

CDP Coding Challenge: Packet Converter

Generated by Doxygen 1.9.4

1 CDP Coding Challenge: Packet Parser	1
2 Todo List	3
3 Data Structure Index	5
3.1 Data Structures	5
4 File Index	7
4.1 File List	7
5 Data Structure Documentation	9
5.1 batt_packet Struct Reference	9
5.1.1 Detailed Description	9
5.1.2 Field Documentation	9
5.1.2.1 batt_status	9
5.1.2.2 err_check	10
5.1.2.3 time_stamp	10
5.2 pwr_packet Struct Reference	10
5.2.1 Detailed Description	10
5.2.2 Field Documentation	10
5.2.2.1 err_check	11
5.2.2.2 milliamps	11
5.2.2.3 milliwatts	11
5.2.2.4 time_stamp	11
5.2.2.5 volts	11
6 File Documentation	13
6.1 README.md File Reference	13
6.2 src/calculations.c File Reference	13
6.3 calculations.c	13
6.4 src/calculations.h File Reference	14
6.4.1 Detailed Description	14
6.4.2 Function Documentation	14
6.4.2.1 calc_power()	14
6.4.2.2 mod_of_array()	15
6.5 calculations.h	15
6.6 src/main.c File Reference	15
6.6.1 Function Documentation	16
6.6.1.1 get_pack_from_file()	16
6.6.1.2 main()	16
6.6.1.3 parse_arguments()	17
6.7 main.c	17
6.8 src/packet_constants.h File Reference	19
6.8.1 Detailed Description	20

6.8.2 Macro Definition Documentation	20
6.8.2.1 BATT_PACKET_TYPE_BYTE	20
6.8.2.2 BATT_START_OF_BATT_STATUS_LOC	20
6.8.2.3 BATT_START_OF_ERR_CHECK_LOC	20
6.8.2.4 BATT_STAT_NUM_OF_BYTES	20
6.8.2.5 ERR_CHECK_NUM_OF_BYTES	20
6.8.2.6 MAX_PACKET_SIZE	21
6.8.2.7 MILLIAMP_NUM_OF_BYTES	21
6.8.2.8 PACKET_TYPE_NUM_OF_BYTES	21
6.8.2.9 PWR_PACKET_TYPE_BYTE	21
6.8.2.10 PWR_START_OF_ERR_CHECK_LOC	21
6.8.2.11 PWR_START_OF_MILLIAMP_LOC	21
6.8.2.12 PWR_START_OF_VOLTS_LOC	22
6.8.2.13 SIZE_OF_BATT_PACK	22
6.8.2.14 SIZE_OF_PWR_PACK	22
6.8.2.15 START_OF_PT_LOC	22
6.8.2.16 START_OF_TS_LOC	22
6.8.2.17 TIMESTAMP_NUM_OF_BYTES	23
6.8.2.18 VOLTS_NUM_OF_BYTES	23
6.8.3 Typedef Documentation	23
6.8.3.1 batt_packet_t	23
6.8.3.2 pwr_packet_t	23
6.8.4 Enumeration Type Documentation	23
6.8.4.1 packet_type_t	23
6.9 packet_constants.h	24
6.10 src/packet_parser.c File Reference	25
6.10.1 Detailed Description	25
6.10.2 Function Documentation	25
6.10.2.1 check_for_pack_error()	25
6.10.2.2 convert_array_to_uint32()	26
6.10.2.3 determine_packet_type()	26
6.10.2.4 process_batt_packet()	27
6.10.2.5 process_pwr_packet()	27
6.11 packet_parser.c	27
6.12 src/packet_parser.h File Reference	29
6.12.1 Detailed Description	30
6.12.2 Function Documentation	30
6.12.2.1 determine_packet_type()	30
6.12.2.2 process_batt_packet()	30
6.12.2.3 process_pwr_packet()	31
6.13 packet_parser.h	31
6.14 src/state_handler.c File Reference	32

6.14.1 Detailed Description	32
6.14.2 Macro Definition Documentation	33
6.14.2.1 STATE_0_UPPER_BOUNDS	33
6.14.2.2 STATE_1_LOWER_BOUNDS	33
6.14.2.3 STATE_1_UPPER_BOUNDS	33
6.14.2.4 STATE_2_LOWER_BOUNDS	33
6.14.2.5 STATE_2_UPPER_BOUNDS	33
6.14.2.6 STATE_3_LOWER_BOUNDS	34
6.14.2.7 STATE_3_UPPER_BOUNDS	34
6.14.3 Enumeration Type Documentation	34
6.14.3.1 states_t	34
6.14.4 Function Documentation	34
6.14.4.1 calc_time_from_start_ms_to_sec()	34
6.14.4.2 determine_state()	35
6.14.4.3 process_state_and_transitions()	35
6.14.4.4 set_initial_timestamp()	36
6.14.4.5 time_check()	36
6.14.5 Variable Documentation	36
6.14.5.1 batt_states	36
6.14.5.2 initial_ts_ms	37
6.15 state_handler.c	37
6.16 src/state_handler.h File Reference	39
6.16.1 Detailed Description	39
6.16.2 Function Documentation	39
6.16.2.1 process_state_and_transitions()	39
6.17 state_handler.h	40
Index	41

Chapter 1

CDP Coding Challenge: Packet Parser

Coding Challenge for Cambridge Design Partnership where this program converts two different packets to a text output.

Toolchains Used

- GCC version 11.2.0
- CMake version 3.18.4 (Can use as low as 3.4 to compile though)
- Doxygen version 1.9.4
- cpputest version 4.0

Building the Application

This project uses CMake to autogenerate the needed compiler configuration files to build the application. This project was built and tested with GCC so CMake generates a GNU makefile but CMake can autogenerate build files for another compilers just as easily if so desired. To use other compilers or build systems, you will need to run CMake manual.

Building using the Bash Script

To easily build this program, open up a bash terminal in the project folder and simply run the following script:

```
./make_program.sh
```

This script will run cmake and make then outputs the executable to `${project_folder}/build/bin`

Building "by hand"

These are the steps to run CMake and build the program without the script. These steps will assume a bash terminal is used, but can be adapted to your own terminal.

1. Open a bash terminal in the project folder
2. Make a folder called "build" and enter that folder

```
mkdir ./build  
cd ./build
```

3. Run CMake while pointing to the project directory

```
cmake ../
```

4. This should generate a makefile in the build folder, so next run make

```
make
```

5. If successful, this should generate an executable located `${project_folder}/build/bin`

Running the application

The program needs an input bin file containing the data to be parsed. It should be passed as the only argument to the program on the command line. In a bash terminal, it should look something like (assuming run from in the bin folder):

```
./pwr_and_batt_packet_converter /path/to/input/file
```

There is a test input file located at `${project_folder}/test_input_file/CodingTest.bin` for demonstration purposes.

Unit Tests

See the [README.md](#) file in the unit test folder for information on how to build and run the unit tests. If `cputest` has already been installed or built in the submodule, then the script `make_and_run_unit_tests.sh` can be run from the project folder in a bash terminal. This script will build the unit tests and then runs them.

Doxygen

The Doxyfile provided generates both HTML and Latex outputs. It will output all files to `${project_folder}/doc/doxy_output` and can be generated with running Doxygen on the command line like so:

```
doxygen ./Doxyfile
```

The reference manual pdf, located in the doc folder was generated from the Latex output of the doxyfile.

Chapter 2

Todo List

Global `convert_array_to_uint32` (`uint8_t` const *const `p_array`, `const size_t` `length`)

assert that the length is ≤ 4 bytes

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

Potentially come back and rethink if we need any reason to error out of this loop

Global `time_check` (`uint32_t` `prev_ts_ms`, `uint32_t` `current_ts_ms`)

There is probably a better way to handle an `uint32_t` overflow or underflow situation.

