObject(json, "build_time", FW_Time);
    time(&rawtime);
    cJSON_AddNumberToObject(json, "sequence_time", (double) rawtime);
    // convert the cJSON object to a JSON string
    char *json_str = cJSON_Print(json);

    pubmsg.payload = json_str;
    pubmsg.payloadlen = strlen(json_str);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;

    MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
    // a busy, wait loop for the async delivery thread to complete
    {
        uint32_t waiting = 0;
        while (ha_flag_vars_ss.deliveredtoken != token) {
            usleep(TOKEN_DELAY);
            if (waiting++ > MQTT_TIMEOUT) {
                fprintf(fout, "\r\n%s SW Still Waiting, timeout\r\n", log_time(false));
                break;
            }
        };
    }

    cJSON_free(json_str);
    cJSON_Delete(json);
}
```

4.27.2.4 mqtt_ha_shutdown()

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

```
{
    cJSON *json;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    ha_flag_vars_ss.deliveredtoken = 0;

    json = cJSON_CreateObject();
    cJSON_AddStringToObject(json, "shutdown", CLIENTID1);
    char *json_str = cJSON_Print(json);

    pubmsg.payload = json_str;
    pubmsg.payloadlen = strlen(json_str);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;

    MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
    // a busy, wait loop for the async delivery thread to complete
    {
        uint32_t waiting = 0;
        while (ha_flag_vars_ss.deliveredtoken != token) {
            usleep(TOKEN_DELAY);
            if (waiting++ > MQTT_TIMEOUT) {
                fprintf(fout, "\r\n%s SW Still Waiting, timeout\r\n", log_time(false));
                break;
            }
        };
    }
}
```

4.27.2.5 mqtt_ha_switch()

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

```
{
    cJSON *json;
#ifdef DEBUG_HA_CMD
    static bool spam = false;
    static uint32_t less_spam = 0;
#endif

    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    ha_flag_vars_ss.deliveredtoken = 0;

    E.link.mqtt_count++;
    json = cJSON_CreateObject();
    if (sw_state) {
        cJSON_AddStringToObject(json, "state", "ON");
#ifdef DEBUG_HA_CMD
        spam = true;
        less_spam = 0;
#endif
    } else {
        if ((uint32_t) E.mvar[V_FCCM] != FLOAT_CODE) { // use max power in FLOAT mode
            cJSON_AddStringToObject(json, "state", "OFF");
        } else {
            cJSON_AddStringToObject(json, "state", "ON");
#ifdef DEBUG_HA_CMD
            spam = true;
            less_spam = 0;
#endif
        }
    }
    // convert the cJSON object to a JSON string
    char *json_str = cJSON_Print(json);

    pubmsg.payload = json_str;
    pubmsg.payloadlen = strlen(json_str);
    pubmsg.qos = QOS;
    pubmsg.retained = 0;

#ifdef DEBUG_HA_CMD
    if (spam) {
        log_time(true);
        fflush(fout);
        fprintf(fout, "HA switch command %s, %d\r\n", topic_p, sw_state);
        fflush(fout);
        if (!sw_state) {
            if (less_spam++ > 3) {
                spam = false;
                less_spam = 0;
            }
        }
    }
#endif

    MQTTClient_publishMessage(client_p, topic_p, &pubmsg, &token);
    // a busy, wait loop for the async delivery thread to complete
    {
        uint32_t waiting = 0;
        while (ha_flag_vars_ss.deliveredtoken != token) {
            usleep(TOKEN_DELAY);
            if (waiting++ > MQTT_TIMEOUT) {
#ifdef DEBUG_HA_CMD
                fflush(fout);
                fprintf(fout, "\r\nSW Still Waiting, timeout\r\n");
                fflush(fout);
#endif
                break;
            }
        };
    }

    cJSON_free(json_str);
    cJSON_Delete(json);
    usleep(HA_SW_DELAY);
    fflush(fout);
}
```

4.27.3 Variable Documentation

4.27.3.1 mqtt_name

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

4.28 mqtt_vars.h

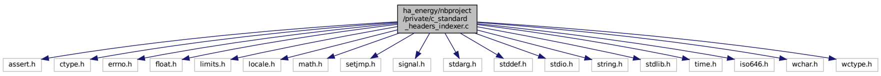
Go to the documentation of this file.

