EagleSat Telemetry and Telecommand System

Generated by Doxygen 1.9.3

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 estts Namespace Reference

Namespaces

- namespace ax25
- namespace cosmos
- namespace dispatcher
- namespace endurosat
- namespace es2_commands
- namespace es2_endpoint
- namespace es2_telemetry
- namespace estts_response_code
- namespace telem_handler
- namespace ti_serial
- namespace ti_socket

Classes

- struct command_object
- struct dispatched_command
- struct telemetry_object
- struct waiting_command

Typedefs

- typedef struct estts::command_object command_object
- typedef struct estts::telemetry_object telemetry_object
- typedef struct estts::dispatched_command dispatched_command
- typedef struct estts::waiting_command waiting_command
- typedef std::function< std::string(estts::command_object *, std::function< estts::Status(std::vector< estts::telemetry_object * >)>)> dispatch_fct

Enumerations

- enum Status {
  ES_OK = 0 , ES_SUCCESS = 0 , ES_UNSUCCESSFUL = 1 , ES_UNINITIALIZED = 2 ,
  ES_MEMORY_ERROR = 3 , ES_WAITING = 3 , ES_BAD_OPTION = 405 , ES_UNAUTHORIZED = 403 ,
  ES_SERVER_ERROR = 500 , ES_INPROGRESS = 300 , ES_NOTFOUND = 404 }

Variables

- const char REMOVABLE_STORAGE_NAME [ ] = "Samsung_T5"
- const int ESTTS_MAX_RETRIES = 2
- const int ESTTS_RETRY_WAIT_SEC = 1
- const int ESTTS_AWAIT_RESPONSE_PERIOD_SEC = 5
- const int ESTTS_SATELLITE_CONNECTION_TIMEOUT_MIN = 90
- const int ESTTS_CHECK_SATELLITE_INRANGE_INTERVAL_SEC = 30
- const int ESTTS_REQUEST_SESSION_TIMEOUT_SECONDS = 300

5.1.1 Typedef Documentation

5.1.1.1 command_object

```
typedef struct estts::command_object estts::command_object
```

5.1.1.2 dispatch_fct

```
typedef std::function<std::string(estts::command_object *, std::function<estts::Status(std↩
::vector<estts::telemetry_object *>)>)> estts::dispatch_fct
```

Definition at line 298 of file constants.h.

5.1.1.3 dispatched_command

```
typedef struct estts::dispatched_command estts::dispatched_command
```

5.1.1.4 telemetry_object

```
typedef struct estts::telemetry_object estts::telemetry_object
```

5.1.1.5 waiting_command

```
typedef struct estts::waiting_command estts::waiting_command
```

5.1.2 Enumeration Type Documentation

5.1.2.1 Status

```
enum estts::Status
```

Enumerator

ES_OK	
ES_SUCCESS	
ES_UNSUCCESSFUL	
ES_UNINITIALIZED	
ES_MEMORY_ERROR	
ES_WAITING	
ES_BAD_OPTION	
ES_UNAUTHORIZED	
ES_SERVER_ERROR	
ES_INPROGRESS	
ES_NOTFOUND	

Definition at line 77 of file constants.h.

5.1.3 Variable Documentation

5.1.3.1 ESTTS_AWAIT_RESPONSE_PERIOD_SEC

```
const int estts::ESTTS_AWAIT_RESPONSE_PERIOD_SEC = 5
```

Definition at line 17 of file constants.h.

5.1.3.2 ESTTS_CHECK_SATELLITE_INRANGE_INTERVAL_SEC

```
const int estts::ESTTS_CHECK_SATELLITE_INRANGE_INTERVAL_SEC = 30
```

Definition at line 19 of file constants.h.

5.1.3.3 ESTTS_MAX_RETRIES

```
const int estts::ESTTS_MAX_RETRIES = 2
```

Definition at line 15 of file constants.h.

5.1.3.4 ESTTS_REQUEST_SESSION_TIMEOUT_SECONDS

```
const int estts::ESTTS_REQUEST_SESSION_TIMEOUT_SECONDS = 300
```

Definition at line 20 of file constants.h.

5.1.3.5 ESTTS_RETRY_WAIT_SEC

```
const int estts::ESTTS_RETRY_WAIT_SEC = 1
```

Definition at line 16 of file constants.h.

5.1.3.6 ESTTS_SATELLITE_CONNECTION_TIMEOUT_MIN

```
const int estts::ESTTS_SATELLITE_CONNECTION_TIMEOUT_MIN = 90
```

Definition at line 18 of file constants.h.

5.1.3.7 REMOVABLE_STORAGE_NAME

```
const char estts::REMOVABLE_STORAGE_NAME[] = "Samsung_T5"
```

Definition at line 13 of file constants.h.

5.2 estts::ax25 Namespace Reference

Variables

- const char AX25_FLAG [] = "7E"
- const char AX25_DESTINATION_ADDRESS [] = "NABCDE"
- const char AX25_SSID0 [] = "E0"
- const char AX25_SOURCE_ADDRESS [] = "NEDCBA"
- const char AX25_SSID1 [] = "E1"
- const char AX25_CONTROL [] = "03"
- const char AX25_PID [] = "F0"
- const char NEW_SESSION_FRAME [] = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
- const char END_SESSION_FRAME [] = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"

5.2.1 Variable Documentation

5.2.1.1 AX25_CONTROL

```
const char estts::ax25::AX25_CONTROL[] = "03"
```

Definition at line 48 of file constants.h.

5.2.1.2 AX25_DESTINATION_ADDRESS

```
const char estts::ax25::AX25_DESTINATION_ADDRESS[] = "NABCDE"
```

Definition at line 44 of file constants.h.

5.2.1.3 AX25_FLAG

```
const char estts::ax25::AX25_FLAG[] = "7E"
```

Definition at line 43 of file constants.h.

5.2.1.4 AX25_PID

```
const char estts::ax25::AX25_PID[] = "F0"
```

Definition at line 49 of file constants.h.

5.2.1.5 AX25_SOURCE_ADDRESS

```
const char estts::ax25::AX25_SOURCE_ADDRESS[] = "NEDCBA"
```

Definition at line 46 of file constants.h.

5.2.1.6 AX25_SSID0

```
const char estts::ax25::AX25_SSID0[] = "E0"
```

Definition at line 45 of file constants.h.

5.2.1.7 AX25_SSID1

```
const char estts::ax25::AX25_SSID1[] = "E1"
```

Definition at line 47 of file constants.h.

5.2.1.8 END_SESSION_FRAME

```
const char estts::ax25::END_SESSION_FRAME[] = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
```

Definition at line 52 of file constants.h.

5.2.1.9 NEW_SESSION_FRAME

```
const char estts::ax25::NEW_SESSION_FRAME[] = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
```

Definition at line 51 of file constants.h.

5.3 estts::cosmos Namespace Reference

Variables

- const char COSMOS_SERVER_ADDR [ ] = "172.30.95.164"
- const int COSMOS_PRIMARY_CMD_TELEM_PORT = 65432
- const int COSMOS_GROUNDSTATION_CMD_TELEM_PORT = 8046
- const int COSMOS_SATELLITE_TXVR_CMD_TELEM_PORT = 55927

5.3.1 Variable Documentation

5.3.1.1 COSMOS_GROUNDSTATION_CMD_TELEM_PORT

```
const int estts::cosmos::COSMOS_GROUNDSTATION_CMD_TELEM_PORT = 8046
```

Definition at line 25 of file constants.h.

5.3.1.2 COSMOS_PRIMARY_CMD_TELEM_PORT

```
const int estts::cosmos::COSMOS_PRIMARY_CMD_TELEM_PORT = 65432
```

Definition at line 24 of file constants.h.

5.3.1.3 COSMOS_SATELLITE_TXVR_CMD_TELEM_PORT

```
const int estts::cosmos::COSMOS_SATELLITE_TXVR_CMD_TELEM_PORT = 55927
```

Definition at line 26 of file constants.h.

5.3.1.4 COSMOS_SERVER_ADDR

```
const char estts::cosmos::COSMOS_SERVER_ADDR[] = "172.30.95.164"
```

Definition at line 23 of file constants.h.

5.4 estts::dispatcher Namespace Reference

Variables

- const int MAX_COMPLETED_CACHE = 20

5.4.1 Variable Documentation

5.4.1.1 MAX_COMPLETED_CACHE

```
const int estts::dispatcher::MAX_COMPLETED_CACHE = 20
```

Definition at line 92 of file constants.h.

5.5 estts::endurosat Namespace Reference

Classes

- class esttc_const

Enumerations

- enum PIPE_State { PIPE_OFF = 0 , PIPE_WAITING = 1 , PIPE_ON = 2 }

Variables

- const int PIPE_DURATION_SEC = 10
- const int MAX_RETRIES = 2
- const int WAIT_TIME_SEC = 2
- const int ES_BAUD = 115200
- const int MAX_ES_TXVR_TEMP = 50

5.5.1 Enumeration Type Documentation

5.5.1.1 PIPE_State

```
enum estts::endurosat::PIPE_State
```

Enumerator

PIPE_OFF	
PIPE_WAITING	
PIPE_ON	

Definition at line 200 of file constants.h.

5.5.2 Variable Documentation

5.5.2.1 ES_BAUD

```
const int estts::endurosat::ES_BAUD = 115200
```

Definition at line 198 of file constants.h.

5.5.2.2 MAX_ES_TXVR_TEMP

```
const int estts::endurosat::MAX_ES_TXVR_TEMP = 50
```

Definition at line 199 of file constants.h.

5.5.2.3 MAX_RETRIES

```
const int estts::endurosat::MAX_RETRIES = 2
```

Definition at line 196 of file constants.h.

5.5.2.4 PIPE_DURATION_SEC

```
const int estts::endurosat::PIPE_DURATION_SEC = 10
```

Definition at line 195 of file constants.h.

5.5.2.5 WAIT_TIME_SEC

```
const int estts::endurosat::WAIT_TIME_SEC = 2
```

Definition at line 197 of file constants.h.

5.6 estts::es2_commands Namespace Reference

Namespaces

- namespace acs
- namespace crp
- namespace eps
- namespace mde
- namespace method
- namespace obc

5.7 estts::es2_commands::acs Namespace Reference

Variables

- const int ACS_GET_GPS_LAT = 01
- const int ACS_GET_GPS_LONG = 02
- const int ACS_GET_POS = 03
- const int ACS_DEP_MAG_BOOM = 07
- const int ACS_ENABLE = 10
- const int ACS_POWER = 11
- const int ACS_SET_CTRL_MODE = 13
- const int ACS_SET_EST_MODE = 14
- const int ACS_SET_MAG_MNT = 33
- const int ACS_SET_MAG_MNT_MTRX = 34
- const int ACS_SET_INERTIA = 41
- const int ACS_SAVE_CONFIG = 63
- const int ACS_SET_ATT_ANG = 146
- const int ACS_SET_ANG_RATE = 147
- const int ACS_GET_MAGNET = 151
- const int ACS_RATE_SENSE_RATE = 155
- const int ACS_SET_MAGNETORQUER = 157
- const int ACS_GET_MAGNETO = 170
- const int ACS_GET_CC_CURRENT = 172
- const int ACS_EST_ANG_RATES_FINE = 201

5.7.1 Variable Documentation

5.7.1.1 ACS_DEP_MAG_BOOM

```
const int estts::es2_commands::acs::ACS_DEP_MAG_BOOM = 07
```

Definition at line 100 of file constants.h.

5.7.1.2 ACS_ENABLE

```
const int estts::es2_commands::acs::ACS_ENABLE = 10
```

Definition at line 101 of file constants.h.

5.7.1.3 ACS_EST_ANG_RATES_FINE

```
const int estts::es2_commands::acs::ACS_EST_ANG_RATES_FINE = 201
```

Definition at line 116 of file constants.h.

5.7.1.4 ACS_GET_CC_CURRENT

```
const int estts::es2_commands::acs::ACS_GET_CC_CURRENT = 172
```

Definition at line 115 of file constants.h.

5.7.1.5 ACS_GET_GPS_LAT

```
const int estts::es2_commands::acs::ACS_GET_GPS_LAT = 01
```

Definition at line 97 of file constants.h.

5.7.1.6 ACS_GET_GPS_LONG

```
const int estts::es2_commands::acs::ACS_GET_GPS_LONG = 02
```

Definition at line 98 of file constants.h.

5.7.1.7 ACS_GET_MAGNET

```
const int estts::es2_commands::acs::ACS_GET_MAGNET = 151
```

Definition at line 111 of file constants.h.

5.7.1.8 ACS_GET_MAGNETO

```
const int estts::es2_commands::acs::ACS_GET_MAGNETO = 170
```

Definition at line 114 of file constants.h.

5.7.1.9 ACS_GET_POS

```
const int estts::es2_commands::acs::ACS_GET_POS = 03
```

Definition at line 99 of file constants.h.

5.7.1.10 ACS_POWER

```
const int estts::es2_commands::acs::ACS_POWER = 11
```

Definition at line 102 of file constants.h.

5.7.1.11 ACS_RATE_SENSE_RATE

```
const int estts::es2_commands::acs::ACS_RATE_SENSE_RATE = 155
```

Definition at line 112 of file constants.h.

5.7.1.12 ACS_SAVE_CONFIG

```
const int estts::es2_commands::acs::ACS_SAVE_CONFIG = 63
```

Definition at line 108 of file constants.h.

5.7.1.13 ACS_SET_ANG_RATE

```
const int estts::es2_commands::acs::ACS_SET_ANG_RATE = 147
```

Definition at line 110 of file constants.h.

5.7.1.14 ACS_SET_ATT_ANG

```
const int estts::es2_commands::acs::ACS_SET_ATT_ANG = 146
```

Definition at line 109 of file constants.h.

5.7.1.15 ACS_SET_CTRL_MODE

```
const int estts::es2_commands::acs::ACS_SET_CTRL_MODE = 13
```

Definition at line 103 of file constants.h.