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

batt_packet	Structure for holding the data related to the battery packet	9
pwr_packet	Structure for holding the data related to the power packet	10

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

src/ calculations.c	
Module for holding the functions used for calculating power	13
src/ calculations.h	
Module for holding the functions used for calculating power	14
src/ main.c	15
src/ packet_constants.h	
Header for information about packets like type and constants	19
src/ packet_parser.c	
Module for parsing the incoming packets	25
src/ packet_parser.h	
Module for parsing the incoming packets	29
src/ state_handler.c	
Module for handling any state related functions	32
src/ state_handler.h	
Module for handling any state related functions	39

Chapter 5

Data Structure Documentation

5.1 batt_packet Struct Reference

Structure for holding the data related to the battery packet.

```
#include <packet_constants.h>
```

Data Fields

- `uint32_t` [time_stamp](#)
Timestamp of the packet recieved in milliseconds.
- `uint8_t` [batt_status](#)
Status of the battery, 0-3 for VLOW,LOW,MED,HIGH.
- `uint8_t` [err_check](#)
Status of the battery, 0-3 for VLOW,LOW,MED,HIGH.

5.1.1 Detailed Description

Structure for holding the data related to the battery packet.

Definition at line 58 of file [packet_constants.h](#).

5.1.2 Field Documentation

5.1.2.1 batt_status

```
uint8_t batt_packet::batt_status
```

Timestamp of the packet recieved in milliseconds.

Definition at line 61 of file [packet_constants.h](#).

5.1.2.2 err_check

```
uint8_t batt_packet::err_check
```

Status of the battery, 0-3 for VLOW,LOW,MED,HIGH.

Definition at line 62 of file [packet_constants.h](#).

5.1.2.3 time_stamp

```
uint32_t batt_packet::time_stamp
```

Definition at line 60 of file [packet_constants.h](#).

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

- [src/packet_constants.h](#)

5.2 pwr_packet Struct Reference

Structure for holding the data related to the power packet.

```
#include <packet_constants.h>
```

Data Fields

- [uint32_t time_stamp](#)
Timestamp of the packet recieved in milliseconds.
- [uint32_t volts](#)
Volts of the power packet recieved.
- [uint8_t err_check](#)
Milliamps of the power packet recieved.
- [uint64_t milliamps](#)
Milliamps of the power packet recieved.
- [uint64_t milliwatts](#)
Error check value of the packet.

5.2.1 Detailed Description

Structure for holding the data related to the power packet.

Definition at line 48 of file [packet_constants.h](#).

5.2.2 Field Documentation

5.2.2.1 err_check

```
uint8_t pwr_packet::err_check
```

Milliamps of the power packet recieved.

Definition at line 53 of file [packet_constants.h](#).

5.2.2.2 milliamps

```
uint64_t pwr_packet::milliamps
```

Volts of the power packet recieved.

Definition at line 52 of file [packet_constants.h](#).

5.2.2.3 milliwatts

```
uint64_t pwr_packet::milliwatts
```

Error check value of the packet.

Definition at line 54 of file [packet_constants.h](#).

5.2.2.4 time_stamp

```
uint32_t pwr_packet::time_stamp
```

Definition at line 50 of file [packet_constants.h](#).

5.2.2.5 volts

```
uint32_t pwr_packet::volts
```

Timestamp of the packet recieved in milliseconds.

Definition at line 51 of file [packet_constants.h](#).

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

- [src/packet_constants.h](#)

Chapter 6

File Documentation

6.1 README.md File Reference

6.2 src/calculations.c File Reference

Module for holding the functions used for calculating power.

```
#include "calculations.h"
```

Include dependency graph for calculations.c:

6.3 calculations.c

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

```
00001 /**
00002  * @file calculations.c
00003  *
00004  * @brief Module for holding the functions used for calculating power.
00005  */
00006 #include "calculations.h"
00007
00008 /**
00009  * @brief Function for calculating the power from the voltage and amperage given
00010  *
00011  * @param pwr_numbers Struct holding the power information
00012  * @return int Success = 0, failure = -1
00013  */
00014 int calc_power(pwr_packet_t * const pwr_numbers)
00015 {
00016     int ret_status = -1;
00017     if (NULL != pwr_numbers)
00018     {
00019         pwr_numbers->milliwatts = (uint64_t)pwr_numbers->volts * pwr_numbers->milliamps;
00020         ret_status = 0;
00021     }
00022
00023     return ret_status;
00024 }
00025
00026 /**
00027  * @brief Function for taking the modulus of the data in an array
00028  *
00029  * @param p_array Array to be used
00030  * @param length Length of that array
00031  * @return uint32_t Modulus of the array
00032  */
00033 uint32_t mod_of_array(uint8_t const * const p_array, size_t length)
00034 {
00035     uint32_t ret_mod = 0;
00036     for (size_t i = 0; length > i; ++i)
00037     {
00038         ret_mod = (ret_mod + p_array[i]);
00039         ret_mod %= 256u;
00040     }
00041     return ret_mod;
00042 }
```

6.4 src/calculations.h File Reference

Module for holding the functions used for calculating power.

```
#include "packet_constants.h"
#include <stdint.h>
#include <stddef.h>
```

Include dependency graph for calculations.h: This graph shows which files directly or indirectly include this file:

Functions

- int [calc_power](#) ([pwr_packet_t](#) *const pwr_numbers)
Function for calculating the power from the voltage and amperage given.
- uint32_t [mod_of_array](#) (uint8_t const *const p_array, size_t length)
Function for taking the modulus of the data in an array.

6.4.1 Detailed Description

Module for holding the functions used for calculating power.

Definition in file [calculations.h](#).

6.4.2 Function Documentation

6.4.2.1 calc_power()

```
int calc_power (
    pwr\_packet\_t *const pwr_numbers )
```

Function for calculating the power from the voltage and amperage given.

Parameters

pwr_numbers	Struct holding the power information
-----------------------------	--------------------------------------

Returns

int Success = 0, failure = -1

Definition at line 14 of file [calculations.c](#).

6.4.2.2 mod_of_array()

```
uint32_t mod_of_array (
    uint8_t const *const p_array,
    size_t length )
```

Function for taking the modulus of the data in an array.

Parameters

<i>p_array</i>	Array to be used
<i>length</i>	Length of that array

Returns

uint32_t Modulus of the array

Definition at line 33 of file [calculations.c](#).

6.5 calculations.h

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

```
00001 /**
00002  * @file calculations.h
00003  *
00004  * @brief Module for holding the functions used for calculating power.
00005  */
00006
00007 #ifndef CALCULATIONS_H
00008 #define CALCULATIONS_H
00009
00010 #include "packet_constants.h"
00011
00012 #include <stdint.h>
00013 #include <stddef.h>
00014
00015 #ifdef __cplusplus
00016 extern "C" {
00017 #endif
00018
00019 int calc_power(pwr_packet_t * const pwr_numbers);
00020 uint32_t mod_of_array(uint8_t const * const p_array, size_t length);
00021
00022 #ifdef __cplusplus
00023 }
00024 #endif
00025
00026 #endif // CALCULATIONS_H
```

6.6 src/main.c File Reference

```
#include "calculations.h"
#include "packet_constants.h"
#include "packet_parser.h"
#include "state_handler.h"
#include <stdint.h>
#include <stdio.h>
#include <stddef.h>
#include <errno.h>
```

Include dependency graph for main.c:

Functions

- int [parse_arguments](#) (const int argc, char *argv[], FILE **p_file)
Function for parsing the inputs to the program.
- void [get_pack_from_file](#) (FILE **p_file, uint8_t *out_buff, const size_t size)
Get the pack from file object.
- int [main](#) (int argc, char *argv[])
Main function of the program.

6.6.1 Function Documentation

6.6.1.1 [get_pack_from_file\(\)](#)

```
void get_pack_from_file (
    FILE ** p_file,
    uint8_t * out_buff,
    const size_t size )
```

Get the pack from file object.

Parameters

<i>p_file</i>	Pointer to the file being processed
<i>out_buff</i>	Outputting the data obtained from the file
<i>size</i>	size of the buffer to output data in.

Definition at line [112](#) of file [main.c](#).

6.6.1.2 [main\(\)](#)

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

Main function of the program.

Parameters

<i>argc</i>	Number of arguments
<i>argv</i>	Pointer to the list of inputed arguments

Returns

int Status of program

Todo Potentially come back and rethink if we need any reason to error out of this loop

Definition at line 21 of file [main.c](#).

6.6.1.3 parse_arguments()

```
int parse_arguments (
    const int argc,
    char * argv[],
    FILE ** p_file )
```

Function for parsing the inputs to the program.