```
/*
* File:      mqtt_vars.h
* Author:  root
*
* Created on February 9, 2024, 6:50 AM
*/

#ifndef MQTT_VARS_H
#define MQTT_VARS_H

#ifdef __cplusplus
extern "C" {
#endif

    //#define GTI_NO_POWER      // do we actually run power commands

    //#define DEBUG_HA_CMD    // show debug text
#define HA_SW_DELAY     400000  // usecs
#define TOKEN_DELAY     250
#define GTI_TOKEN_DELAY 300

#define QOS             1

    extern const char* mqtt_name[V_DLAST];

    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 *);

#ifdef __cplusplus
}
#endif

#endif /* MQTT_VARS_H */

```

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

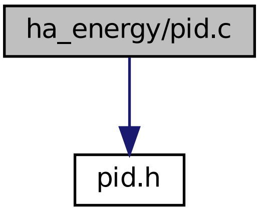
```
#include <cstdlib>
#include <csignal>
#include <csetjmp>
#include <cstdarg>
#include <typeinfo>
#include <bitset>
#include <functional>
#include <utility>
#include <ctime>
#include <cstddef>
#include <new>
#include <memory>
#include <climits>
#include <cfloat>
#include <limits>
#include <exception>
#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 )
```

```
                                               {
    pid->dState = 0.0; // not used but cleared
    pid->iState = 0.0;
}
```

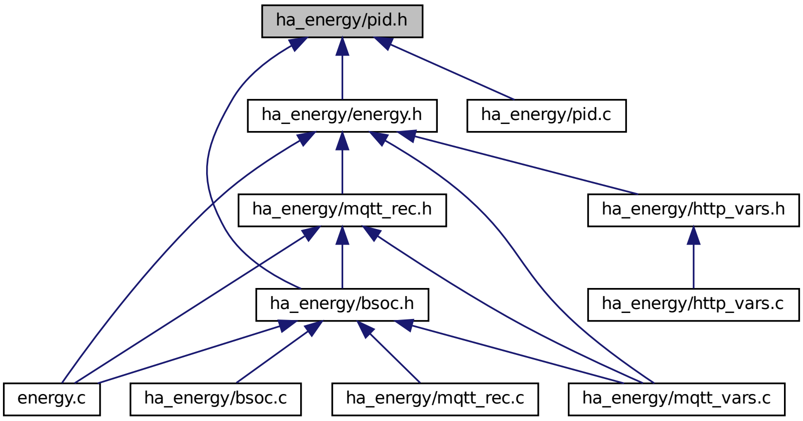
4.31.1.2 UpdatePI()

```
double UpdatePI (
              volatile struct SPid *const pid,
              double const error )
```

```
                                                                     {
   double pTerm, iTerm;

   pTerm = pid->pGain * error; // calculate the proportional term
    // calculate the integral state with appropriate limiting
    pid->iState += error;

    if (pid->iState > pid->iMax) {
        pid->iState = pid->iMax;
    } else if (pid->iState < pid->iMin) {
        pid->iState = pid->iMin;
    } else {

    }

    iTerm = (pid->iGain * pid->iState); // calculate the integral term

    if ((pTerm + iTerm) > pid->iMax) {
        iTerm = 0.0f;
        pTerm = pid->iMax;
    }
    return pTerm + iTerm;
}
```

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 )
{
    pid->dState = 0.0; // not used but cleared
    pid->iState = 0.0;
}
```

4.32.1.2 UpdatePI()

```
double UpdatePI (
              volatile struct SPid * const pid,
              const double error )
```

```
                                                                      {
    double pTerm, iTerm;

    pTerm = pid->pGain * error; // calculate the proportional term
    // calculate the integral state with appropriate limiting
    pid->iState += error;

    if (pid->iState > pid->iMax) {
        pid->iState = pid->iMax;
    } else if (pid->iState < pid->iMin) {
        pid->iState = pid->iMin;
    } else {

    }

    iTerm = (pid->iGain * pid->iState); // calculate the integral term

    if ((pTerm + iTerm) > pid->iMax) {
        iTerm = 0.0f;
        pTerm = pid->iMax;
    }
    return pTerm + iTerm;
}
```

4.33 pid.h

Go to the documentation of this file.

```
/*
 * File:      pid.h
 * Author:   root
 *
 * Created on March 6, 2024, 7:03 AM
 */

#ifndef PID_H
#define PID_H

#ifdef __cplusplus
extern "C" {
#endif

    struct SPid {
        double dState; // Last position input
        double iState; // Integrator state
        double iMax, iMin; // Maximum and minimum allowable integrator state
        double iGain, // integral gain
        pGain, // proportional gain
        dGain; // derivative gain
    };

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


#ifdef __cplusplus
}
#endif

#endif /* PID_H */

```

Index