5.7.1.16 ACS_SET_EST_MODE

```
const int estts::es2_commands::acs::ACS_SET_EST_MODE = 14
```

Definition at line 104 of file constants.h.

5.7.1.17 ACS_SET_INERTIA

```
const int estts::es2_commands::acs::ACS_SET_INERTIA = 41
```

Definition at line 107 of file constants.h.

5.7.1.18 ACS_SET_MAG_MNT

```
const int estts::es2_commands::acs::ACS_SET_MAG_MNT = 33
```

Definition at line 105 of file constants.h.

5.7.1.19 ACS_SET_MAG_MNT_MTRX

```
const int estts::es2_commands::acs::ACS_SET_MAG_MNT_MTRX = 34
```

Definition at line 106 of file constants.h.

5.7.1.20 ACS_SET_MAGNETORQUER

```
const int estts::es2_commands::acs::ACS_SET_MAGNETORQUER = 157
```

Definition at line 113 of file constants.h.

5.8 estts::es2_commands::crp Namespace Reference

Variables

- const int CRP_GET_DATA = 01

5.8.1 Variable Documentation

5.8.1.1 CRP_GET_DATA

```
const int estts::es2_commands::crp::CRP_GET_DATA = 01
```

Definition at line 138 of file constants.h.

5.9 estts::es2_commands::eps Namespace Reference

Variables

- const int EPS_GET_HEALTH = 01
- const int EPS_GET_COMMAND_43 = 43
- const int EPS_GET_BATTERY_VOLTAGE = 1
- const int EPS_GET_BATTERY_CURRENT = 2
- const int EPS_GET_5VBUS_CURRENT = 15
- const int EPS_GET_3VBUS_CURRENT = 14
- const int EPS_GET_TEMP_SENSOR5 = 38
- const int EPS_GET_TEMP_SENSOR6 = 39
- const int EPS_GET_TEMP_SENSOR7 = 40
- const int EPS_GET_BATTERY_TEMP_SENSOR1 = 19
- const int EPS_GET_BATTERY_TEMP_SENSOR2 = 20
- const int EPS_GET_BATTERY_TEMP_SENSOR3 = 21
- const int EPS_GET_BATTERY_TEMP_SENSOR4 = 22

5.9.1 Variable Documentation

5.9.1.1 EPS_GET_3VBUS_CURRENT

```
const int estts::es2_commands::eps::EPS_GET_3VBUS_CURRENT = 14
```

Definition at line 124 of file constants.h.

5.9.1.2 EPS_GET_5VBUS_CURRENT

```
const int estts::es2_commands::eps::EPS_GET_5VBUS_CURRENT = 15
```

Definition at line 123 of file constants.h.

5.9.1.3 EPS_GET_BATTERY_CURRENT

```
const int estts::es2_commands::eps::EPS_GET_BATTERY_CURRENT = 2
```

Definition at line 122 of file constants.h.

5.9.1.4 EPS_GET_BATTERY_TEMP_SENSOR1

```
const int estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR1 = 19
```

Definition at line 128 of file constants.h.

5.9.1.5 EPS_GET_BATTERY_TEMP_SENSOR2

```
const int estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR2 = 20
```

Definition at line 129 of file constants.h.

5.9.1.6 EPS_GET_BATTERY_TEMP_SENSOR3

```
const int estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR3 = 21
```

Definition at line 130 of file constants.h.

5.9.1.7 EPS_GET_BATTERY_TEMP_SENSOR4

```
const int estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR4 = 22
```

Definition at line 131 of file constants.h.

5.9.1.8 EPS_GET_BATTERY_VOLTAGE

```
const int estts::es2_commands::eps::EPS_GET_BATTERY_VOLTAGE = 1
```

Definition at line 121 of file constants.h.

5.9.1.9 EPS_GET_COMMAND_43

```
const int estts::es2_commands::eps::EPS_GET_COMMAND_43 = 43
```

Definition at line 120 of file constants.h.

5.9.1.10 EPS_GET_HEALTH

```
const int estts::es2_commands::eps::EPS_GET_HEALTH = 01
```

Definition at line 119 of file constants.h.

5.9.1.11 EPS_GET_TEMP_SENSOR5

```
const int estts::es2_commands::eps::EPS_GET_TEMP_SENSOR5 = 38
```

Definition at line 125 of file constants.h.

5.9.1.12 EPS_GET_TEMP_SENSOR6

```
const int estts::es2_commands::eps::EPS_GET_TEMP_SENSOR6 = 39
```

Definition at line 126 of file constants.h.

5.9.1.13 EPS_GET_TEMP_SENSOR7

```
const int estts::es2_commands::eps::EPS_GET_TEMP_SENSOR7 = 40
```

Definition at line 127 of file constants.h.

5.10 estts::es2_commands::mde Namespace Reference

Variables

- const int MDE_GET_STATUS = 01

5.10.1 Variable Documentation

5.10.1.1 MDE_GET_STATUS

```
const int estts::es2_commands::mde::MDE_GET_STATUS = 01
```

Definition at line 135 of file constants.h.

5.11 estts::es2_commands::method Namespace Reference

Variables

- const int ES_READ = 0
- const int ES_WRITE = 1

5.11.1 Variable Documentation

5.11.1.1 ES_READ

```
const int estts::es2_commands::method::ES_READ = 0
```

Definition at line 144 of file constants.h.

5.11.1.2 ES_WRITE

```
const int estts::es2_commands::method::ES_WRITE = 1
```

Definition at line 145 of file constants.h.

5.12 estts::es2_commands::obc Namespace Reference

Variables

- const int OBC_GET_HEALTH = 01

5.12.1 Variable Documentation

5.12.1.1 OBC_GET_HEALTH

```
const int estts::es2_commands::obc::OBC_GET_HEALTH = 01
```

Definition at line 141 of file constants.h.

5.13 estts::es2_endpoint Namespace Reference

Variables

- const int ES_OBC = 01
- const int ES_EPS = 02
- const int ES_ACS = 03
- const int ES_CRP = 05
- const int ES_MDE = 04
- const int ES_OFFLINE_LOG = 05
- const int ES_TELEMETRY = 06

5.13.1 Variable Documentation

5.13.1.1 ES_ACS

```
const int estts::es2_endpoint::ES_ACS = 03
```

Definition at line 69 of file constants.h.

5.13.1.2 ES_CRP

```
const int estts::es2_endpoint::ES_CRP = 05
```

Definition at line 70 of file constants.h.

5.13.1.3 ES_EPS

```
const int estts::es2_endpoint::ES_EPS = 02
```

Definition at line 68 of file constants.h.

5.13.1.4 ES_MDE

```
const int estts::es2_endpoint::ES_MDE = 04
```

Definition at line 71 of file constants.h.

5.13.1.5 ES_OBC

```
const int estts::es2_endpoint::ES_OBC = 01
```

Definition at line 67 of file constants.h.

5.13.1.6 ES_OFFLINE_LOG

```
const int estts::es2_endpoint::ES_OFFLINE_LOG = 05
```

Definition at line 72 of file constants.h.

5.13.1.7 ES_TELEMETRY

```
const int estts::es2_endpoint::ES_TELEMETRY = 06
```

Definition at line 73 of file constants.h.

5.14 estts::es2_telemetry Namespace Reference

Namespaces

- namespace acs
- namespace eps

5.15 estts::es2_telemetry::acs Namespace Reference

5.16 estts::es2_telemetry::eps Namespace Reference

Classes

- struct eps_3Vbus_current
- struct eps_5Vbus_current
- struct eps_batteryTemp_sensor1
- struct eps_batteryTemp_sensor2
- struct eps_batteryTemp_sensor3
- struct eps_batteryTemp_sensor4
- struct eps_current
- struct eps_externalTemp_sensor5
- struct eps_externalTemp_sensor6
- struct eps_externalTemp_sensor7
- struct eps_voltage
- struct vitals

5.17 estts::estts_response_code Namespace Reference

Variables

- const int SUCCESS = 0
- const int UNRECOGNIZED_REQUEST = 1
- const int OBC_FAILURE = 2

5.17.1 Variable Documentation

5.17.1.1 OBC_FAILURE

```
const int estts::estts_response_code::OBC_FAILURE = 2
```

Definition at line 62 of file constants.h.

5.17.1.2 SUCCESS

```
const int estts::estts_response_code::SUCCESS = 0
```

Definition at line 60 of file constants.h.

5.17.1.3 UNRECOGNIZED_REQUEST

```
const int estts::estts_response_code::UNRECOGNIZED_REQUEST = 1
```

Definition at line 61 of file constants.h.

5.18 estts::telem_handler Namespace Reference

Variables

- const char TELEM_HANDLER_STATE_FILE [ ] = "es2_state.json"

5.18.1 Variable Documentation

5.18.1.1 TELEM_HANDLER_STATE_FILE

```
const char estts::telem_handler::TELEM_HANDLER_STATE_FILE[] = "es2_state.json"
```

Definition at line 56 of file constants.h.

5.19 estts::ti_serial Namespace Reference

Variables

- const char TI_SERIAL_ADDRESS [] = "/dev/cu.usbserial-A10JVB3P"

5.19.1 Variable Documentation

5.19.1.1 TI_SERIAL_ADDRESS

```
const char estts::ti_serial::TI_SERIAL_ADDRESS[] = "/dev/cu.usbserial-A10JVB3P"
```

Definition at line 30 of file constants.h.

5.20 estts::ti_socket Namespace Reference

Variables

- const int MAX_RETRIES = 2
- const int WAIT_TIME_SEC = 2
- const int TI_SOCKET_BUF_SZ = 1024
- const char TI_SOCKET_ADDRESS [] = "127.0.0.1"
- const int TI_SOCKET_PORT = 65548

5.20.1 Variable Documentation

5.20.1.1 MAX_RETRIES

```
const int estts::ti_socket::MAX_RETRIES = 2
```

Definition at line 34 of file constants.h.

5.20.1.2 TI_SOCKET_ADDRESS

```
const char estts::ti_socket::TI_SOCKET_ADDRESS[] = "127.0.0.1"
```

Definition at line 37 of file constants.h.

5.20.1.3 TI_SOCKET_BUF_SZ

```
const int estts::ti_socket::TI_SOCKET_BUF_SZ = 1024
```

Definition at line 36 of file constants.h.

5.20.1.4 TI_SOCKET_PORT

```
const int estts::ti_socket::TI_SOCKET_PORT = 65548
```

Definition at line 38 of file constants.h.

5.20.1.5 WAIT_TIME_SEC

```
const int estts::ti_socket::WAIT_TIME_SEC = 2
```

Definition at line 35 of file constants.h.

Chapter 6

Class Documentation

6.1 acs_command Class Reference

`#include <acs_command.h>`

Public Member Functions

- estts::Status get_current_position ()
- std::string deploy_magnetometer_boom (const estts::dispatch_fct &dispatch)
- std::string enable_acs (const estts::dispatch_fct &dispatch)
- std::string power_acs (const estts::dispatch_fct &dispatch)
- std::string set_ctrl_mode (const estts::dispatch_fct &dispatch)
- std::string set_est_mode (const estts::dispatch_fct &dispatch)

6.1.1 Detailed Description

Definition at line 10 of file acs_command.h.

6.1.2 Member Function Documentation

6.1.2.1 deploy_magnetometer_boom()

```
std::string acs_command::deploy_magnetometer_boom (
            const estts::dispatch_fct & dispatch )
```

Definition at line 13 of file acs_command.cpp.

6.1.2.2 enable_acs()

```
std::string acs_command::enable_acs (
            const estts::dispatch_fct & dispatch )
```

Definition at line 36 of file acs_command.cpp.

6.1.2.3 get_current_position()

```
estts::Status acs_command::get_current_position ( )
```

Definition at line 7 of file acs_command.cpp.

6.1.2.4 power_acs()

```
std::string acs_command::power_acs (
            const estts::dispatch_fct & dispatch )
```

Definition at line 60 of file acs_command.cpp.

6.1.2.5 set_ctrl_mode()

```
std::string acs_command::set_ctrl_mode (
            const estts::dispatch_fct & dispatch )
```

Definition at line 83 of file acs_command.cpp.

6.1.2.6 set_est_mode()

```
std::string acs_command::set_est_mode (
            const estts::dispatch_fct & dispatch )
```

Definition at line 106 of file acs_command.cpp.

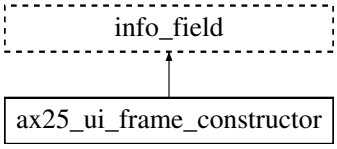
The documentation for this class was generated from the following files:

- src/fapi/command_handler/acs_command.h
- src/fapi/command_handler/acs_command.cpp

6.2 ax25_ui_frame_constructor Class Reference

`#include <ax25_ui_frame_constructor.h>`

Inheritance diagram for ax25_ui_frame_constructor:



Public Member Functions

- ax25_ui_frame_constructor (estts::command_object *command)
- std::string construct_ax25 ()

  Constructs and encodes AX.25 frame using AX.25 constant values.

Protected Member Functions

- std::string encode_ax25_frame (std::string raw)

  Encodes AX.25 frame for transmission. Encoding includes scrambling and NRZI.

Additional Inherited Members

6.2.1 Detailed Description

Definition at line 14 of file ax25_ui_frame_constructor.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 ax25_ui_frame_constructor()

```
ax25_ui_frame_constructor::ax25_ui_frame_constructor (
            estts::command_object * command )  [inline], [explicit]
```

Definition at line 46 of file ax25_ui_frame_constructor.h.

6.2.3 Member Function Documentation

6.2.3.1 construct_ax25()

```
string ax25_ui_frame_constructor::construct_ax25 ( )
```

Constructs and encodes AX.25 frame using AX.25 constant values.

Returns

Returns constructed and encoded AX.25 frame according to EnduroSat specification

Definition at line 95 of file ax25_ui_frame_constructor.cpp.