Parameters

<i>argc</i>	Number of arguments being entered
<i>argv</i>	Array of the arguments
<i>p_file</i>	Output of the file being expected to be passed. Will be NULL if invalid.

Returns

int Return status, 0 for success, error code for failure

Definition at line 80 of file [main.c](#).

6.7 main.c

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

```
00001 #include "calculations.h"
00002 #include "packet_constants.h"
00003 #include "packet_parser.h"
00004 #include "state_handler.h"
00005
00006 #include <stdint.h>
00007 #include <stdio.h>
00008 #include <stddef.h>
00009 #include <errno.h>
00010
00011 int parse_arguments(const int argc, char * argv[], FILE ** p_file);
00012 void get_pack_from_file(FILE ** p_file, uint8_t * out_buff, const size_t size);
00013
00014 /**
00015  * @brief Main function of the program
00016  *
00017  * @param argc Number of arguments
00018  * @param argv Pointer to the list of inputted arguments
00019  * @return int Status of program
00020  */
00021 int main(int argc, char * argv[])
00022 {
00023     int ret_status = 0;
00024     FILE * p_bin_file = NULL;
00025
00026     ret_status = parse_arguments(argc, argv, &p_bin_file);
00027
00028     uint8_t packet_buffer[MAX_PACKET_SIZE] = {0};
00029     pwr_packet_t pwr_pack = {0};
00030     batt_packet_t batt_pack = {0};
```

```

00031     if (0 == ret_status)
00032     {
00033         ///

```



```
00118 }
```

6.8 src/packet_constants.h File Reference

Header for information about packets like type and constants.

```
#include <stdint.h>
```

Include dependency graph for packet_constants.h: This graph shows which files directly or indirectly include this file:

Data Structures

- struct [pwr_packet](#)
Structure for holding the data related to the power packet.
- struct [batt_packet](#)
Structure for holding the data related to the battery packet.

Macros

- #define [PACKET_TYPE_NUM_OF_BYTES](#) 1u
- #define [TIMESTAMP_NUM_OF_BYTES](#) 4u
- #define [VOLTS_NUM_OF_BYTES](#) 4u
- #define [MILLIAMP_NUM_OF_BYTES](#) 8u
- #define [ERR_CHECK_NUM_OF_BYTES](#) 1u
- #define [BATT_STAT_NUM_OF_BYTES](#) 1u
- #define [SIZE_OF_PWR_PACK](#)
- #define [SIZE_OF_BATT_PACK](#)
- #define [MAX_PACKET_SIZE](#)
- #define [PWR_PACKET_TYPE_BYTE](#) 0x00u
- #define [BATT_PACKET_TYPE_BYTE](#) 0x01u
- #define [START_OF_PT_LOC](#) (0u)
- #define [START_OF_TS_LOC](#) (START_OF_PT_LOC + [PACKET_TYPE_NUM_OF_BYTES](#))
- #define [PWR_START_OF_VOLTS_LOC](#) (START_OF_TS_LOC + [TIMESTAMP_NUM_OF_BYTES](#))
- #define [PWR_START_OF_MILLIAMP_LOC](#) (PWR_START_OF_VOLTS_LOC + [VOLTS_NUM_OF_BYTES](#))
- #define [PWR_START_OF_ERR_CHECK_LOC](#) (PWR_START_OF_MILLIAMP_LOC + [MILLIAMP_NUM_OF_BYTES](#))
- #define [BATT_START_OF_BATT_STATUS_LOC](#) (START_OF_TS_LOC + [TIMESTAMP_NUM_OF_BYTES](#))
- #define [BATT_START_OF_ERR_CHECK_LOC](#) (BATT_START_OF_BATT_STATUS_LOC + [BATT_STAT_NUM_OF_BYTES](#))

Typedefs

- typedef struct [pwr_packet](#) [pwr_packet_t](#)
Structure for holding the data related to the power packet.
- typedef struct [batt_packet](#) [batt_packet_t](#)
Structure for holding the data related to the battery packet.

Enumerations

- enum [packet_type_t](#) { [power_pack](#) = 0 , [battery_pack](#) , [error_type](#) , [num_of_types](#) }
Typedef Enum to indicate which type of packet we are working with.

6.8.1 Detailed Description

Header for information about packets like type and constants.

Definition in file [packet_constants.h](#).

6.8.2 Macro Definition Documentation

6.8.2.1 BATT_PACKET_TYPE_BYTE

```
#define BATT_PACKET_TYPE_BYTE 0x01u
```

Definition at line 38 of file [packet_constants.h](#).

6.8.2.2 BATT_START_OF_BATT_STATUS_LOC

```
#define BATT_START_OF_BATT_STATUS_LOC (START_OF_TS_LOC + TIMESTAMP_NUM_OF_BYTES)
```

Definition at line 44 of file [packet_constants.h](#).

6.8.2.3 BATT_START_OF_ERR_CHECK_LOC

```
#define BATT_START_OF_ERR_CHECK_LOC (BATT_START_OF_BATT_STATUS_LOC + BATT_STAT_NUM_OF_BYTES)
```

Definition at line 45 of file [packet_constants.h](#).

6.8.2.4 BATT_STAT_NUM_OF_BYTES

```
#define BATT_STAT_NUM_OF_BYTES 1u
```

Definition at line 21 of file [packet_constants.h](#).

6.8.2.5 ERR_CHECK_NUM_OF_BYTES

```
#define ERR_CHECK_NUM_OF_BYTES 1u
```

Definition at line 20 of file [packet_constants.h](#).

6.8.2.6 MAX_PACKET_SIZE

```
#define MAX_PACKET_SIZE
```

Value:

```
((SIZE_OF_PWR_PACK > SIZE_OF_BATT_PACK) ? \  
SIZE_OF_PWR_PACK : SIZE_OF_BATT_PACK)
```

Definition at line 34 of file [packet_constants.h](#).

6.8.2.7 MILLIAMP_NUM_OF_BYTES

```
#define MILLIAMP_NUM_OF_BYTES 8u
```

Definition at line 19 of file [packet_constants.h](#).

6.8.2.8 PACKET_TYPE_NUM_OF_BYTES

```
#define PACKET_TYPE_NUM_OF_BYTES 1u
```

Definition at line 16 of file [packet_constants.h](#).

6.8.2.9 PWR_PACKET_TYPE_BYTE

```
#define PWR_PACKET_TYPE_BYTE 0x00u
```

Definition at line 37 of file [packet_constants.h](#).

6.8.2.10 PWR_START_OF_ERR_CHECK_LOC

```
#define PWR_START_OF_ERR_CHECK_LOC (PWR_START_OF_MILLIAMP_LOC + MILLIAMP_NUM_OF_BYTES)
```

Definition at line 43 of file [packet_constants.h](#).

6.8.2.11 PWR_START_OF_MILLIAMP_LOC

```
#define PWR_START_OF_MILLIAMP_LOC (PWR_START_OF_VOLTS_LOC + VOLTS_NUM_OF_BYTES)
```

Definition at line 42 of file [packet_constants.h](#).

6.8.2.12 PWR_START_OF_VOLTS_LOC

```
#define PWR_START_OF_VOLTS_LOC (START_OF_TS_LOC + TIMESTAMP_NUM_OF_BYTES)
```

Definition at line 41 of file [packet_constants.h](#).

6.8.2.13 SIZE_OF_BATT_PACK

```
#define SIZE_OF_BATT_PACK
```

Value:

```
(PACKET_TYPE_NUM_OF_BYTES \
+ TIMESTAMP_NUM_OF_BYTES \
+ BATT_STAT_NUM_OF_BYTES \
+ ERR_CHECK_NUM_OF_BYTES)
```

Definition at line 29 of file [packet_constants.h](#).

6.8.2.14 SIZE_OF_PWR_PACK

```
#define SIZE_OF_PWR_PACK
```

Value:

```
( PACKET_TYPE_NUM_OF_BYTES \
+ TIMESTAMP_NUM_OF_BYTES \
+ VOLTS_NUM_OF_BYTES \
+ MILLIAMP_NUM_OF_BYTES \
+ ERR_CHECK_NUM_OF_BYTES)
```

Definition at line 23 of file [packet_constants.h](#).