6.2.3.2 encode_ax25_frame()

```
std::string ax25_ui_frame_constructor::encode_ax25_frame (
            std::string raw )  [protected]
```

Encodes AX.25 frame for transmission. Encoding includes scrambling and NRZI.

Parameters

raw	Raw hexadecimal AX.25 frame

Returns

Returns encoded frame for transmission

Definition at line 164 of file ax25_ui_frame_constructor.cpp.

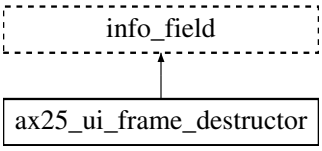
The documentation for this class was generated from the following files:

- src/tnc_emulator/ax25_ui_frame_constructor.h
- src/tnc_emulator/ax25_ui_frame_constructor.cpp

6.3 ax25_ui_frame_destructor Class Reference

```
#include <ax25_ui_frame_destructor.h>
```

Inheritance diagram for ax25_ui_frame_destructor:



Public Member Functions

- ax25_ui_frame_destructor (std::string raw)
- std::vector< estts::telemetry_object * > destruct_ax25 ()

Additional Inherited Members

6.3.1 Detailed Description

Definition at line 13 of file ax25_ui_frame_destructor.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 ax25_ui_frame_destructor()

```
ax25_ui_frame_destructor::ax25_ui_frame_destructor (
            std::string raw )  [inline], [explicit]
```

Definition at line 40 of file ax25_ui_frame_destructor.h.

6.3.3 Member Function Documentation

6.3.3.1 destruct_ax25()

```
std::vector< estts::telemetry_object * > ax25_ui_frame_destructor::destruct_ax25 ( )
```

Definition at line 152 of file ax25_ui_frame_destructor.cpp.

The documentation for this class was generated from the following files:

- src/tnc_emulator/ax25_ui_frame_destructor.h
- src/tnc_emulator/ax25_ui_frame_destructor.cpp

6.4 bin_converter Class Reference

```
#include <bin_converter.h>
```

Public Member Functions

- std::string toBinary (short int size, const std::string &hexField)
- std::string toBinary (const std::string &hexField)
- std::string toBinary (const unsigned char field[ ])
- std::string toBinary (unsigned char field)

6.4.1 Detailed Description

Definition at line 11 of file bin_converter.h.

6.4.2 Member Function Documentation

6.4.2.1 toBinary() [1/4]

```
std::string bin_converter::toBinary (
            const std::string & hexField )
```

6.4.2.2 toBinary() [2/4]

```
string bin_converter::toBinary (
            const unsigned char field[] )
```

@description Converts a ASCII character array to hex

Parameters

const	unsigned char field[]

Returns

string of bits (e.g. "10010101")

Definition at line 63 of file bin_converter.cpp.

6.4.2.3 toBinary() [3/4]

```
std::string bin_converter::toBinary (
            short int size,
            const std::string & hexField )
```

6.4.2.4 toBinary() [4/4]

```
string bin_converter::toBinary (
            unsigned char field )
```

@description Converts an unsigned character hex value to binary

Parameters

const	unsigned char field

Returns

string of bits (e.g. "10010101")

Definition at line 79 of file bin_converter.cpp.

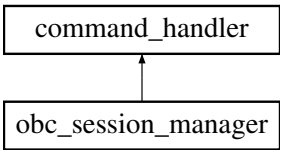
The documentation for this class was generated from the following files:

- src/utils/bin_converter.h
- src/utils/bin_converter.cpp

6.5 command_handler Class Reference

```
#include <command_handler.h>
```

Inheritance diagram for command_handler:



Protected Member Functions

- command_handler ()

  Default constructor. Note that the command_handler requires that init_command_handler be called before use.
- estts::Status init_command_handler (transmission_interface *ti)

  Initializes command handler by allocating local transmission interface instance.
- ~command_handler ()
- estts::Status execute (estts::waiting_command *command)

Protected Attributes

- std::vector< estts::dispatched_command * > dispatched
- std::vector< completed * > completed_cache

6.5.1 Detailed Description

Definition at line 15 of file command_handler.h.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 command_handler()

```
command_handler::command_handler ( )  [explicit], [protected]
```

Default constructor. Note that the command_handler requires that init_command_handler be called before use.

Definition at line 32 of file command_handler.cpp.

6.5.2.2 ~command_handler()

```
command_handler::~command_handler ( )  [protected], [default]
```

6.5.3 Member Function Documentation

6.5.3.1 execute()

```
estts::Status command_handler::execute (
            estts::waiting_command * command )  [protected]
```

Primary command execution point. Takes argument for a waiting command object and calls associated execution handler.

Parameters

command	Pointer to an estts::waiting_command object.

Returns

ES_OK if command was executed properly.

Definition at line 36 of file command_handler.cpp.

6.5.3.2 init_command_handler()

```
estts::Status command_handler::init_command_handler (
            transmission_interface * ti )  [protected]
```

Initializes command handler by allocating local transmission interface instance.

Parameters

ti	Transmission interface object

Returns

Definition at line 99 of file command_handler.cpp.

6.5.4 Member Data Documentation

6.5.4.1 completed_cache

```
std::vector<completed *> command_handler::completed_cache  [protected]
```

Definition at line 42 of file command_handler.h.

6.5.4.2 dispatched

```
std::vector<estts::dispatched_command *> command_handler::dispatched  [protected]
```

Definition at line 40 of file command_handler.h.

The documentation for this class was generated from the following files:

- src/fapi/command_handler.h
- src/fapi/command_handler.cpp

6.6 estts::command_object Struct Reference

```
#include <constants.h>
```

Public Attributes

- int address {}
- int timeStamp {}
- int sequence {}
- int commandID {}
- int method {}
- const char * data {}

6.6.1 Detailed Description

Definition at line 261 of file constants.h.

6.6.2 Member Data Documentation

6.6.2.1 address

```
int estts::command_object::address {}
```

Definition at line 262 of file constants.h.

6.6.2.2 commandID

```
int estts::command_object::commandID {}
```

Definition at line 265 of file constants.h.

6.6.2.3 data

```
const char* estts::command_object::data {}
```

Definition at line 267 of file constants.h.

6.6.2.4 method

```
int estts::command_object::method {}
```

Definition at line 266 of file constants.h.

6.6.2.5 sequence

```
int estts::command_object::sequence {}
```

Definition at line 264 of file constants.h.

6.6.2.6 timeStamp

```
int estts::command_object::timeStamp {}
```

Definition at line 263 of file constants.h.

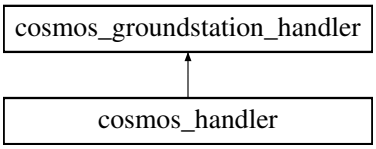
The documentation for this struct was generated from the following file:

- src/utils/constants.h

6.7 cosmos_groundstation_handler Class Reference

```
#include <cosmos_groundstation_handler.h>
```

Inheritance diagram for cosmos_groundstation_handler:



Public Member Functions

- cosmos_groundstation_handler ()
- estts::Status cosmos_groundstation_init (transmission_interface *ti)

6.7.1 Detailed Description

Definition at line 12 of file cosmos_groundstation_handler.h.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 cosmos_groundstation_handler()

```
cosmos_groundstation_handler::cosmos_groundstation_handler ( )
```

Default constructor that initializes socket.

Definition at line 8 of file cosmos_groundstation_handler.cpp.

6.7.3 Member Function Documentation

6.7.3.1 cosmos_groundstation_init()

```
estts::Status cosmos_groundstation_handler::cosmos_groundstation_init (
            transmission_interface * ti )
```

Function that initializes ESTTS to work with COSMOS. This includes defining the telemetry callback and creating the COSMOS worker thread

Returns

ES_OK if successful, anything else if not

Definition at line 22 of file cosmos_groundstation_handler.cpp.

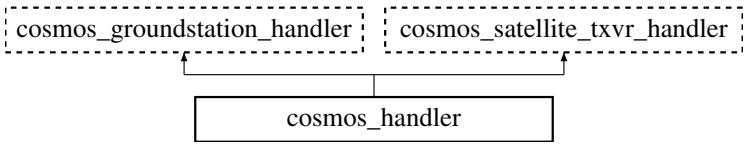
The documentation for this class was generated from the following files:

- src/fapi/cosmos_groundstation_handler.h
- src/fapi/cosmos_groundstation_handler.cpp

6.8 cosmos_handler Class Reference

```
#include <cosmos_handler.h>
```

Inheritance diagram for cosmos_handler:



Public Member Functions

- cosmos_handler ()
- estts::Status cosmos_init ()
- void initialize_cosmos_daemon ()

6.8.1 Detailed Description

The primary COSMOS handler is in charge of communication between COSMOS and the OBC. When the primary COSMOS handler receives a command from COSMOS, the logic that it uses inside ESTTS handles bi-directional communication between the ground station and the satellite. The primary COSMOS handler also manages the COSMOS groundstation and satellite transceiver handlers.

Definition at line 22 of file cosmos_handler.h.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 cosmos_handler()

```
cosmos_handler::cosmos_handler ( )
```

Default constructor that initializes socket.

Definition at line 10 of file cosmos_handler.cpp.

6.8.3 Member Function Documentation

6.8.3.1 cosmos_init()

```
estts::Status cosmos_handler::cosmos_init ( )
```

Function that initializes ESTTS to work with COSMOS. This includes defining the telemetry callback and creating the COSMOS worker thread

Returns

ES_OK if successful, anything else if not

Definition at line 26 of file cosmos_handler.cpp.

6.8.3.2 initialize_cosmos_daemon()

```
void cosmos_handler::initialize_cosmos_daemon ( )  [inline]
```

Joins the primary_cosmos_worker worker thread, which doesn't exit unless something catastrophic occurs. This function is used to initialize the ESTTS runtime, as it never returns.

Definition at line 71 of file cosmos_handler.h.

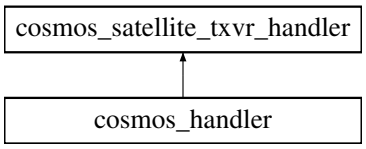
The documentation for this class was generated from the following files:

- src/fapi/cosmos_handler.h
- src/fapi/cosmos_handler.cpp

6.9 cosmos_satellite_txvr_handler Class Reference

`#include <cosmos_satellite_txvr_handler.h>`

Inheritance diagram for cosmos_satellite_txvr_handler:



Public Member Functions

- cosmos_satellite_txvr_handler ()
- estts::Status cosmos_satellite_txvr_init ()

6.9.1 Detailed Description

Definition at line 11 of file cosmos_satellite_txvr_handler.h.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 cosmos_satellite_txvr_handler()

`cosmos_satellite_txvr_handler::cosmos_satellite_txvr_handler ( )`

Default constructor that initializes socket.

Definition at line 5 of file cosmos_satellite_txvr_handler.cpp.

6.9.3 Member Function Documentation

6.9.3.1 cosmos_satellite_txvr_init()

`estts::Status cosmos_satellite_txvr_handler::cosmos_satellite_txvr_init ( )`

Function that initializes ESTTS to work with COSMOS. This includes defining the telemetry callback and creating the COSMOS worker thread

Returns

ES_OK if successful, anything else if not

Definition at line 9 of file cosmos_satellite_txvr_handler.cpp.

The documentation for this class was generated from the following files:

- src/fapi/cosmos_satellite_txvr_handler.h
- src/fapi/cosmos_satellite_txvr_handler.cpp

6.10 crp_command Class Reference

`#include <crp_command.h>`

6.10.1 Detailed Description

Definition at line 9 of file crp_command.h.

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

- src/fapi/command_handler/crp_command.h

6.11 estts::dispatched_command Struct Reference

`#include <constants.h>`

Public Attributes

- std::string frame
- command_object * command
- std::vector< telemetry_object * > telem_obj
- std::string telem_str
- Status response_code
- std::string serial_number
- std::function< estts::Status(std::vector< estts::telemetry_object * >)> obj_callback
- std::function< estts::Status(std::string)> str_callback

6.11.1 Detailed Description

Definition at line 279 of file constants.h.

6.11.2 Member Data Documentation

6.11.2.1 command

`command_object* estts::dispatched_command::command`

Definition at line 281 of file constants.h.

6.11.2.2 frame

```
std::string estts::dispatched_command::frame
```

Definition at line 280 of file constants.h.

6.11.2.3 obj_callback

```
std::function<estts::Status(std::vector<estts::telemetry_object *>)> estts::dispatched_↩
command::obj_callback
```

Definition at line 286 of file constants.h.

6.11.2.4 response_code

```
Status estts::dispatched_command::response_code
```

Definition at line 284 of file constants.h.

6.11.2.5 serial_number

```
std::string estts::dispatched_command::serial_number
```

Definition at line 285 of file constants.h.

6.11.2.6 str_callback

```
std::function<estts::Status(std::string)> estts::dispatched_command::str_callback
```

Definition at line 287 of file constants.h.

6.11.2.7 telem_obj

```
std::vector<telemetry_object *> estts::dispatched_command::telem_obj
```

Definition at line 282 of file constants.h.

6.11.2.8 telem_str

```
std::string estts::dispatched_command::telem_str
```

Definition at line 283 of file constants.h.

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

- src/utils/constants.h

6.12 estts::es2_telemetry::eps::eps_3Vbus_current Struct Reference

```
#include <constants.h>
```

Public Attributes

- double bus_current

6.12.1 Detailed Description

Definition at line 165 of file constants.h.

6.12.2 Member Data Documentation

6.12.2.1 bus_current

```
double estts::es2_telemetry::eps::eps_3Vbus_current::bus_current
```

Definition at line 166 of file constants.h.

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

- src/utils/constants.h

6.13 estts::es2_telemetry::eps::eps_5Vbus_current Struct Reference

```
#include <constants.h>
```

Public Attributes

- double bus_current

6.13.1 Detailed Description

Definition at line 162 of file constants.h.

6.13.2 Member Data Documentation

6.13.2.1 bus_current

```
double estts::es2_telemetry::eps::eps_5Vbus_current::bus_current
```

Definition at line 163 of file constants.h.

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

- src/utils/constants.h

6.14 estts::es2_telemetry::eps::eps_batteryTemp_sensor1 Struct Reference

```
#include <constants.h>
```

Public Attributes

- double battery_temperature

6.14.1 Detailed Description

Definition at line 177 of file constants.h.

6.14.2 Member Data Documentation

6.14.2.1 battery_temperature

```
double estts::es2_telemetry::eps::eps_batteryTemp_sensor1::battery_temperature
```

Definition at line 178 of file constants.h.

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

- src/utils/constants.h

6.15 estts::es2_telemetry::eps::eps_batteryTemp_sensor2 Struct Reference

`#include <constants.h>`

Public Attributes

- double battery_temperature

6.15.1 Detailed Description

Definition at line 180 of file constants.h.

6.15.2 Member Data Documentation

6.15.2.1 battery_temperature

`double estts::es2_telemetry::eps::eps_batteryTemp_sensor2::battery_temperature`

Definition at line 181 of file constants.h.

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

- src/utils/constants.h

6.16 estts::es2_telemetry::eps::eps_batteryTemp_sensor3 Struct Reference

`#include <constants.h>`

Public Attributes

- double battery_temperature

6.16.1 Detailed Description

Definition at line 183 of file constants.h.

6.16.2 Member Data Documentation

6.16.2.1 battery_temperature

```
double estts::es2_telemetry::eps::eps_batteryTemp_sensor3::battery_temperature
```

Definition at line 184 of file constants.h.

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

- src/utils/constants.h

6.17 estts::es2_telemetry::eps::eps_batteryTemp_sensor4 Struct Reference

```
#include <constants.h>
```

Public Attributes

- double battery_temperature

6.17.1 Detailed Description

Definition at line 186 of file constants.h.

6.17.2 Member Data Documentation

6.17.2.1 battery_temperature

```
double estts::es2_telemetry::eps::eps_batteryTemp_sensor4::battery_temperature
```

Definition at line 187 of file constants.h.

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

- src/utils/constants.h

6.18 eps_command Class Reference

`#include <eps_command.h>`

Public Member Functions

- eps_command ()
- std::string get_eps_vitals (const estts::dispatch_fct &dispatch, const std::function< estts::Status(estts::es2_telemetry::eps::vitals *)> &telem_callback)
- std::string get_eps_batteryVoltage (const estts::dispatch_fct &dispatch)
- std::string get_eps_batteryCurrent (const estts::dispatch_fct &dispatch)
- std::string get_eps_5Vbus_current (const estts::dispatch_fct &dispatch)
- std::string get_eps_3Vbus_current (const estts::dispatch_fct &dispatch)
- std::string get_eps_temp_sensor5 (const estts::dispatch_fct &dispatch)
- std::string get_eps_temp_sensor6 (const estts::dispatch_fct &dispatch)
- std::string get_eps_temp_sensor7 (const estts::dispatch_fct &dispatch)
- std::string get_eps_battery_temp_sensor1 (const estts::dispatch_fct &dispatch)
- std::string get_eps_battery_temp_sensor2 (const estts::dispatch_fct &dispatch)
- std::string get_eps_battery_temp_sensor3 (const estts::dispatch_fct &dispatch)
- std::string get_eps_battery_temp_sensor4 (const estts::dispatch_fct &dispatch)

6.18.1 Detailed Description

Definition at line 12 of file eps_command.h.

6.18.2 Constructor & Destructor Documentation

6.18.2.1 eps_command()

```
eps_command::eps_command ( )  [explicit], [default]
```

6.18.3 Member Function Documentation

6.18.3.1 get_eps_3Vbus_current()

```
std::string eps_command::get_eps_3Vbus_current (
            const estts::dispatch_fct & dispatch )
```

Definition at line 144 of file eps_command.cpp.

6.18.3.2 get_eps_5Vbus_current()

```
std::string eps_command::get_eps_5Vbus_current (
            const estts::dispatch_fct & dispatch )
```

Definition at line 112 of file eps_command.cpp.

6.18.3.3 get_eps_battery_temp_sensor1()

```
std::string eps_command::get_eps_battery_temp_sensor1 (
            const estts::dispatch_fct & dispatch )
```

Definition at line 270 of file eps_command.cpp.

6.18.3.4 get_eps_battery_temp_sensor2()

```
std::string eps_command::get_eps_battery_temp_sensor2 (
            const estts::dispatch_fct & dispatch )
```

Definition at line 301 of file eps_command.cpp.

6.18.3.5 get_eps_battery_temp_sensor3()

```
std::string eps_command::get_eps_battery_temp_sensor3 (
            const estts::dispatch_fct & dispatch )
```

Definition at line 332 of file eps_command.cpp.

6.18.3.6 get_eps_battery_temp_sensor4()

```
std::string eps_command::get_eps_battery_temp_sensor4 (
            const estts::dispatch_fct & dispatch )
```

Definition at line 363 of file eps_command.cpp.

6.18.3.7 get_eps_batteryCurrent()

```
std::string eps_command::get_eps_batteryCurrent (
            const estts::dispatch_fct & dispatch )
```

Definition at line 82 of file eps_command.cpp.

6.18.3.8 get_eps_batteryVoltage()

```
std::string eps_command::get_eps_batteryVoltage (
            const estts::dispatch_fct & dispatch )
```

Definition at line 54 of file eps_command.cpp.

6.18.3.9 get_eps_temp_sensor5()

```
std::string eps_command::get_eps_temp_sensor5 (
            const estts::dispatch_fct & dispatch )
```

Definition at line 177 of file eps_command.cpp.

6.18.3.10 get_eps_temp_sensor6()

```
std::string eps_command::get_eps_temp_sensor6 (
            const estts::dispatch_fct & dispatch )
```

Definition at line 208 of file eps_command.cpp.

6.18.3.11 get_eps_temp_sensor7()

```
std::string eps_command::get_eps_temp_sensor7 (
            const estts::dispatch_fct & dispatch )
```

Definition at line 239 of file eps_command.cpp.

6.18.3.12 get_eps_vitals()

```
std::string eps_command::get_eps_vitals (
            const estts::dispatch_fct & dispatch,
            const std::function< estts::Status(estts::es2_telemetry::eps::vitals *)> & telem↩
_callback )
```

Definition at line 8 of file eps_command.cpp.

The documentation for this class was generated from the following files:

- src/fapi/command_handler/eps_command.h
- src/fapi/command_handler/eps_command.cpp

6.19 estts::es2_telemetry::eps::eps_current Struct Reference

`#include <constants.h>`

Public Attributes

- double battery_current

6.19.1 Detailed Description

Definition at line 159 of file constants.h.

6.19.2 Member Data Documentation

6.19.2.1 battery_current

`double estts::es2_telemetry::eps::eps_current::battery_current`

Definition at line 160 of file constants.h.

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

- src/utils/constants.h

6.20 estts::es2_telemetry::eps::eps_externalTemp_sensor5 Struct Reference

`#include <constants.h>`

Public Attributes

- double external_temperature

6.20.1 Detailed Description

Definition at line 168 of file constants.h.

6.20.2 Member Data Documentation

6.20.2.1 external_temperature

```
double estts::es2_telemetry::eps::eps_externalTemp_sensor5::external_temperature
```

Definition at line 169 of file constants.h.

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

- src/utils/constants.h

6.21 estts::es2_telemetry::eps::eps_externalTemp_sensor6 Struct Reference

```
#include <constants.h>
```

Public Attributes

- double external_temperature

6.21.1 Detailed Description

Definition at line 171 of file constants.h.

6.21.2 Member Data Documentation

6.21.2.1 external_temperature

```
double estts::es2_telemetry::eps::eps_externalTemp_sensor6::external_temperature
```

Definition at line 172 of file constants.h.

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

- src/utils/constants.h

6.22 estts::es2_telemetry::eps::eps_externalTemp_sensor7 Struct Reference

```
#include <constants.h>
```

Public Attributes

- double external_temperature

6.22.1 Detailed Description

Definition at line 174 of file constants.h.

6.22.2 Member Data Documentation

6.22.2.1 external_temperature

```
double estts::es2_telemetry::eps::eps_externalTemp_sensor7::external_temperature
```

Definition at line 175 of file constants.h.

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

- src/utils/constants.h

6.23 estts::es2_telemetry::eps::eps_voltage Struct Reference

```
#include <constants.h>
```

Public Attributes

- double battery_voltage

6.23.1 Detailed Description

Definition at line 156 of file constants.h.

6.23.2 Member Data Documentation

6.23.2.1 battery_voltage

```
double estts::es2_telemetry::eps::eps_voltage::battery_voltage
```

Definition at line 157 of file constants.h.

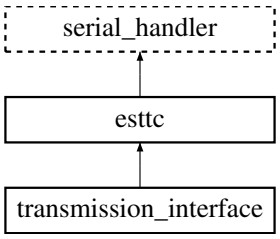
The documentation for this struct was generated from the following file:

- src/utils/constants.h

6.24 esttc Class Reference

```
#include <esttc.h>
```

Inheritance diagram for esttc:



Public Member Functions

- estts::Status default_mode ()

  Enables the default value of every SCW field on Endurosat UHF Transceiver module.
- estts::Status write_scw (uint16_t scw_command)

  Sends a command with a Status Control Word (SCW) that changes the Endurosat UHF Transceiver's settings.
- estts::Status read_scw (std::string &RSSI, std::string &dvc_addr, std::string &rst_ctr, std::string &scw)

  Send a command to read the current status of the Transceiver given its current Status Control Word (SCW)
- estts::Status write_radio_freq_config (const std::string &frac="76620F41", const std::string &div="41")
- estts::Status read_radio_freq (std::string &RSSI, std::string &frac, std::string &div)

  Get the radio frequency and the last RSSI (Received signal strength indication) of the UHF Transceiver module.
- estts::Status read_uptime (std::string &RSSI, std::string &uptime)

  Get the uptime of the UHF Transceiver.
- estts::Status read_trans_pckts (std::string &RSSI, std::string &pckt_num)

  Get the number of transmitted packets from the UHF Transceiver.
- estts::Status read_trans_pckts_crc (std::string &RSSI, std::string &pckt_num)

  Get the number of transmitted packets that contained CRC errors from the UHF Transceiver.
- estts::Status write_bcn_trans_period (const std::string &period="003C")

  Configure the beacon message transmission period of the UHF Transceiver.
- estts::Status read_bcn_trans_period (std::string &RSSI, std::string &period)
- estts::Status write_res_default_vals ()

  restore default values to UHF Transceiver for Destination/Source/Morse code call sign, Audio beacon period and message, Text beadon period, and Pipe timeout period
- estts::Status write_i2c_resist_config (const std::string &resistor_config)

  Configure the I2C Pull-Up resistor configuration of the UHF Transceiver.

- estts::Status read_i2c_resist_config (std::string &selected_resistor)

  Read the current I2C Pull-Up resistor configuration of the UHF Transceiver.
- estts::Status write_radio_crc16 (const std::string &mode)

  Enable or disable the UHF Transceiver's radio packet CRC16.
- estts::Status read_radio_crc16 (std::string &mode)

  Read whether radio packet CRC16 is enabled or disabled for the UHF Transceiver.
- estts::Status write_config_ax25_decode (const std::string &config_bit)

  Write the configuration bit that determines whether the UHF Transceiver's Auto AX.25 Decode is Enabled/Disabled.
- estts::Status read_config_ax25_decode (std::string &config_bit)

  Get the configuration bit that determines whether the UHF Transceiver's Auto AX.25 Decode is Enabled/Disabled.
- estts::Status write_ant_release_config (const std::string &ant_config)

  Configure the UHF Transceiver antenna release configuration.
- estts::Status read_ant_release_config (std::string &ant_config)

  Read the UHF Transceiver's current antenna release configuration.
- estts::Status write_low_pwr_mode ()

  Toggle the low power mode of the UHF Transceiver.
- estts::Status read_low_pwr_mode (std::string &mode)

  Read the current power mode of the UHF Transceiver.
- estts::Status write_src_call_sign (const std::string &call_sign="XX0UHF")

  Write a new source call sign for the UHF Transceiver.
- estts::Status read_src_call_sign (std::string &call_sign)

  Read the current source call sign of the UHF Transceiver.
- estts::Status read_dvc_payload_size (std::string &payload_size)

  Read the current size of the device payload of the UHF Transceiver.
- estts::Status write_dvc_addr_config (const std::string &new_addr="22")

  Write a new device address for the UHF Transceiver.
- estts::Status write_radio_trans_prop_config (const std::string &prop_group, const std::string &bytes, const std::string &offset, const std::string &data)

  Write a new configuration for the Radio Transceiver Property of the UHF Transceiver.
- estts::Status read_radio_trans_prop_config (const std::string &prop_group, const std::string &bytes, const std::string &offset, std::string &data)

  Read the current configuration of the Radio Transceiver Property of the UHF Transceiver.
- estts::Status update_firmware (const std::string &all_lines)

  Fully update the firmware of the UHF Transceiver.
- estts::Status update_firmware_sequence (const std::string &one_line)

  Update the firmware of the UHF Transceiver.
- esttc ()

  esttc default constructor that initializes serial_handler
- ~esttc ()

Static Public Member Functions

- static std::string calculate_crc32 (std::string string)

  Calculates CRC32 of command string specified by EnduroSat.

Additional Inherited Members

6.24.1 Detailed Description

Definition at line 11 of file esttc.h.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 esttc()

```
esttc::esttc ( )
```

esttc default constructor that initializes serial_handler

Returns

None

Definition at line 24 of file esttc.cpp.

6.24.2.2 ~esttc()

```
esttc::~esttc ( )
```

Definition at line 864 of file esttc.cpp.