6.8.2.15 START_OF_PT_LOC

```
#define START_OF_PT_LOC (0u)
```

Definition at line 39 of file [packet_constants.h](#).

6.8.2.16 START_OF_TS_LOC

```
#define START_OF_TS_LOC (START_OF_PT_LOC + PACKET_TYPE_NUM_OF_BYTES)
```

Definition at line 40 of file [packet_constants.h](#).

6.8.2.17 TIMESTAMP_NUM_OF_BYTES

```
#define TIMESTAMP_NUM_OF_BYTES 4u
```

Definition at line 17 of file [packet_constants.h](#).

6.8.2.18 VOLTS_NUM_OF_BYTES

```
#define VOLTS_NUM_OF_BYTES 4u
```

Definition at line 18 of file [packet_constants.h](#).

6.8.3 Typedef Documentation

6.8.3.1 batt_packet_t

```
typedef struct batt_packet batt_packet_t
```

Structure for holding the data related to the battery packet.

6.8.3.2 pwr_packet_t

```
typedef struct pwr_packet pwr_packet_t
```

Structure for holding the data related to the power packet.

6.8.4 Enumeration Type Documentation

6.8.4.1 packet_type_t

```
enum packet_type_t
```

Typedef Enum to indicate which type of packet we are working with.

Enumerator

power_pack	
battery_pack	
error_type	
num_of_types	

Definition at line 66 of file [packet_constants.h](#).

6.9 packet_constants.h

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

```

00001 /**
00002  * @file packet_constants.h
00003  *
00004  * @brief Header for information about packets like type and constants
00005  */
00006
00007 #ifndef PACKET_CONSTANTS_H
00008 #define PACKET_CONSTANTS_H
00009
00010 #include <stdint.h>
00011
00012 #ifdef __cplusplus
00013 extern "C" {
00014 #endif
00015
00016 #define PACKET_TYPE_NUM_OF_BYTES 1u
00017 #define TIMESTAMP_NUM_OF_BYTES 4u
00018 #define VOLTS_NUM_OF_BYTES 4u
00019 #define MILLIAMP_NUM_OF_BYTES 8u
00020 #define ERR_CHECK_NUM_OF_BYTES 1u
00021 #define BATT_STAT_NUM_OF_BYTES 1u
00022
00023 #define SIZE_OF_PWR_PACK ( PACKET_TYPE_NUM_OF_BYTES \
00024                          + TIMESTAMP_NUM_OF_BYTES \
00025                          + VOLTS_NUM_OF_BYTES \
00026                          + MILLIAMP_NUM_OF_BYTES \
00027                          + ERR_CHECK_NUM_OF_BYTES)
00028
00029 #define SIZE_OF_BATT_PACK (PACKET_TYPE_NUM_OF_BYTES \
00030                           + TIMESTAMP_NUM_OF_BYTES \
00031                           + BATT_STAT_NUM_OF_BYTES \
00032                           + ERR_CHECK_NUM_OF_BYTES)
00033
00034 #define MAX_PACKET_SIZE ((SIZE_OF_PWR_PACK > SIZE_OF_BATT_PACK) ? \
00035                          SIZE_OF_PWR_PACK : SIZE_OF_BATT_PACK)
00036
00037 #define PWR_PACKET_TYPE_BYTE 0x00u
00038 #define BATT_PACKET_TYPE_BYTE 0x01u
00039 #define START_OF_PT_LOC (0u)
00040 #define START_OF_TS_LOC (START_OF_PT_LOC + PACKET_TYPE_NUM_OF_BYTES)
00041 #define PWR_START_OF_VOLTS_LOC (START_OF_TS_LOC + TIMESTAMP_NUM_OF_BYTES)
00042 #define PWR_START_OF_MILLIAMP_LOC (PWR_START_OF_VOLTS_LOC + VOLTS_NUM_OF_BYTES)
00043 #define PWR_START_OF_ERR_CHECK_LOC (PWR_START_OF_MILLIAMP_LOC + MILLIAMP_NUM_OF_BYTES)
00044 #define BATT_START_OF_BATT_STATUS_LOC (START_OF_TS_LOC + TIMESTAMP_NUM_OF_BYTES)
00045 #define BATT_START_OF_ERR_CHECK_LOC (BATT_START_OF_BATT_STATUS_LOC + BATT_STAT_NUM_OF_BYTES)
00046
00047 /// Structure for holding the data related to the power packet
00048 typedef struct pwr_packet
00049 {
00050     uint32_t time_stamp; /// Timestamp of the packet recieved in milliseconds
00051     uint32_t volts; /// Volts of the power packet recieved
00052     uint64_t milliamps; /// Milliamps of the power packet recieved
00053     uint8_t err_check; /// Error check value of the packet
00054     uint64_t milliwatts; /// The calculated power from the volts * milliamps
00055 } pwr_packet_t;
00056
00057 /// Structure for holding the data related to the battery packet
00058 typedef struct batt_packet
00059 {
00060     uint32_t time_stamp; /// Timestamp of the packet recieved in milliseconds
00061     uint8_t batt_status; /// Status of the battery, 0-3 for VLOW,LOW,MED,HIGH
00062     uint8_t err_check; /// Error check value of the packet
00063 } batt_packet_t;
00064
00065 /// Typedef Enum to indicate which type of packet we are working with
00066 typedef enum
00067 {
00068     power_pack = 0,
00069     battery_pack,
00070     error_type,
00071     num_of_types
00072 } packet_type_t;
00073
00074 #ifdef __cplusplus
00075 }
00076 #endif
00077
00078 #endif // PACKET_CONSTANTS_H

```

6.10 src/packet_parser.c File Reference

Module for parsing the incoming packets.

```
#include "packet_parser.h"
#include "calculations.h"
#include <stdbool.h>
#include <stdio.h>
Include dependency graph for packet_parser.c:
```

Functions

- `uint32_t convert_array_to_uint32` (`uint8_t const *const p_array`, `const size_t length`)
Function for converting an uint8_t array to an uint32_t.
- `int check_for_pack_error` (`const packet_type_t pack_type`, `uint8_t const *const p_packet`)
Helper function to check the error byte for integraty.
- `packet_type_t determine_packet_type` (`const uint8_t first_byte_of_pack`)
Function for determining the type of the incoming packet.
- `int process_pwr_packet` (`uint8_t const *const p_packet_buf`, `pwr_packet_t *const p_out_pack`)
Function for creating a power packet from an input buffer.
- `int process_batt_packet` (`uint8_t const *const p_packet_buf`, `batt_packet_t *const p_out_pack`)
Function for creating a battery status packet from an input buffer.

6.10.1 Detailed Description

Module for parsing the incoming packets.

Definition in file `packet_parser.c`.

6.10.2 Function Documentation

6.10.2.1 check_for_pack_error()

```
int check_for_pack_error (
    const packet_type_t pack_type,
    uint8_t const *const p_packet )
```

Helper function to check the error byte for integraty.

Parameters

<i>pack_type</i>	Type of packet being checked
<i>p_packet</i>	pointer to the packet buffer

Returns

int Success = 0

Definition at line 131 of file [packet_parser.c](#).

6.10.2.2 convert_array_to_uint32()

```
uint32_t convert_array_to_uint32 (
    uint8_t const *const p_array,
    const size_t length )
```

Function for converting an uint8_t array to an uint32_t.

Todo assert that the length is <=4 bytes

Parameters

<i>p_array</i>	Array to be converted
<i>length</i>	length of the array in bytes

Returns

uint32_t

Definition at line 111 of file [packet_parser.c](#).

6.10.2.3 determine_packet_type()

```
packet_type_t determine_packet_type (
    const uint8_t first_byte_of_pack )
```

Function for determining the type of the incoming packet.

Parameters

<i>first_byte_of_pack</i>	1st byte of the packet
---------------------------	------------------------

Returns

packet_type_t Type of packet, will return the Enum error_type if it's an invalid packet type.

Definition at line 24 of file [packet_parser.c](#).

6.10.2.4 process_batt_packet()

```
int process_batt_packet (
    uint8_t const *const p_packet_buf,
    batt_packet_t *const p_out_pack )
```

Function for creating a battery status packet from an input buffer.