6.24.3 Member Function Documentation

6.24.3.1 calculate_crc32()

```
std::string esttc::calculate_crc32 (
            std::string string )  [static]
```

Calculates CRC32 of command string specified by EnduroSat.

Parameters

string	String input to calculate CRC32

Returns

Calculated CRC32 of inputted string

Definition at line 859 of file esttc.cpp.

6.24.3.2 default_mode()

```
estts::Status esttc::default_mode ( )
```

Enables the default value of every SCW field on Endurosat UHF Transceiver module.

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 34 of file esttc.cpp.

6.24.3.3 read_ant_release_config()

```
estts::Status esttc::read_ant_release_config (
            std::string & ant_config )
```

Read the UHF Transceiver's current antenna release configuration.

Parameters

ant_config	The current UHF Antenna Release Configuration in HEX

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 465 of file esttc.cpp.

6.24.3.4 read_bcn_trans_period()

```
estts::Status esttc::read_bcn_trans_period (
            std::string & RSSI,
            std::string & period )
```

Read the beacon message transmission period from the UHF Transceiver

Parameters

RSSI	Received Signal Strength Indicator
period	The current beacon message transmission period

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 261 of file esttc.cpp.

6.24.3.5 read_config_ax25_decode()

```
estts::Status esttc::read_config_ax25_decode (
            std::string & config_bit )
```

Get the configuration bit that determines whether the UHF Transceiver's Auto AX.25 Decode is Enabled/Disabled.

Parameters

config_bit	The bit ('1' or '0') that determines if automatic ax.25 decoding is enabled

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 421 of file esttc.cpp.

6.24.3.6 read_dvc_payload_size()

```
estts::Status esttc::read_dvc_payload_size (
            std::string & payload_size )
```

Read the current size of the device payload of the UHF Transceiver.

Parameters

payload_size	The size of the payload in HEX

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 575 of file esttc.cpp.

6.24.3.7 read_i2c_resist_config()

```
estts::Status esttc::read_i2c_resist_config (
            std::string & selected_resistor )
```

Read the current I2C Pull-Up resistor configuration of the UHF Transceiver.

Parameters

selected_resistor	The currently selected I2C pull-up resistor configuration in HEX

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 332 of file esttc.cpp.

6.24.3.8 read_low_pwr_mode()

```
estts::Status esttc::read_low_pwr_mode (
            std::string & mode )
```

Read the current power mode of the UHF Transceiver.

Parameters

power_mode	The current power mode of the UHF Transceiver ("00" [Normal Mode] or "01" [Low Power Mode])

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 508 of file esttc.cpp.

6.24.3.9 read_radio_crc16()

```
estts::Status esttc::read_radio_crc16 (
            std::string & mode )
```

Read whether radio packet CRC16 is enabled or disabled for the UHF Transceiver.

Parameters

mode	A bit ('0', or '1') that determines if radio packet CRC16 is enabled or disabled

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 376 of file esttc.cpp.

6.24.3.10 read_radio_freq()

```
estts::Status esttc::read_radio_freq (
            std::string & RSSI,
            std::string & frac,
            std::string & div )
```

Get the radio frequency and the last RSSI (Received signal strength indication) of the UHF Transceiver module.

Parameters

RSSI	Received Signal Strength Indicator
frac	Fractional part of the radio PLL synthesizer in HEX format
div	Integer divider of the radio PLL synthesizer in HEX format

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 138 of file esttc.cpp.

6.24.3.11 read_radio_trans_prop_config()

```
estts::Status esttc::read_radio_trans_prop_config (
            const std::string & prop_group,
            const std::string & bytes,
            const std::string & offset,
            std::string & data )
```

Read the current configuration of the Radio Transceiver Property of the UHF Transceiver.

Parameters

prop_group	The current property group in HEX
bytes	The current number of bytes in HEX
offset	The current start offset of the property content in HEX
data	The current variable size data in HEX

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 662 of file esttc.cpp.

6.24.3.12 read_scw()

```
estts::Status esttc::read_scw (
            std::string & RSSI,
            std::string & dvc_addr,
            std::string & rst_ctr,
            std::string & scw )
```

Send a command to read the current status of the Transceiver given its current Status Control Word (SCW)

Parameters

RSSI	Received Signal Strength Indicator
dvc_addr	The device address in HEX format
rst_ctr	Reset Counter - Counts number of times the device has been reset
scw	The current Status Control Word

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 87 of file esttc.cpp.

6.24.3.13 read_src_call_sign()

```
estts::Status esttc::read_src_call_sign (
            std::string & call_sign )
```

Read the current source call sign of the UHF Transceiver.

Parameters

call_sign	The current source call sign (ex. "XX0UHF")

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 552 of file esttc.cpp.

6.24.3.14 read_trans_pckts()

```
estts::Status esttc::read_trans_pckts (
            std::string & RSSI,
            std::string & pckt_num )
```

Get the number of transmitted packets from the UHF Transceiver.

Parameters

RSSI	Received Signal Strength Indicator
pckt_num	The number of transmitted packets in HEX format

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 189 of file esttc.cpp.

6.24.3.15 read_trans_pckts_crc()

```
estts::Status esttc::read_trans_pckts_crc (
            std::string & RSSI,
            std::string & pckt_num )
```

Get the number of transmitted packets that contained CRC errors from the UHF Transceiver.

Parameters

RSSI	Received Signal Strength Indicator
pckt_num	The number of transmitted packets with CRC errors in HEX format

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 214 of file esttc.cpp.

6.24.3.16 read_uptime()

```
estts::Status esttc::read_uptime (
            std::string & RSSI,
            std::string & uptime )
```

Get the uptime of the UHF Transceiver.

Parameters

RSSI	Received Signal Strength Indicator
uptime	Uptime value in seconds in HEX format

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 164 of file esttc.cpp.

6.24.3.17 update_firmware()

```
estts::Status esttc::update_firmware (
            const std::string & all_lines )
```

Fully update the firmware of the UHF Transceiver.

Parameters

all_lines	The variable length data which comprises of all lines of the .SCRM file separated by new line characters '\n'

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 702 of file esttc.cpp.

6.24.3.18 update_firmware_sequence()

```
estts::Status esttc::update_firmware_sequence (
            const std::string & one_line )
```

Update the firmware of the UHF Transceiver.

Parameters

one_line	The variable length data which comprises of a single line of the .SCRM file

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 757 of file esttc.cpp.

6.24.3.19 write_ant_release_config()

```
estts::Status esttc::write_ant_release_config (
            const std::string & ant_config )
```

Configure the UHF Transceiver antenna release configuration.

Parameters

ant_config	UHF Antenna Release Configuration (4 chars)

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 444 of file esttc.cpp.

6.24.3.20 write_bcn_trans_period()

```
estts::Status esttc::write_bcn_trans_period (
            const std::string & period = "003C" )
```

Configure the beacon message transmission period of the UHF Transceiver.

Parameters

period	The desired period (in seconds in HEX format) to set

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 238 of file esttc.cpp.

6.24.3.21 write_config_ax25_decode()

```
estts::Status esttc::write_config_ax25_decode (
            const std::string & config_bit )
```

Write the configuration bit that determines whether the UHF Transceiver's Auto AX.25 Decode is Enabled/Disabled.

Parameters

config_bit	The bit ('1' or '0') that determines if automatic ax.25 decoding is enabled

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 399 of file esttc.cpp.

6.24.3.22 write_dvc_addr_config()

```
estts::Status esttc::write_dvc_addr_config (
            const std::string & new_addr = "22" )
```

Write a new device address for the UHF Transceiver.

Parameters

new_addr	The new device address for the UHF Transceiver in HEX ("22" or "23")

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 598 of file esttc.cpp.

6.24.3.23 write_i2c_resist_config()

```
estts::Status esttc::write_i2c_resist_config (
            const std::string & resistor_config )
```

Configure the I2C Pull-Up resistor configuration of the UHF Transceiver.

Parameters

resistor_config	The desired I2C pull-up resistor configuration in HEX

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 311 of file esttc.cpp.

6.24.3.24 write_low_pwr_mode()

```
estts::Status esttc::write_low_pwr_mode ( )
```

Toggle the low power mode of the UHF Transceiver.

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 487 of file esttc.cpp.

6.24.3.25 write_radio_crc16()

```
estts::Status esttc::write_radio_crc16 (
            const std::string & mode )
```

Enable or disable the UHF Transceiver's radio packet CRC16.

Parameters

mode	A bit ('0', or '1') that determines if radio packet CRC16 is enabled or disabled

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 355 of file esttc.cpp.

6.24.3.26 write_radio_freq_config()

```
estts::Status esttc::write_radio_freq_config (
            const std::string & frac = "76620F41",
            const std::string & div = "41" )
```

6.24.3.27 write_radio_trans_prop_config()

```
estts::Status esttc::write_radio_trans_prop_config (
            const std::string & prop_group,
            const std::string & bytes,
            const std::string & offset,
            const std::string & data )
```

Write a new configuration for the Radio Transceiver Property of the UHF Transceiver.

Parameters

prop_group	The property group in HEX
bytes	The number of bytes in HEX
offset	The start offset of the property content in HEX
data	The variable size data in HEX

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 624 of file esttc.cpp.

6.24.3.28 write_res_default_vals()

```
estts::Status esttc::write_res_default_vals ( )
```

restore default values to UHF Transceiver for Destination/Source/Morse code call sign, Audio beacon period and message, Text beadon period, and Pipe timeout period

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 288 of file esttc.cpp.

6.24.3.29 write_scw()

```
estts::Status esttc::write_scw (
            uint16_t scw_command )
```

Sends a command with a Status Control Word (SCW) that changes the Endurosat UHF Transceiver's settings.

Parameters

scw_command	A command to change the SCW to a different configuration (e.g. enable_pipe)

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 43 of file esttc.cpp.

6.24.3.30 write_src_call_sign()

```
estts::Status esttc::write_src_call_sign (
            const std::string & call_sign = "XX0UHF" )
```

Write a new source call sign for the UHF Transceiver.

Parameters

call_sign	The new source call sign you want to replace the old one with (ex. "XX0UHF")

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 531 of file esttc.cpp.

The documentation for this class was generated from the following files:

- src/ti/esttc.h
- src/ti/esttc.cpp

6.25 estts::endurosat::esttc_const Class Reference

`#include <constants.h>`

Public Types

- enum SCW_Commands { default_mode , enable_pipe , scw_stopper }

Public Attributes

- const uint8_t NUM_OF_RETRIES = 5
- const char * HEADER = "ES+"
- const char METHOD_READ = 'R'
- const char METHOD_WRITE = 'W'
- const char METHOD_FIRMWARE_UPDATE = 'D'
- const char * ADDRESS = "22"
- const char * BLANK = " "
- const char * END = "\r"
- const char * DOWNLINK_XOR = "AB7563CD"
- const char * UPLINK_XOR = "6ACD3B57"
- const char * CMD_SCW = "00"
- const char * CMD_RADIO_FREQ_CONFIG = "01"
- const char * CMD_READ_UPTIME = "02"
- const char * CMD_READ_TRANS_PCKTS = "03"
- const char * CMD_READ_RECEIV_PCKTS = "04"
- const char * CMD_READ_TRANS_PCKTS_CRC = "05"
- const char * CMD_PIPE_MODE_TMOUT_CONFIG = "06"
- const char * CMD_BCN_MSG_TRANS_CONFIG = "07"
- const char * CMD_AUDIO_BCN_P_TRANS = "08"
- const char * CMD_RESTORE = "09"
- const char * CMD_TEMP_VAL = "0A"
- const char * CMD_I2C_RESIST_CONFIG = "0B"
- const char * CMD_TERM_RESIST_CONFIG = "EC"
- const char * CMD_ENABLE_DISABLE_RADIO_CRC = "ED"
- const char * CMD_FORCE_BCN_CMD = "EE"
- const char * CMD_AUTO_AX25_DECODE = "EF"
- const char * CMD_READ_WRITE_I2C = "F1"
- const char * CMD_ANT_RELEASE_CONFIG = "F2"
- const char * CMD_ANT_READ_WRITE = "F3"
- const char * CMD_LOW_PWR_MODE = "F4"
- const char * CMD_DEST_CALL_SIGN = "F5"
- const char * CMD_SRC_CALL_SIGN = "F6"
- const char * CMD_READ_SFTWR_VER = "F9"
- const char * CMD_READ_DVC_PAYLOAD = "FA"
- const char * CMD_BCN_MSG_CONFIG = "FB"
- const char * CMD_DVC_ADDR_CONFIG = "FC"
- const char * CMD_FRAM_MEM_READ_WRITE = "FD"
- const char * CMD_RADIO_TRANS_PROP_CONFIG = "FE"
- const char * CMD_SECURE_MODE = "FF"
- const char * CMD_FRMWR_UPDATE = "AA"
- const char * scw_body [scw_stopper]

6.25.1 Detailed Description

Definition at line 205 of file constants.h.

6.25.2 Member Enumeration Documentation

6.25.2.1 SCW_Commands

```
enum estts::endurosat::esttc_const::SCW_Commands
```

Enumerator

default_mode	
enable_pipe	
scw_stopper	

Definition at line 248 of file constants.h.

6.25.3 Member Data Documentation

6.25.3.1 ADDRESS

```
const char* estts::endurosat::esttc_const::ADDRESS = "22"
```

Definition at line 212 of file constants.h.

6.25.3.2 BLANK

```
const char* estts::endurosat::esttc_const::BLANK = " "
```

Definition at line 213 of file constants.h.

6.25.3.3 CMD_ANT_READ_WRITE

```
const char* estts::endurosat::esttc_const::CMD_ANT_READ_WRITE = "F3"
```

Definition at line 235 of file constants.h.

6.25.3.4 CMD_ANT_RELEASE_CONFIG

```
const char* estts::endurosat::esttc_const::CMD_ANT_RELEASE_CONFIG = "F2"
```

Definition at line 234 of file constants.h.

6.25.3.5 CMD_AUDIO_BCN_P_TRANS

```
const char* estts::endurosat::esttc_const::CMD_AUDIO_BCN_P_TRANS = "08"
```

Definition at line 225 of file constants.h.

6.25.3.6 CMD_AUTO_AX25_DECODE

```
const char* estts::endurosat::esttc_const::CMD_AUTO_AX25_DECODE = "EF"
```

Definition at line 232 of file constants.h.

6.25.3.7 CMD_BCN_MSG_CONFIG

```
const char* estts::endurosat::esttc_const::CMD_BCN_MSG_CONFIG = "FB"
```

Definition at line 241 of file constants.h.

6.25.3.8 CMD_BCN_MSG_TRANS_CONFIG

```
const char* estts::endurosat::esttc_const::CMD_BCN_MSG_TRANS_CONFIG = "07"
```

Definition at line 224 of file constants.h.

6.25.3.9 CMD_DEST_CALL_SIGN

```
const char* estts::endurosat::esttc_const::CMD_DEST_CALL_SIGN = "F5"
```

Definition at line 237 of file constants.h.

6.25.3.10 CMD_DVC_ADDR_CONFIG

```
const char* estts::endurosat::esttc_const::CMD_DVC_ADDR_CONFIG = "FC"
```

Definition at line 242 of file constants.h.

6.25.3.11 CMD_ENABLE_DISABLE_RADIO_CRC

```
const char* estts::endurosat::esttc_const::CMD_ENABLE_DISABLE_RADIO_CRC = "ED"
```

Definition at line 230 of file constants.h.

6.25.3.12 CMD_FORCE_BCN_CMD

```
const char* estts::endurosat::esttc_const::CMD_FORCE_BCN_CMD = "EE"
```

Definition at line 231 of file constants.h.

6.25.3.13 CMD_FRAM_MEM_READ_WRITE

```
const char* estts::endurosat::esttc_const::CMD_FRAM_MEM_READ_WRITE = "FD"
```

Definition at line 243 of file constants.h.

6.25.3.14 CMD_FRMWR_UPDATE

```
const char* estts::endurosat::esttc_const::CMD_FRMWR_UPDATE = "AA"
```

Definition at line 246 of file constants.h.

6.25.3.15 CMD_I2C_RESIST_CONFIG

```
const char* estts::endurosat::esttc_const::CMD_I2C_RESIST_CONFIG = "0B"
```

Definition at line 228 of file constants.h.

6.25.3.16 CMD_LOW_PWR_MODE

```
const char* estts::endurosat::esttc_const::CMD_LOW_PWR_MODE = "F4"
```

Definition at line 236 of file constants.h.

6.25.3.17 CMD_PIPE_MODE_TMOUT_CONFIG

```
const char* estts::endurosat::esttc_const::CMD_PIPE_MODE_TMOUT_CONFIG = "06"
```

Definition at line 223 of file constants.h.

6.25.3.18 CMD_RADIO_FREQ_CONFIG

```
const char* estts::endurosat::esttc_const::CMD_RADIO_FREQ_CONFIG = "01"
```

Definition at line 218 of file constants.h.

6.25.3.19 CMD_RADIO_TRANS_PROP_CONFIG

```
const char* estts::endurosat::esttc_const::CMD_RADIO_TRANS_PROP_CONFIG = "FE"
```

Definition at line 244 of file constants.h.

6.25.3.20 CMD_READ_DVC_PAYLOAD

```
const char* estts::endurosat::esttc_const::CMD_READ_DVC_PAYLOAD = "FA"
```

Definition at line 240 of file constants.h.

6.25.3.21 CMD_READ_RECEIV_PCKTS

```
const char* estts::endurosat::esttc_const::CMD_READ_RECEIV_PCKTS = "04"
```

Definition at line 221 of file constants.h.

6.25.3.22 CMD_READ_SFTWR_VER

```
const char* estts::endurosat::esttc_const::CMD_READ_SFTWR_VER = "F9"
```

Definition at line 239 of file constants.h.

6.25.3.23 CMD_READ_TRANS_PCKTS

```
const char* estts::endurosat::esttc_const::CMD_READ_TRANS_PCKTS = "03"
```

Definition at line 220 of file constants.h.

6.25.3.24 CMD_READ_TRANS_PCKTS_CRC

```
const char* estts::endurosat::esttc_const::CMD_READ_TRANS_PCKTS_CRC = "05"
```

Definition at line 222 of file constants.h.

6.25.3.25 CMD_READ_UPTIME

```
const char* estts::endurosat::esttc_const::CMD_READ_UPTIME = "02"
```

Definition at line 219 of file constants.h.

6.25.3.26 CMD_READ_WRITE_I2C

```
const char* estts::endurosat::esttc_const::CMD_READ_WRITE_I2C = "F1"
```

Definition at line 233 of file constants.h.

6.25.3.27 CMD_RESTORE

```
const char* estts::endurosat::esttc_const::CMD_RESTORE = "09"
```

Definition at line 226 of file constants.h.

6.25.3.28 CMD_SCW

```
const char* estts::endurosat::esttc_const::CMD_SCW = "00"
```

Definition at line 217 of file constants.h.

6.25.3.29 CMD_SECURE_MODE

```
const char* estts::endurosat::esttc_const::CMD_SECURE_MODE = "FF"
```

Definition at line 245 of file constants.h.

6.25.3.30 CMD_SRC_CALL_SIGN

```
const char* estts::endurosat::esttc_const::CMD_SRC_CALL_SIGN = "F6"
```

Definition at line 238 of file constants.h.

6.25.3.31 CMD_TEMP_VAL

```
const char* estts::endurosat::esttc_const::CMD_TEMP_VAL = "0A"
```

Definition at line 227 of file constants.h.

6.25.3.32 CMD_TERM_RESIST_CONFIG

```
const char* estts::endurosat::esttc_const::CMD_TERM_RESIST_CONFIG = "EC"
```

Definition at line 229 of file constants.h.

6.25.3.33 DOWNLINK_XOR

```
const char* estts::endurosat::esttc_const::DOWNLINK_XOR = "AB7563CD"
```

Definition at line 215 of file constants.h.

6.25.3.34 END

```
const char* estts::endurosat::esttc_const::END = "\r"
```

Definition at line 214 of file constants.h.

6.25.3.35 HEADER

```
const char* estts::endurosat::esttc_const::HEADER = "ES+"
```

Definition at line 208 of file constants.h.

6.25.3.36 METHOD_FIRMWARE_UPDATE

```
const char estts::endurosat::esttc_const::METHOD_FIRMWARE_UPDATE = 'D'
```

Definition at line 211 of file constants.h.

6.25.3.37 METHOD_READ

```
const char estts::endurosat::esttc_const::METHOD_READ = 'R'
```

Definition at line 209 of file constants.h.

6.25.3.38 METHOD_WRITE

```
const char estts::endurosat::esttc_const::METHOD_WRITE = 'W'
```

Definition at line 210 of file constants.h.

6.25.3.39 NUM_OF_RETRIES

```
const uint8_t estts::endurosat::esttc_const::NUM_OF_RETRIES = 5
```

Definition at line 207 of file constants.h.

6.25.3.40 scw_body

```
const char* estts::endurosat::esttc_const::scw_body[scw_stopper]
```

Initial value:

```
= {
                    "4343",
                    "3323"
            }
```

Definition at line 254 of file constants.h.

6.25.3.41 UPLINK_XOR

```
const char* estts::endurosat::esttc_const::UPLINK_XOR = "6ACD3B57"
```

Definition at line 216 of file constants.h.

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

- src/utils/constants.h

6.26 groundstation_cmdtelem_manager Class Reference

```
#include <groundstation_cmdtelem_manager.h>
```

Public Member Functions

- groundstation_cmdtelem_manager (transmission_interface *ti, std::function< estts::Status(std::string)> telem_callback)

  Default constructor for ground station command/telemetry manager. Takes argument for a transmission interface and initializes the command handler. Note that in order to run the dispatcher, dispatcher_init must be called.
- std::string schedule_command (std::string command, const std::function< estts::Status(std::string)> &callback)

6.26.1 Detailed Description

Definition at line 15 of file groundstation_cmdtelem_manager.h.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 groundstation_cmdtelem_manager()

```
groundstation_cmdtelem_manager::groundstation_cmdtelem_manager (
            transmission_interface * ti,
            std::function< estts::Status(std::string)> telem_callback )
```

Default constructor for ground station command/telemetry manager. Takes argument for a transmission interface and initializes the command handler. Note that in order to run the dispatcher, dispatcher_init must be called.

Definition at line 26 of file groundstation_cmdtelem_manager.cpp.

6.26.3 Member Function Documentation

6.26.3.1 schedule_command()

```
std::string groundstation_cmdtelem_manager::schedule_command (
            std::string command,
            const std::function< estts::Status(std::string)> & callback )
```

Function that takes argument for a string command passed by a higher level interface, and a callback function that is called with the response to the command as argument. It's implied that this function should handle whatever telemetry is returned by the dispatch process. This function creates a waiting_command object and stores a serial number, the callback function, and the command string expected to dispatch during the next satellite pass.

Parameters

command	String command received by higher level interface
callback	Callback with form std::function<estts::Status(std::string)>

Returns

String serial number associated with newly scheduled command

Definition at line 8 of file groundstation_cmdtelem_manager.cpp.

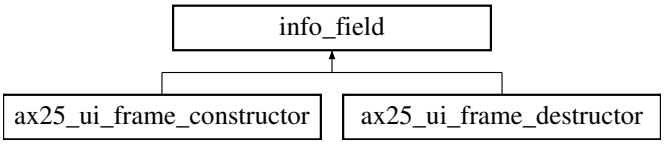
The documentation for this class was generated from the following files:

- src/fapi/groundstation_cmdtelem_manager.h
- src/fapi/groundstation_cmdtelem_manager.cpp

6.27 info_field Class Reference

`#include <info_field.h>`

Inheritance diagram for info_field:



Protected Member Functions

- std::string build_info_field ()
- estts::telemetry_object * build_telemetry_object (std::string info_field)
- info_field ()
- info_field (estts::command_object *esttsCommand)

Protected Attributes

- estts::command_object * command

6.27.1 Detailed Description

Definition at line 14 of file info_field.h.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 info_field() [1/2]

```
info_field::info_field ( )  [inline], [explicit], [protected]
```

Definition at line 37 of file info_field.h.

6.27.2.2 info_field() [2/2]

```
info_field::info_field (
            estts::command_object * esttsCommand )  [inline], [explicit], [protected]
```

Definition at line 39 of file info_field.h.

6.27.3 Member Function Documentation

6.27.3.1 build_info_field()

```
string info_field::build_info_field ( )  [protected]
```

@description Retrieves the encodings of the entire AX.25 Information Field

Returns

string of bits (e.g. "10010101...")

Definition at line 84 of file info_field.cpp.

6.27.3.2 build_telemetry_object()

```
estts::telemetry_object * info_field::build_telemetry_object (
            std::string info_field )  [protected]
```

Definition at line 92 of file info_field.cpp.

6.27.4 Member Data Documentation

6.27.4.1 command

```
estts::command_object* info_field::command  [protected]
```

Definition at line 30 of file info_field.h.

The documentation for this class was generated from the following files:

- src/tnc_emulator/info_field.h
- src/tnc_emulator/info_field.cpp

6.28 mde_command Class Reference

```
#include <mde_command.h>
```

6.28.1 Detailed Description

Definition at line 9 of file mde_command.h.

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

- src/fapi/command_handler/mde_command.h

6.29 obc_command Class Reference

```
#include <obc_command.h>
```

6.29.1 Detailed Description

Definition at line 9 of file obc_command.h.

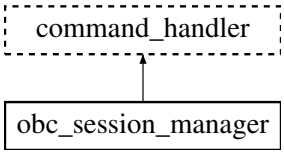
The documentation for this class was generated from the following file:

- src/fapi/command_handler/obc_command.h

6.30 obc_session_manager Class Reference

`#include <obc_session_manager.h>`

Inheritance diagram for obc_session_manager:



Public Member Functions

- obc_session_manager (transmission_interface *ti)

  Default constructor for command scheduler. Takes argument for a transmission interface and initializes the command handler. Note that in order to run the dispatcher, dispatcher_init must be called.
- obc_session_manager (transmission_interface *ti, std::function< estts::Status(std::string)> telem_callback)
- ~obc_session_manager ()

  Cleans up internal structures.
- std::string schedule_command (std::string command, const std::function< estts::Status(std::string)> &callback)
- std::string schedule_command (estts::command_object *command, std::function< estts::Status(std::vector< estts::telemetry_object * >)> decomp_callback)

  Function that takes argument to a vector of command objects and a obj_callback function expecting a pointer to a telemetry object as argument. It's implied that this function should handle whatever telemetry is returned by the dispatch process. This function creates a waiting_command object and stores a serial number, the obj_callback function, and the command frames (as command_objects) expected to dispatch during the next satellite pass.
- void await_completion ()

  Uses std::thread::join() to await thread completion. If commands continue to be added to queue, this function will block indefinitely.

Additional Inherited Members

6.30.1 Detailed Description

Definition at line 28 of file obc_session_manager.h.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 obc_session_manager() [1/2]

```
obc_session_manager::obc_session_manager (
            transmission_interface * ti )  [explicit]
```

Default constructor for command scheduler. Takes argument for a transmission interface and initializes the command handler. Note that in order to run the dispatcher, dispatcher_init must be called.

Definition at line 49 of file obc_session_manager.cpp.

6.30.2.2 obc_session_manager() [2/2]

```
obc_session_manager::obc_session_manager (
            transmission_interface * ti,
            std::function< estts::Status(std::string)> telem_callback )  [explicit]
```

Secondary constructor for command scheduler that takes argument for a telemetry callback handler that is called when lower layers receive telemetry.

Parameters

telem_callback	Callback with form std::function<estts::Status(std::string)>

Definition at line 55 of file obc_session_manager.cpp.