Parameters

<i>packet_buf</i>	Buffer to be converted
-------------------	------------------------

Returns

batt_packet_t Struct with the battery information in it.

Definition at line 82 of file [packet_parser.c](#).

6.10.2.5 process_pwr_packet()

```
int process_pwr_packet (
    uint8_t const *const p_packet_buf,
    pwr_packet_t *const p_out_pack )
```

Function for creating a power packet from an input buffer.

Parameters

<i>packet_buf</i>	Buffer to be converted
-------------------	------------------------

Returns

pwr_packet_t Struct with the power information in it

Definition at line 49 of file [packet_parser.c](#).

6.11 packet_parser.c

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

```
00001 /**
00002  * @file packet_parser.c
00003  *
00004  * @brief Module for parsing the incoming packets
00005  */
00006
00007 #include "packet_parser.h"
00008 #include "calculations.h"
00009 #include <stdbool.h>
00010 #include <stdio.h>
00011
```

```

00012 uint32_t convert_array_to_uint32(uint8_t const * const p_array,
00013                                 const size_t length);
00014 int check_for_pack_error(const packet_type_t pack_type,
00015                         uint8_t const * const p_packet);
00016
00017 /**
00018  * @brief Function for determining the type of the incoming packet
00019  *
00020  * @param first_byte_of_pack 1st byte of the packet
00021  * @return packet_type_t Type of packet, will return the Enum error_type if it's
00022  *         an invalid packet type.
00023  */
00024 packet_type_t determine_packet_type(const uint8_t first_byte_of_pack)
00025 {
00026     packet_type_t ret_type = error_type;
00027     if (PWR_PACKET_TYPE_BYTE == first_byte_of_pack)
00028     {
00029         ret_type = power_pack;
00030     }
00031     else if (BATT_PACKET_TYPE_BYTE == first_byte_of_pack)
00032     {
00033         ret_type = battery_pack;
00034     }
00035     else
00036     {
00037         // Error so do nothing
00038     }
00039     return ret_type;
00040 }
00041
00042 /**
00043  * @brief Function for creating a power packet from an input buffer
00044  *
00045  * @param packet_buf Buffer to be converted
00046  * @return pwr_packet_t Struct with the power information in it
00047  */
00048 int process_pwr_packet(uint8_t const * const p_packet_buf,
00049                       pwr_packet_t * const p_out_pack)
00050 {
00051     int ret_status = -1;
00052     if ((NULL != p_packet_buf) && (NULL != p_out_pack))
00053     {
00054         p_out_pack->time_stamp = convert_array_to_uint32(&p_packet_buf[START_OF_TS_LOC],
00055                                                         TIMESTAMP_NUM_OF_BYTES);
00056         p_out_pack->volts = convert_array_to_uint32(&p_packet_buf[PWR_START_OF_VOLTS_LOC],
00057                                                    VOLTS_NUM_OF_BYTES);
00058         p_out_pack->milliamps = convert_array_to_uint32(&p_packet_buf[PWR_START_OF_MILLIAMP_LOC],
00059                                                        MILLIAMP_NUM_OF_BYTES);
00060         p_out_pack->err_check = (uint16_t)p_packet_buf[PWR_START_OF_ERR_CHECK_LOC];
00061         ret_status = calc_power(p_out_pack);
00062         if (0 == ret_status)
00063         {
00064             ret_status = check_for_pack_error(power_pack, p_packet_buf);
00065         }
00066     }
00067     return ret_status;
00068 }
00069
00070 /**
00071  * @brief Function for creating a battery status packet from an input buffer
00072  *
00073  * @param packet_buf Buffer to be converted
00074  * @return batt_packet_t Struct with the battery information in it.
00075  */
00076 int process_batt_packet(uint8_t const * const p_packet_buf,
00077                        batt_packet_t * const p_out_pack)
00078 {
00079     int ret_status = -1;
00080     if ((NULL != p_packet_buf) && (NULL != p_out_pack))
00081     {
00082         p_out_pack->time_stamp = convert_array_to_uint32(&p_packet_buf[START_OF_TS_LOC],
00083                                                         TIMESTAMP_NUM_OF_BYTES);
00084         p_out_pack->batt_status = p_packet_buf[BATT_START_OF_BATT_STATUS_LOC];
00085         p_out_pack->err_check = (uint16_t)p_packet_buf[BATT_START_OF_ERR_CHECK_LOC];
00086         ret_status = check_for_pack_error(battery_pack, p_packet_buf);
00087     }
00088     return ret_status;
00089 }
00090
00091
00092
00093
00094
00095
00096
00097
00098

```

```

00099     return ret_status;
00100 }
00101
00102 /**
00103  * @brief Function for converting an uint8_t array to an uint32_t
00104  *
00105  * @todo assert that the length is <=4 bytes
00106  *
00107  * @param p_array Array to be converted
00108  * @param length length of the array in bytes
00109  * @return uint32_t
00110  */
00111 uint32_t convert_array_to_uint32(uint8_t const * const p_array,
00112                                 const size_t length)
00113 {
00114     uint32_t ret_val = 0;
00115     for (size_t i = 0u; length > i; ++i)
00116     {
00117         ret_val <= 8u;
00118         ret_val += p_array[i];
00119     }
00120     return ret_val;
00121 }
00122
00123 /**
00124  * @brief Helper function to check the error byte for integraty
00125  *
00126  *
00127  * @param pack_type Type of packet being checked
00128  * @param p_packet pointer to the packet buffer
00129  * @return int Success = 0
00130  */
00131 int check_for_pack_error(const packet_type_t pack_type,
00132                          uint8_t const * const p_packet)
00133 {
00134     int ret_status = 0;
00135     uint32_t mod_of_pack = 0;
00136     switch (pack_type)
00137     {
00138     case power_pack:
00139         mod_of_pack = mod_of_array(p_packet, (SIZE_OF_PWR_PACK - ERR_CHECK_NUM_OF_BYTES));
00140         if (mod_of_pack != p_packet[PWR_START_OF_ERR_CHECK_LOC])
00141         {
00142             ret_status = -1;
00143             printf("ERR: Packet Failed Error Check\n");
00144         }
00145         break;
00146     case battery_pack:
00147         mod_of_pack = mod_of_array(p_packet, (SIZE_OF_BATT_PACK - ERR_CHECK_NUM_OF_BYTES));
00148         if (mod_of_pack != p_packet[BATT_START_OF_ERR_CHECK_LOC])
00149         {
00150             ret_status = -1;
00151             printf("ERR: Packet Failed Error Check\n");
00152         }
00153         break;
00154     default:
00155         // Do nothing, but return an error.
00156         ret_status = -1;
00157         break;
00158     }
00159     return ret_status;
00160 }
00161 }

```

6.12 src/packet_parser.h File Reference

Module for parsing the incoming packets.

```

#include "packet_constants.h"
#include <stdint.h>
#include <stddef.h>

```

Include dependency graph for packet_parser.h: This graph shows which files directly or indirectly include this file:

Functions

- [packet_type_t determine_packet_type](#) (const uint8_t first_byte_of_pack)

Function for determining the type of the incoming packet.

- int [process_pwr_packet](#) (uint8_t const *const p_packet_buf, [pwr_packet_t](#) *const p_out_pack)

Function for creating a power packet from an input buffer.

- int [process_batt_packet](#) (uint8_t const *const p_packet_buf, [batt_packet_t](#) *const p_out_pack)

Function for creating a battery status packet from an input buffer.

6.12.1 Detailed Description

Module for parsing the incoming packets.

Definition in file [packet_parser.h](#).

6.12.2 Function Documentation

6.12.2.1 `determine_packet_type()`

```
packet_type_t determine_packet_type (
    const uint8_t first_byte_of_pack )
```

Function for determining the type of the incoming packet.

Parameters

<code>first_byte_of_pack</code>	1st byte of the packet
---------------------------------	------------------------

Returns

`packet_type_t` Type of packet, will return the Enum `error_type` if it's an invalid packet type.

Definition at line 24 of file [packet_parser.c](#).