6.30.2.3 ~obc_session_manager()

```
obc_session_manager::~obc_session_manager ( )
```

Cleans up internal structures.

Definition at line 72 of file obc_session_manager.cpp.

6.30.3 Member Function Documentation

6.30.3.1 await_completion()

```
void obc_session_manager::await_completion ( )
```

Uses std::thread::join() to await thread completion. If commands continue to be added to queue, this function will block indefinitely.

Definition at line 65 of file obc_session_manager.cpp.

6.30.3.2 schedule_command() [1/2]

```
std::string obc_session_manager::schedule_command (
            estts::command_object * command,
            std::function< estts::Status(std::vector< estts::telemetry_object * >)> decomp↩
_callback )
```

Function that takes argument to a vector of command objects and a obj_callback function expecting a pointer to a telemetry object as argument. It's implied that this function should handle whatever telemetry is returned by the dispatch process. This function creates a waiting_command object and stores a serial number, the obj_callback function, and the command frames (as command_objects) expected to dispatch during the next satellite pass.

Parameters

command	Vector of command_object pointers to schedule for dispatching.
decomp_callback	Callback pointer to a function that returns a Status and takes argument for a vector of telemetry_objects. It's implied that this obj_callback function knows how to decode a vector of telemetry frames.

Returns

Returns a unique string serial number for later retrieval of the command status.

Definition at line 13 of file obc_session_manager.cpp.

6.30.3.3 schedule_command() [2/2]

```
std::string obc_session_manager::schedule_command (
            std::string command,
            const std::function< estts::Status(std::string)> & callback )
```

Function that takes argument for a string command passed by a higher level interface, and a callback function that is called with the response to the command as argument. It's implied that this function should handle whatever telemetry is returned by the dispatch process. This function creates a waiting_command object and stores a serial number, the callback function, and the command string expected to dispatch during the next satellite pass.

Parameters

command	String command received by higher level interface
callback	Callback with form std::function<estts::Status(std::string)>

Returns

String serial number associated with newly scheduled command

Definition at line 31 of file obc_session_manager.cpp.

The documentation for this class was generated from the following files:

- src/fapi/obc_session_manager.h
- src/fapi/obc_session_manager.cpp

6.31 posix_serial Class Reference

```
#include <posix_serial.h>
```

Protected Member Functions

- posix_serial (const char *port, int baud)

  Base constructor that initializes port and baud, opens specified port as serial port, and configures it using Terminos.
- ~posix_serial ()
- virtual ssize_t write_serial_uc (unsigned char *data, int size) const

  Takes argument for a pointer to an unsigned char and transmits it across the open serial port.
- virtual unsigned char * read_serial_uc ()

  Reads available data from open serial port.
- virtual estts::Status write_serial_s (const std::string &data) const

  Writes string to open serial port.
- virtual std::string read_serial_s ()

  Reads available data from serial port and returns data as string.
- virtual void clear_serial_fifo ()
- virtual int check_serial_bytes_avail () const

Protected Attributes

- std::stringstream cache

6.31.1 Detailed Description

Definition at line 12 of file posix_serial.h.

6.31.2 Constructor & Destructor Documentation

6.31.2.1 posix_serial()

```
posix_serial::posix_serial (
            const char * port,
            int baud )  [protected]
```

Base constructor that initializes port and baud, opens specified port as serial port, and configures it using Terminos.

Parameters

port	Serial port (EX "/dev/cu.usbmodem")
baud	Serial baud rate (EX 115200)

Returns

None

Definition at line 27 of file posix_serial.cpp.

6.31.2.2 ~posix_serial()

```
posix_serial::~posix_serial ( )  [protected]
```

Definition at line 268 of file posix_serial.cpp.

6.31.3 Member Function Documentation

6.31.3.1 check_serial_bytes_avail()

```
int posix_serial::check_serial_bytes_avail ( ) const  [protected], [virtual]
```

Definition at line 278 of file posix_serial.cpp.

6.31.3.2 clear_serial_fifo()

```
void posix_serial::clear_serial_fifo ( )  [protected], [virtual]
```

Definition at line 272 of file posix_serial.cpp.

6.31.3.3 read_serial_s()

```
std::string posix_serial::read_serial_s ( )  [protected], [virtual]
```

Reads available data from serial port and returns data as string.

Returns

Returns translated string of received data

Definition at line 249 of file posix_serial.cpp.

6.31.3.4 read_serial_uc()

```
unsigned char * posix_serial::read_serial_uc ( )  [protected], [virtual]
```

Reads available data from open serial port.

Returns

Returns nullptr if nothing was read, or the data read from serial port if read was successful (and data was available).

CRITICAL NOTE: delete MUST be called when done with the value returned. If this is not done, a memory leak will be created. To avoid this issue, use read_serial_s

Definition at line 199 of file posix_serial.cpp.

6.31.3.5 write_serial_s()

```
estts::Status posix_serial::write_serial_s (
            const std::string & data ) const  [protected], [virtual]
```

Writes string to open serial port.

Parameters

data	String argument

Returns

Number of bytes transferred across open serial port

Definition at line 235 of file posix_serial.cpp.

6.31.3.6 write_serial_uc()

```
ssize_t posix_serial::write_serial_uc (
            unsigned char * data,
            int size ) const  [protected], [virtual]
```

Takes argument for a pointer to an unsigned char and transmits it across the open serial port.

Parameters

data	Unsigned char * containing bytes to be written
size	Size of data being transmitted

Returns

Returns -1 if write failed, or the number of bytes written if call succeeded

Definition at line 169 of file posix_serial.cpp.

6.31.4 Member Data Documentation

6.31.4.1 cache

```
std::stringstream posix_serial::cache  [protected]
```

Definition at line 26 of file posix_serial.h.

The documentation for this class was generated from the following files:

- src/ti/posix_serial.h
- src/ti/posix_serial.cpp

6.32 satellite_txvr_cmdtelem_manager Class Reference

`#include <satellite_txvr_cmdtelem_manager.h>`

Public Member Functions

- satellite_txvr_cmdtelem_manager (transmission_interface *ti, std::function< estts::Status(std::string)> telem_callback)

  Default constructor for satellite transceiver command/telemetry manager. Takes argument for a transmission interface and initializes the command handler. Note that in order to run the dispatcher, dispatcher_init must be called.
- std::string schedule_command (std::string command, const std::function< estts::Status(std::string)> &callback)

6.32.1 Detailed Description

Definition at line 15 of file satellite_txvr_cmdtelem_manager.h.

6.32.2 Constructor & Destructor Documentation

6.32.2.1 satellite_txvr_cmdtelem_manager()

```
satellite_txvr_cmdtelem_manager::satellite_txvr_cmdtelem_manager (
            transmission_interface * ti,
            std::function< estts::Status(std::string)> telem_callback )
```

Default constructor for satellite transceiver command/telemetry manager. Takes argument for a transmission interface and initializes the command handler. Note that in order to run the dispatcher, dispatcher_init must be called.

Definition at line 8 of file satellite_txvr_cmdtelem_manager.cpp.

6.32.3 Member Function Documentation

6.32.3.1 schedule_command()

```
std::string satellite_txvr_cmdtelem_manager::schedule_command (
            std::string command,
            const std::function< estts::Status(std::string)> & callback )
```

Function that takes argument for a string command passed by a higher level interface, and a callback function that is called with the response to the command as argument. It's implied that this function should handle whatever telemetry is returned by the dispatch process. This function creates a waiting_command object and stores a serial number, the callback function, and the command string expected to dispatch during the next satellite pass.

Parameters

command	String command received by higher level interface
callback	Callback with form std::function<estts::Status(std::string)>

Returns

String serial number associated with newly scheduled command

Definition at line 19 of file satellite_txvr_cmdtelem_manager.cpp.

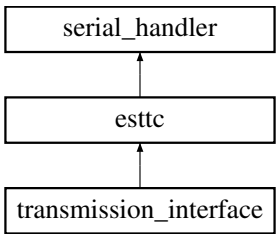
The documentation for this class was generated from the following files:

- src/fapi/satellite_txvr_cmdtelem_manager.h
- src/fapi/satellite_txvr_cmdtelem_manager.cpp

6.33 serial_handler Class Reference

`#include <serial_handler.h>`

Inheritance diagram for serial_handler:



Public Member Functions

- void read_serial_async (const std::function< estts::Status(char *, size_t)> &cb)

Protected Member Functions

- serial_handler ()

  Base constructor that initializes port and baud, opens specified port as serial port, and configures it using Termios.
- estts::Status initialize_serial_port ()

  Initializes serial terminal port using Termios and Boost.
- ~serial_handler ()
- size_t write_serial_uc (unsigned char *data, int size)

  Takes argument for a pointer to an unsigned char and transmits it across the open serial port.
- unsigned char * read_serial_uc ()

  Reads available data from open serial port.
- estts::Status write_serial_s (const std::string &data)

  Writes string to open serial port.
- std::string read_serial_s ()

  Reads available data from serial port and returns data as string.
- unsigned char * read_serial_uc (int bytes)
- std::string read_serial_s (int bytes)
- std::function< void(boost::system::error_code, size_t)> get_generic_async_read_lambda (const std::function< estts::Status(char *, size_t)> &estts_callback)
- void clear_serial_fifo ()
- void clear_serial_fifo (const std::function< estts::Status(std::string)> &cb)
- int check_serial_bytes_avail ()

Protected Attributes

- std::stringstream cache
- char async_buf [MAX_SERIAL_READ]

6.33.1 Detailed Description

Definition at line 14 of file serial_handler.h.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 serial_handler()

```
serial_handler::serial_handler ( )  [protected]
```

Base constructor that initializes port and baud, opens specified port as serial port, and configures it using Termios.

Returns

None

Definition at line 25 of file serial_handler.cpp.

6.33.2.2 ~serial_handler()

```
serial_handler::~serial_handler ( )  [protected]
```

Definition at line 191 of file serial_handler.cpp.

6.33.3 Member Function Documentation

6.33.3.1 check_serial_bytes_avail()

```
int serial_handler::check_serial_bytes_avail ( )  [protected]
```

Definition at line 207 of file serial_handler.cpp.

6.33.3.2 clear_serial_fifo() [1/2]

```
void serial_handler::clear_serial_fifo ( )  [protected]
```

Definition at line 195 of file serial_handler.cpp.

6.33.3.3 clear_serial_fifo() [2/2]

```
void serial_handler::clear_serial_fifo (
            const std::function< estts::Status(std::string)> & cb )  [protected]
```

Definition at line 201 of file serial_handler.cpp.

6.33.3.4 get_generic_async_read_lambda()

```
std::function< void(boost::system::error_code, size_t)> serial_handler::get_generic_async_↩
read_lambda (
            const std::function< estts::Status(char *, size_t)> & estts_callback )  [protected]
```

Definition at line 220 of file serial_handler.cpp.

6.33.3.5 initialize_serial_port()

```
estts::Status serial_handler::initialize_serial_port ( )  [protected]
```

Initializes serial terminal port using Termios and Boost.

Returns

#ES_OK if port configures successfully, or #ES_UNSUCCESSFUL if not

Definition at line 48 of file serial_handler.cpp.

6.33.3.6 read_serial_async()

```
void serial_handler::read_serial_async (
            const std::function< estts::Status(char *, size_t)> & cb )
```

Definition at line 216 of file serial_handler.cpp.

6.33.3.7 read_serial_s() [1/2]

```
std::string serial_handler::read_serial_s ( )  [protected]
```

Reads available data from serial port and returns data as string.

Returns

Returns translated string of received data

Definition at line 175 of file serial_handler.cpp.

6.33.3.8 read_serial_s() [2/2]

```
std::string serial_handler::read_serial_s (
            int bytes )  [protected]
```

Definition at line 264 of file serial_handler.cpp.

6.33.3.9 read_serial_uc() [1/2]

```
unsigned char * serial_handler::read_serial_uc ( )  [protected]
```

Reads available data from open serial port.

Returns

Returns nullptr if nothing was read, or the data read from serial port if read was successful (and data was available).

CRITICAL NOTE: delete MUST be called when done with the value returned. If this is not done, a memory leak will be created. To avoid this issue, use read_serial_s

Definition at line 130 of file serial_handler.cpp.

6.33.3.10 read_serial_uc() [2/2]

```
unsigned char * serial_handler::read_serial_uc (
            int bytes )  [protected]
```

Definition at line 238 of file serial_handler.cpp.

6.33.3.11 write_serial_s()

```
estts::Status serial_handler::write_serial_s (
            const std::string & data )  [protected]
```

Writes string to open serial port.

Parameters

data	String argument

Returns

Number of bytes transferred across open serial port

Definition at line 159 of file serial_handler.cpp.

6.33.3.12 write_serial_uc()

```
size_t serial_handler::write_serial_uc (
            unsigned char * data,
            int size )  [protected]
```

Takes argument for a pointer to an unsigned char and transmits it across the open serial port.

Parameters

data	Unsigned char * containing bytes to be written
size	Size of data being transmitted

Returns

Returns -1 if write failed, or the number of bytes written if call succeeded

Definition at line 103 of file serial_handler.cpp.

6.33.4 Member Data Documentation

6.33.4.1 async_buf

```
char serial_handler::async_buf[MAX_SERIAL_READ]  [protected]
```

Definition at line 28 of file serial_handler.h.

6.33.4.2 cache

```
std::stringstream serial_handler::cache  [protected]
```

Definition at line 26 of file serial_handler.h.