6.12.2.2 `process_batt_packet()`

```
int process_batt_packet (
    uint8_t const *const p_packet_buf,
    batt_packet_t *const p_out_pack )
```

Function for creating a battery status packet from an input buffer.

Parameters

<code>packet_buf</code>	Buffer to be converted
-------------------------	------------------------

Returns

batt_packet_t Struct with the battery information in it.

Definition at line 82 of file [packet_parser.c](#).

6.12.2.3 process_pwr_packet()

```
int process_pwr_packet (
    uint8_t const *const p_packet_buf,
    pwr_packet_t *const p_out_pack )
```

Function for creating a power packet from an input buffer.

Parameters

<i>packet_buf</i>	Buffer to be converted
-------------------	------------------------

Returns

pwr_packet_t Struct with the power information in it

Definition at line 49 of file [packet_parser.c](#).

6.13 packet_parser.h

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

```
00001 /**
00002  * @file packet_parser.h
00003  *
00004  * @brief Module for parsing the incoming packets
00005  */
00006
00007 #ifndef PACKET_PARSER_H
00008 #define PACKET_PARSER_H
00009
00010 #include "packet_constants.h"
00011 #include <stdint.h>
00012 #include <stddef.h>
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 packet_type_t determine_packet_type(const uint8_t first_byte_of_pack);
00019 int process_pwr_packet(uint8_t const * const p_packet_buf,
00020                       pwr_packet_t * const p_out_pack);
00021 int process_batt_packet(uint8_t const * const p_packet_buf,
00022                        batt_packet_t * const p_out_pack);
00023
00024 #ifdef __cplusplus
00025 }
00026 #endif
00027
00028 #endif // PACKET_PARSER_H
```

6.14 src/state_handler.c File Reference

Module for handling any state related functions.

```
#include "state_handler.h"
#include <stdbool.h>
#include <stdio.h>
Include dependency graph for state_handler.c:
```

Macros

- `#define STATE_0_UPPER_BOUNDS 200u`
- `#define STATE_1_LOWER_BOUNDS 300u`
- `#define STATE_1_UPPER_BOUNDS 450u`
- `#define STATE_2_LOWER_BOUNDS 550u`
- `#define STATE_2_UPPER_BOUNDS 650u`
- `#define STATE_3_LOWER_BOUNDS 800u`
- `#define STATE_3_UPPER_BOUNDS 1200u`

Enumerations

- enum `states_t` {
`STATE_0 = 0` , `STATE_1` , `STATE_2` , `STATE_3` ,
`NUM_OF_STATES` }

Functions

- `uint32_t calc_time_from_start_ms_to_sec` (`uint32_t current_ts_ms`)
Function for calculating the time since the start of the program.
- `states_t determine_state` (`pwr_packet_t` const *const pwr)
Function for determining the state from power.
- void `set_initial_timestamp` (const `uint32_t ts_ms`)
Set the initial timestamp object.
- bool `time_check` (`uint32_t prev_ts_ms`, `uint32_t current_ts_ms`)
Function to check if 2 timestamps are greater than 10 ms apart.
- int `process_state_and_transitions` (`packet_type_t` pack_type, `pwr_packet_t` const *const pwr, `batt_packet_t` const *const batt)
Function for processing the states of either a power or battery state.

Variables

- const char * `batt_states` [4] = {"VLOW", "LOW", "MED", "HIGH"}
- static `uint32_t initial_ts_ms` = 0

6.14.1 Detailed Description

Module for handling any state related functions.

Definition in file `state_handler.c`.

6.14.2 Macro Definition Documentation

6.14.2.1 STATE_0_UPPER_BOUNDS

```
#define STATE_0_UPPER_BOUNDS 200u
```

Definition at line 10 of file [state_handler.c](#).

6.14.2.2 STATE_1_LOWER_BOUNDS

```
#define STATE_1_LOWER_BOUNDS 300u
```

Definition at line 11 of file [state_handler.c](#).

6.14.2.3 STATE_1_UPPER_BOUNDS

```
#define STATE_1_UPPER_BOUNDS 450u
```

Definition at line 12 of file [state_handler.c](#).

6.14.2.4 STATE_2_LOWER_BOUNDS

```
#define STATE_2_LOWER_BOUNDS 550u
```

Definition at line 13 of file [state_handler.c](#).

6.14.2.5 STATE_2_UPPER_BOUNDS

```
#define STATE_2_UPPER_BOUNDS 650u
```

Definition at line 14 of file [state_handler.c](#).

6.14.2.6 STATE_3_LOWER_BOUNDS

```
#define STATE_3_LOWER_BOUNDS 800u
```

Definition at line 15 of file [state_handler.c](#).

6.14.2.7 STATE_3_UPPER_BOUNDS

```
#define STATE_3_UPPER_BOUNDS 1200u
```

Definition at line 16 of file [state_handler.c](#).

6.14.3 Enumeration Type Documentation

6.14.3.1 states_t

```
enum states\_t
```

Enumerator

STATE_0	
STATE_1	Initial State for Power reading, 0 to 200 mW.
STATE_2	1st State for Power reading, 300 to 450 mW
STATE_3	2nd State for Power reading, 550 to 650 mW
NUM_OF_STATES	2nd State for Power reading, 800 to 1200 mW Number of total states

Definition at line 18 of file [state_handler.c](#).

6.14.4 Function Documentation

6.14.4.1 calc_time_from_start_ms_to_sec()

```
uint32_t calc_time_from_start_ms_to_sec (
    uint32_t current_ts_ms )
```

Function for calculating the time since the start of the program.

Parameters

<i>current_ts_ms</i>	Current timestamp in milliseconds
----------------------	-----------------------------------

Returns

uint32_t Time from start of program in seconds

Definition at line 109 of file [state_handler.c](#).

6.14.4.2 determine_state()

```
states_t determine_state (  
    pwr_packet_t const *const pwr )
```

Function for determining the state from power.

Parameters

<i>pwr</i>	Pointer to the struct that holds the power information
------------	--------------------------------------------------------

Returns

states_t Current state, returns NUM_OF_STATES if invalid

Definition at line 121 of file [state_handler.c](#).

6.14.4.3 process_state_and_transitions()

```
int process_state_and_transitions (  
    packet_type_t pack_type,  
    pwr_packet_t const *const pwr,  
    batt_packet_t const *const batt )
```

Function for processing the states of either a power or battery state.

Parameters

<i>pack_type</i>	Type of packet being processed
<i>pwr</i>	Pointer to the struct that holds the power information
<i>batt</i>	Pointer to the struct that holds the battery information

Returns

int 0 == Success

Definition at line 43 of file [state_handler.c](#).

6.14.4.4 `set_initial_timestamp()`

```
void set_initial_timestamp (
    const uint32_t ts_ms )
```

Set the initial timestamp object.

Parameters

<i>ts</i>	Initial timestamp to be saved in milliseconds
-----------	-----------------------------------------------

Definition at line 156 of file [state_handler.c](#).

6.14.4.5 `time_check()`

```
bool time_check (
    uint32_t prev_ts_ms,
    uint32_t current_ts_ms )
```

Function to check if 2 timestamps are greater than 10 ms apart.

Todo There is probably a better way to handle an uint32_t overflow or underflow situation.

Parameters

<i>prev_ts_ms</i>	Previous timestamp
<i>current_ts_ms</i>	Current timestamp

Returns

true If (current - prev) > 10ms
false If (current - prev) < 10ms

Definition at line 172 of file [state_handler.c](#).

6.14.5 Variable Documentation

6.14.5.1 `batt_states`

```
const char* batt_states[4] = {"VLOW", "LOW", "MED", "HIGH"}
```

Definition at line 27 of file [state_handler.c](#).

6.14.5.2 inital_ts_ms

```
uint32_t inital_ts_ms = 0 [static]
```

Definition at line 33 of file [state_handler.c](#).

6.15 state_handler.c

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

```
00001 /**
00002  * @file state_handler.c
00003  *
00004  * @brief Module for handling any state related functions.
00005  */
00006 #include "state_handler.h"
00007 #include <stdbool.h>
00008 #include <stdio.h>
00009
00010 #define STATE_0_UPPER_BOUNDS 200u
00011 #define STATE_1_LOWER_BOUNDS 300u
00012 #define STATE_1_UPPER_BOUNDS 450u
00013 #define STATE_2_LOWER_BOUNDS 550u
00014 #define STATE_2_UPPER_BOUNDS 650u
00015 #define STATE_3_LOWER_BOUNDS 800u
00016 #define STATE_3_UPPER_BOUNDS 1200u
00017
00018 typedef enum
00019 {
00020     STATE_0 = 0, /// Initial State for Power reading, 0 to 200 mW
00021     STATE_1, /// 1st State for Power reading, 300 to 450 mW
00022     STATE_2, /// 2nd State for Power reading, 550 to 650 mW
00023     STATE_3, /// 2nd State for Power reading, 800 to 1200 mW
00024     NUM_OF_STATES /// Number of total states
00025 } states_t;
00026
00027 const char * batt_states[4] = {"VLOW", "LOW", "MED", "HIGH"};
00028 uint32_t calc_time_from_start_ms_to_sec(uint32_t current_ts_ms);
00029 states_t determine_state(pwr_packet_t const * const pwr);
00030 void set_initial_timestamp(const uint32_t ts_ms);
00031 bool time_check(uint32_t prev_ts_ms, uint32_t current_ts_ms);
00032
00033 static uint32_t inital_ts_ms = 0;
00034
00035 /**
00036  * @brief Function for processing the states of either a power or battery state
00037  *
00038  * @param pack_type Type of packet being processed
00039  * @param pwr Pointer to the struct that holds the power information
00040  * @param batt Pointer to the struct that holds the battery information
00041  * @return int 0 == Success
00042  */
00043 int process_state_and_transitions(packet_type_t pack_type,
00044                                   pwr_packet_t const * const pwr,
00045                                   batt_packet_t const * const batt)
00046 {
00047     int ret_status = 0;
00048     static uint32_t prev_ts_ms = 0;
00049     static states_t prev_state = STATE_0;
00050     static bool init_inital_ts = true;
00051     if ((NULL == pwr) || (NULL == batt))
00052     {
00053         ret_status = -1;
00054     }
00055     else
00056     {
00057         switch (pack_type)
00058         {
00059             case power_pack:
00060             {
00061                 if (init_inital_ts)
00062                 {
00063                     set_initial_timestamp(pwr->time_stamp);
00064                     init_inital_ts = false;
00065                 }
00066                 states_t current_state = determine_state(pwr);
00067                 if (current_state != prev_state)
00068                 {
00069                     if(time_check(prev_ts_ms, pwr->time_stamp))
```

```

00070         {
00071             printf("S;%u;%u-%u\n",
00072                   calc_time_from_start_ms_to_sec(pwr->time_stamp),
00073                   prev_state,
00074                   current_state);
00075         }
00076     }
00077     prev_state = current_state;
00078     break;
00079 }
00080 case battery_pack:
00081 {
00082     if (init_intial_ts)
00083     {
00084         set_initial_timestamp(batt->time_stamp);
00085         init_intial_ts = false;
00086     }
00087     printf("B;%u;%s\n",
00088           calc_time_from_start_ms_to_sec(batt->time_stamp),
00089           batt_states[batt->batt_status]);
00090     break;
00091 }
00092 default:
00093 {
00094     ret_status = -1;
00095     break;
00096 }
00097 }
00098 }
00099
00100 return ret_status;
00101 }
00102
00103 /**
00104  * @brief Function for calculating the time since the start of the program
00105  *
00106  * @param current_ts_ms Current timestamp in milliseconds
00107  * @return uint32_t Time from start of program in seconds
00108  */
00109 uint32_t calc_time_from_start_ms_to_sec(uint32_t current_ts_ms)
00110 {
00111     uint32_t ts_ms = (current_ts_ms - initial_ts_ms);
00112     return (ts_ms / 1000);
00113 }
00114
00115 /**
00116  * @brief Function for determining the state from power
00117  *
00118  * @param pwr Pointer to the struct that holds the power information
00119  * @return states_t Current state, returns NUM_OF_STATES if invalid
00120  */
00121 states_t determine_state(pwr_packet_t const * const pwr)
00122 {
00123     states_t ret_state = NUM_OF_STATES;
00124     if (STATE_0_UPPER_BOUNDS >= pwr->milliwatts)
00125     {
00126         ret_state = STATE_0;
00127     }
00128     else if ((STATE_1_LOWER_BOUNDS <= pwr->milliwatts)
00129             && (STATE_1_UPPER_BOUNDS >= pwr->milliwatts))
00130     {
00131         ret_state = STATE_1;
00132     }
00133     else if ((STATE_2_LOWER_BOUNDS <= pwr->milliwatts)
00134             && (STATE_2_UPPER_BOUNDS >= pwr->milliwatts))
00135     {
00136         ret_state = STATE_2;
00137     }
00138     else if ((STATE_3_LOWER_BOUNDS <= pwr->milliwatts)
00139             && (STATE_3_UPPER_BOUNDS >= pwr->milliwatts))
00140     {
00141         ret_state = STATE_3;
00142     }
00143     else
00144     {
00145         //do nothing
00146     }
00147     return ret_state;
00148 }
00149 }
00150
00151 /**
00152  * @brief Set the initial timestamp object
00153  *
00154  * @param ts Initial timestamp to be saved in milliseconds
00155  */
00156 void set_initial_timestamp(const uint32_t ts_ms)

```

```

00157 {
00158     initial_ts_ms = ts_ms;
00159 }
00160
00161 /**
00162  * @brief Function to check if 2 timestamps are greater than 10 ms apart
00163  *
00164  * @todo There is probably a better way to handle an uint32_t overflow or
00165  *        underflow situation.
00166  *
00167  * @param prev_ts_ms Previous timestamp
00168  * @param current_ts_ms Current timestamp
00169  * @return true If (current - prev) > 10ms
00170  * @return false If (current - prev) < 10ms
00171  */
00172 bool time_check(uint32_t prev_ts_ms, uint32_t current_ts_ms)
00173 {
00174     bool greater_than_10 = false;
00175
00176     if (prev_ts_ms < current_ts_ms)
00177     {
00178         if (10u < (current_ts_ms - prev_ts_ms))
00179         {
00180             greater_than_10 = true;
00181         }
00182     }
00183
00184     return greater_than_10;
00185 }

```

6.16 src/state_handler.h File Reference

Module for handling any state related functions.

```
#include "packet_constants.h"
```

```
#include <stdint.h>
```

Include dependency graph for state_handler.h: This graph shows which files directly or indirectly include this file:

Functions

- int [process_state_and_transitions](#) ([packet_type_t](#) pack_type, [pwr_packet_t](#) const *const pwr, [batt_packet_t](#) const *const batt)

Function for processing the states of either a power or battery state.

6.16.1 Detailed Description

Module for handling any state related functions.

Definition in file [state_handler.h](#).

6.16.2 Function Documentation

6.16.2.1 process_state_and_transitions()

```

int process_state_and_transitions (
    packet_type_t pack_type,
    pwr_packet_t const *const pwr,
    batt_packet_t const *const batt )

```

Function for processing the states of either a power or battery state.

Parameters

<i>pack_type</i>	Type of packet being processed
<i>pwr</i>	Pointer to the struct that holds the power information
<i>batt</i>	Pointer to the struct that holds the battery information

Returns

int 0 == Success

Definition at line 43 of file [state_handler.c](#).

6.17 state_handler.h

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

```
00001 /**
00002  * @file state_handler.h
00003  *
00004  * @brief Module for handling any state related functions.
00005  */
00006
00007 #ifndef STATE_HANDLER_H
00008 #define STATE_HANDLER_H
00009
00010 #include "packet_constants.h"
00011
00012 #include <stdint.h>
00013
00014 #ifdef __cplusplus
00015 extern "C" {
00016 #endif
00017
00018 int process_state_and_transitions(packet_type_t pack_type,
00019                                   pwr_packet_t const * const pwr,
00020                                   batt_packet_t const * const batt);
00021
00022 #ifdef __cplusplus
00023 }
00024 #endif
00025
00026 #endif // STATE_HANDLER_H
```

Index

- batt_packet, [9](#)
 - batt_status, [9](#)
 - err_check, [9](#)
 - time_stamp, [10](#)
- batt_packet_t
 - packet_constants.h, [23](#)
- BATT_PACKET_TYPE_BYTE
 - packet_constants.h, [20](#)
- BATT_START_OF_BATT_STATUS_LOC
 - packet_constants.h, [20](#)
- BATT_START_OF_ERR_CHECK_LOC
 - packet_constants.h, [20](#)
- BATT_STAT_NUM_OF_BYTES
 - packet_constants.h, [20](#)
- batt_states
 - state_handler.c, [36](#)
- batt_status
 - batt_packet, [9](#)
- battery_pack
 - packet_constants.h, [23](#)
- calc_power
 - calculations.h, [14](#)
- calc_time_from_start_ms_to_sec
 - state_handler.c, [34](#)
- calculations.h
 - calc_power, [14](#)
 - mod_of_array, [14](#)
- check_for_pack_error
 - packet_parser.c, [25](#)
- convert_array_to_uint32
 - packet_parser.c, [26](#)
- determine_packet_type
 - packet_parser.c, [26](#)
 - packet_parser.h, [30](#)
- determine_state
 - state_handler.c, [35](#)
- err_check
 - batt_packet, [9](#)
 - pwr_packet, [10](#)
- ERR_CHECK_NUM_OF_BYTES
 - packet_constants.h, [20](#)
- error_type
 - packet_constants.h, [23](#)
- get_pack_from_file
 - main.c, [16](#)
- inital_ts_ms
 - state_handler.c, [36](#)
- main
 - main.c, [16](#)
- main.c
 - get_pack_from_file, [16](#)
 - main, [16](#)
 - parse_arguments, [17](#)
- MAX_PACKET_SIZE
 - packet_constants.h, [20](#)
- MILLIAMP_NUM_OF_BYTES
 - packet_constants.h, [21](#)
- milliamps
 - pwr_packet, [11](#)
- milliwatts
 - pwr_packet, [11](#)
- mod_of_array
 - calculations.h, [14](#)
- NUM_OF_STATES
 - state_handler.c, [34](#)
- num_of_types
 - packet_constants.h, [23](#)
- packet_constants.h
 - batt_packet_t, [23](#)
 - BATT_PACKET_TYPE_BYTE, [20](#)
 - BATT_START_OF_BATT_STATUS_LOC, [20](#)
 - BATT_START_OF_ERR_CHECK_LOC, [20](#)
 - BATT_STAT_NUM_OF_BYTES, [20](#)
 - battery_pack, [23](#)
 - ERR_CHECK_NUM_OF_BYTES, [20](#)
 - error_type, [23](#)
 - MAX_PACKET_SIZE, [20](#)
 - MILLIAMP_NUM_OF_BYTES, [21](#)
 - num_of_types, [23](#)
 - PACKET_TYPE_NUM_OF_BYTES, [21](#)
 - packet_type_t, [23](#)
 - power_pack, [23](#)
 - pwr_packet_t, [23](#)
 - PWR_PACKET_TYPE_BYTE, [21](#)
 - PWR_START_OF_ERR_CHECK_LOC, [21](#)
 - PWR_START_OF_MILLIAMP_LOC, [21](#)
 - PWR_START_OF_VOLTS_LOC, [21](#)
 - SIZE_OF_BATT_PACK, [22](#)
 - SIZE_OF_PWR_PACK, [22](#)
 - START_OF_PT_LOC, [22](#)
 - START_OF_TS_LOC, [22](#)
 - TIMESTAMP_NUM_OF_BYTES, [22](#)
 - VOLTS_NUM_OF_BYTES, [23](#)

- packet_parser.c
 - check_for_pack_error, [25](#)
 - convert_array_to_uint32, [26](#)
 - determine_packet_type, [26](#)
 - process_batt_packet, [26](#)
 - process_pwr_packet, [27](#)
- packet_parser.h
 - determine_packet_type, [30](#)
 - process_batt_packet, [30](#)
 - process_pwr_packet, [31](#)
- PACKET_TYPE_NUM_OF_BYTES
 - packet_constants.h, [21](#)
- packet_type_t
 - packet_constants.h, [23](#)
- parse_arguments
 - main.c, [17](#)
- power_pack
 - packet_constants.h, [23](#)
- process_batt_packet
 - packet_parser.c, [26](#)
 - packet_parser.h, [30](#)
- process_pwr_packet
 - packet_parser.c, [27](#)
 - packet_parser.h, [31](#)
- process_state_and_transitions
 - state_handler.c, [35](#)
 - state_handler.h, [39](#)
- pwr_packet, [10](#)
 - err_check, [10](#)
 - milliamps, [11](#)
 - milliwatts, [11](#)
 - time_stamp, [11](#)
 - volts, [11](#)
- pwr_packet_t
 - packet_constants.h, [23](#)
- PWR_PACKET_TYPE_BYTE
 - packet_constants.h, [21](#)
- PWR_START_OF_ERR_CHECK_LOC
 - packet_constants.h, [21](#)
- PWR_START_OF_MILLIAMP_LOC
 - packet_constants.h, [21](#)
- PWR_START_OF_VOLTS_LOC
 - packet_constants.h, [21](#)
- README.md, [13](#)
- set_initial_timestamp
 - state_handler.c, [35](#)
- SIZE_OF_BATT_PACK
 - packet_constants.h, [22](#)
- SIZE_OF_PWR_PACK
 - packet_constants.h, [22](#)
- src/calculations.c, [13](#)
- src/calculations.h, [14](#), [15](#)
- src/main.c, [15](#), [17](#)
- src/packet_constants.h, [19](#), [24](#)
- src/packet_parser.c, [25](#), [27](#)
- src/packet_parser.h, [29](#), [31](#)
- src/state_handler.c, [32](#), [37](#)
- src/state_handler.h, [39](#), [40](#)
- START_OF_PT_LOC
 - packet_constants.h, [22](#)
- START_OF_TS_LOC
 - packet_constants.h, [22](#)
- STATE_0
 - state_handler.c, [34](#)
- STATE_0_UPPER_BOUNDS
 - state_handler.c, [33](#)
- STATE_1
 - state_handler.c, [34](#)
- STATE_1_LOWER_BOUNDS
 - state_handler.c, [33](#)
- STATE_1_UPPER_BOUNDS
 - state_handler.c, [33](#)
- STATE_2
 - state_handler.c, [34](#)
- STATE_2_LOWER_BOUNDS
 - state_handler.c, [33](#)
- STATE_2_UPPER_BOUNDS
 - state_handler.c, [33](#)
- STATE_3
 - state_handler.c, [34](#)
- STATE_3_LOWER_BOUNDS
 - state_handler.c, [33](#)
- STATE_3_UPPER_BOUNDS
 - state_handler.c, [34](#)
- state_handler.c
 - batt_states, [36](#)
 - calc_time_from_start_ms_to_sec, [34](#)
 - determine_state, [35](#)
 - inital_ts_ms, [36](#)
 - NUM_OF_STATES, [34](#)
 - process_state_and_transitions, [35](#)
 - set_initial_timestamp, [35](#)
 - STATE_0, [34](#)
 - STATE_0_UPPER_BOUNDS, [33](#)
 - STATE_1, [34](#)
 - STATE_1_LOWER_BOUNDS, [33](#)
 - STATE_1_UPPER_BOUNDS, [33](#)
 - STATE_2, [34](#)
 - STATE_2_LOWER_BOUNDS, [33](#)
 - STATE_2_UPPER_BOUNDS, [33](#)
 - STATE_3, [34](#)
 - STATE_3_LOWER_BOUNDS, [33](#)
 - STATE_3_UPPER_BOUNDS, [34](#)
 - states_t, [34](#)
 - time_check, [36](#)
- state_handler.h
 - process_state_and_transitions, [39](#)
- states_t
 - state_handler.c, [34](#)
- time_check
 - state_handler.c, [36](#)
- time_stamp
 - batt_packet, [10](#)
 - pwr_packet, [11](#)
- TIMESTAMP_NUM_OF_BYTES

packet_constants.h, [22](#)

volts

pwr_packet, [11](#)

VOLTS_NUM_OF_BYTES

packet_constants.h, [23](#)