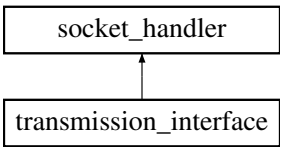
The documentation for this class was generated from the following files:

- src/ti/serial_handler.h
- src/ti/serial_handler.cpp

6.34 socket_handler Class Reference

```
#include <socket_handler.h>
```

Inheritance diagram for socket_handler:



Public Member Functions

- socket_handler (const char *address, int port)

  Base constructor that initializes address and port, opens specified port as socket, and configures it. NOTE:
- ~socket_handler ()
- std::string read_socket_s () const

  Reads available data from socket and returns data as string.
- unsigned char * read_socket_uc () const

  Reads available data from open socket.
- estts::Status write_socket_s (const std::string &data) const

  Writes string to open socket.
- ssize_t write_socket_uc (unsigned char *data, int size) const

  Takes argument for a pointer to an unsigned char and transmits it across the open socket.
- estts::Status init_socket_handle ()

Public Attributes

- int sock
- int port

Protected Member Functions

- int check_sock_bytes_avail () const

6.34.1 Detailed Description

Definition at line 14 of file socket_handler.h.

6.34.2 Constructor & Destructor Documentation

6.34.2.1 socket_handler()

```
socket_handler::socket_handler (
            const char * address,
            int port )
```

Base constructor that initializes address and port, opens specified port as socket, and configures it. NOTE:

Parameters

address	Server address to connect to (EX "127.0.0.1")
port	Server port to connect to (EX 8080)

Returns

None

Definition at line 24 of file socket_handler.cpp.

6.34.2.2 ~socket_handler()

```
socket_handler::~socket_handler ( )
```

Definition at line 183 of file socket_handler.cpp.

6.34.3 Member Function Documentation

6.34.3.1 check_sock_bytes_avail()

```
int socket_handler::check_sock_bytes_avail ( ) const  [protected]
```

Definition at line 162 of file socket_handler.cpp.

6.34.3.2 init_socket_handle()

```
estts::Status socket_handler::init_socket_handle ( )
```

Definition at line 168 of file socket_handler.cpp.

6.34.3.3 read_socket_s()

```
std::string socket_handler::read_socket_s ( ) const
```

Reads available data from socket and returns data as string.

Returns

Returns translated string of received data

Definition at line 145 of file socket_handler.cpp.

6.34.3.4 read_socket_uc()

```
unsigned char * socket_handler::read_socket_uc ( ) const
```

Reads available data from open socket.

Returns

Returns nullptr if nothing was read, or the data read from socket if read was successful (and data was available).

CRITICAL NOTE: delete MUST be called when done with the value returned. If this is not done, a memory leak will be created. To avoid this issue, use read_socket_s

Definition at line 104 of file socket_handler.cpp.

6.34.3.5 write_socket_s()

```
estts::Status socket_handler::write_socket_s (
            const std::string & data ) const
```

Writes string to open socket.

Parameters

data	String argument

Returns

Number of bytes transferred across open socket

Definition at line 131 of file socket_handler.cpp.

6.34.3.6 write_socket_uc()

```
ssize_t socket_handler::write_socket_uc (
            unsigned char * data,
            int size ) const
```

Takes argument for a pointer to an unsigned char and transmits it across the open socket.

Parameters

data	Unsigned char * containing bytes to be written
size	Size of data being transmitted

Returns

Returns -1 if write failed, or the number of bytes written if call succeeded

Definition at line 81 of file socket_handler.cpp.

6.34.4 Member Data Documentation

6.34.4.1 port

```
int socket_handler::port
```

Definition at line 34 of file socket_handler.h.

6.34.4.2 sock

```
int socket_handler::sock
```

Definition at line 34 of file socket_handler.h.

The documentation for this class was generated from the following files:

- src/ti/socket_handler.h
- src/ti/socket_handler.cpp

6.35 estts::telemetry_object Struct Reference

`#include <constants.h>`

Public Attributes

- int address {}
- int timeStamp {}
- int sequence {}
- int commandID {}
- int response_code {}
- const char * data {}

6.35.1 Detailed Description

Definition at line 270 of file constants.h.

6.35.2 Member Data Documentation

6.35.2.1 address

```
int estts::telemetry_object::address {}
```

Definition at line 271 of file constants.h.

6.35.2.2 commandID

```
int estts::telemetry_object::commandID {}
```

Definition at line 274 of file constants.h.

6.35.2.3 data

```
const char* estts::telemetry_object::data {}
```

Definition at line 276 of file constants.h.

6.35.2.4 response_code

```
int estts::telemetry_object::response_code {}
```

Definition at line 275 of file constants.h.

6.35.2.5 sequence

```
int estts::telemetry_object::sequence {}
```

Definition at line 273 of file constants.h.

6.35.2.6 timeStamp

```
int estts::telemetry_object::timeStamp {}
```

Definition at line 272 of file constants.h.

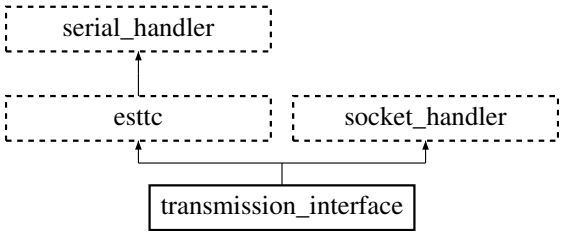
The documentation for this struct was generated from the following file:

- src/utils/constants.h

6.36 transmission_interface Class Reference

```
#include <transmission_interface.h>
```

Inheritance diagram for transmission_interface:



Public Member Functions

- void register_dispatch_function (const std::function< void()> &fct)
- transmission_interface ()
- ~transmission_interface ()
- void set_telem_callback (const std::function< estts::Status(std::string)> &cb)
- estts::Status transmit (const std::string &value)
- estts::Status transmit (const unsigned char *value, int length)
- std::string receive ()
- std::string nonblock_receive ()
- unsigned char * receive_uc ()
- estts::Status request_obc_session ()
- estts::Status request_gstxvr_session ()
- estts::Status end_gstxvr_session ()
- bool check_session_active () const
- estts::Status end_obc_session (const std::string &end_frame)
- bool check_data_available ()
- estts::Status gs_transmit (const std::string &value)
- estts::Status enable_pipe ()

  Enables transparent pipe mode on Endurosat UHF Transceiver module.
- estts::Status disable_pipe ()

Public Attributes

- bool obc_session_active
- bool satellite_in_range
- bool gstxvr_session_active

Additional Inherited Members

6.36.1 Detailed Description

Definition at line 14 of file transmission_interface.h.

6.36.2 Constructor & Destructor Documentation

6.36.2.1 transmission_interface()

```
transmission_interface::transmission_interface ( )  [explicit]
```

Default constructor that initializes each class inherited by transmission_interface, and ensures that the mutex is unlocked.

Definition at line 16 of file transmission_interface.cpp.

6.36.2.2 ~transmission_interface()

```
transmission_interface::~transmission_interface ( )
```

Default destructor that ensures a safe exit of transmission_interface

Definition at line 88 of file transmission_interface.cpp.

6.36.3 Member Function Documentation

6.36.3.1 check_data_available()

```
bool transmission_interface::check_data_available ( )
```

Function that returns the data available on the underlying interface.

Returns

bool

Definition at line 98 of file transmission_interface.cpp.

6.36.3.2 check_session_active()

```
bool transmission_interface::check_session_active ( ) const  [inline]
```

Function that returns the status of obc_session_active

Returns

bool

Definition at line 138 of file transmission_interface.h.

6.36.3.3 disable_pipe()

```
Status transmission_interface::disable_pipe ( )
```

Definition at line 239 of file transmission_interface.cpp.

6.36.3.4 enable_pipe()

```
Status transmission_interface::enable_pipe ( )
```

Enables transparent pipe mode on Endurosat UHF Transceiver module.

Returns

estts::Status indication success/failure of ESTTC command transmission

Definition at line 199 of file transmission_interface.cpp.

6.36.3.5 end_gstxvr_session()

```
estts::Status transmission_interface::end_gstxvr_session ( )
```

Definition at line 461 of file transmission_interface.cpp.

6.36.3.6 end_obc_session()

```
Status transmission_interface::end_obc_session (
            const std::string & end_frame )
```

Function that waits for PIPE to exit, thereby ending the communication session with the satellite.

Parameters

end_frame	Deprecated

Returns

ES_OK if session ends successfully

Definition at line 366 of file transmission_interface.cpp.

6.36.3.7 gs_transmit()

```
Status transmission_interface::gs_transmit (
            const std::string & value )
```

Uses EnduroSat transceiver to transmit string Value. Note that this is designed to be used to transmit data to the groundstation transceiver, not to other peripherals. Other methods should be created if another communication medium is required.

Parameters

value	String value to transmit.

Returns

ES_OK if transmission was successful

Definition at line 179 of file transmission_interface.cpp.

6.36.3.8 nonblock_receive()

```
std::string transmission_interface::nonblock_receive ( )
```

Nonblocking receive that returns immediately if no data is available.

Returns

"" if no data is available, or a string if data is received.

Definition at line 166 of file transmission_interface.cpp.

6.36.3.9 receive()

```
std::string transmission_interface::receive ( )
```

Function that uses underlying communication interface to receive data. This function blocks for ESTTS_AWAIT_↩ RESPONSE_PERIOD_SEC seconds while data is not available on the interface.

Returns

"" if function times out, or a string if data is received.

Definition at line 54 of file transmission_interface.cpp.

6.36.3.10 receive_uc()

```
unsigned char * transmission_interface::receive_uc ( )
```

Function that receives data in the unsigned char * form

Returns

nullptr if no data is found, or an unsigned char *

Definition at line 150 of file transmission_interface.cpp.

6.36.3.11 register_dispatch_function()

```
void transmission_interface::register_dispatch_function (
            const std::function< void()> & fct )
```

Definition at line 420 of file transmission_interface.cpp.

6.36.3.12 request_gstxvr_session()

```
estts::Status transmission_interface::request_gstxvr_session ( )
```

Definition at line 446 of file transmission_interface.cpp.

6.36.3.13 request_obc_session()

```
Status transmission_interface::request_obc_session ( )
```

Function that requests a new session with the OBC on the satellite. This function enables PIPE on the ground txvr and on the satellite txvr, and creates a new thread to keep the session as long as obc_session_active is true.

Returns

ES_OK if a session is active

Definition at line 283 of file transmission_interface.cpp.

6.36.3.14 set_telem_callback()

```
void transmission_interface::set_telem_callback (
            const std::function< estts::Status(std::string)> & cb )  [inline]
```

Function that sets a telemetry callback used by the request new session function, which clears the serial FIFO register before requesting a session. This ensures that whatever is receiving telemetry gets all available telemetry.

Parameters

cb	Telemetry callback with form std::function<estts::Status(std::string)>

Definition at line 83 of file transmission_interface.h.

6.36.3.15 transmit() [1/2]

```
Status transmission_interface::transmit (
            const std::string & value )
```

Uses EnduroSat transceiver to transmit const unsigned char * value. Function warns if a session is not currently active. Note that this is designed to be used to transmit data to the satellite, not to other peripherals. Other methods should be created if another communication medium is required.

Parameters

value	String value to transmit.

Returns

ES_OK if transmission was successful

Definition at line 30 of file transmission_interface.cpp.

6.36.3.16 transmit() [2/2]

```
Status transmission_interface::transmit (
            const unsigned char * value,
            int length )
```

Uses EnduroSat transceiver to transmit string value. Function warns if a session is not currently active. Note that this is designed to be used to transmit data to the satellite, not to other peripherals. Other methods should be created if another communication medium is required.

Parameters

value	
length	

Returns

Definition at line 111 of file transmission_interface.cpp.

6.36.4 Member Data Documentation

6.36.4.1 gstxvr_session_active

```
bool transmission_interface::gstxvr_session_active
```

Definition at line 62 of file transmission_interface.h.

6.36.4.2 obc_session_active

```
bool transmission_interface::obc_session_active
```

Definition at line 58 of file transmission_interface.h.

6.36.4.3 satellite_in_range

```
bool transmission_interface::satellite_in_range
```

Definition at line 60 of file transmission_interface.h.

The documentation for this class was generated from the following files:

- src/ti/transmission_interface.h
- src/ti/transmission_interface.cpp

6.37 estts::es2_telemetry::eps::vitals Struct Reference

```
#include <constants.h>
```

Public Attributes

- double battery_voltage
- double brownouts
- double charge_time_mins

6.37.1 Detailed Description

Definition at line 151 of file constants.h.

6.37.2 Member Data Documentation

6.37.2.1 battery_voltage

```
double estts::es2_telemetry::eps::vitals::battery_voltage
```

Definition at line 152 of file constants.h.

6.37.2.2 brownouts

```
double estts::es2_telemetry::eps::vitals::brownouts
```

Definition at line 153 of file constants.h.

6.37.2.3 charge_time_mins

```
double estts::es2_telemetry::eps::vitals::charge_time_mins
```

Definition at line 154 of file constants.h.

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

- src/utils/constants.h

6.38 estts::waiting_command Struct Reference

```
#include <constants.h>
```

Public Attributes

- std::string frame
- command_object * command
- std::string serial_number
- std::function< estts::Status(std::vector< estts::telemetry_object * >)> obj_callback
- std::function< estts::Status(std::string)> str_callback

6.38.1 Detailed Description

Definition at line 290 of file constants.h.

6.38.2 Member Data Documentation

6.38.2.1 command

```
command_object* estts::waiting_command::command
```

Definition at line 292 of file constants.h.

6.38.2.2 frame

```
std::string estts::waiting_command::frame
```

Definition at line 291 of file constants.h.

6.38.2.3 obj_callback

```
std::function<estts::Status(std::vector<estts::telemetry_object *>)> estts::waiting_command↩
::obj_callback
```

Definition at line 294 of file constants.h.

6.38.2.4 serial_number

```
std::string estts::waiting_command::serial_number
```

Definition at line 293 of file constants.h.

6.38.2.5 str_callback

```
std::function<estts::Status(std::string)> estts::waiting_command::str_callback
```

Definition at line 295 of file constants.h.

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

- src/utils/constants.h

Chapter 7

File Documentation

7.1 src/fapi/command_handler.cpp File Reference

```
#include <chrono>
#include <thread>
#include <iostream>
#include <vector>
#include <sstream>
#include "command_handler.h"
```

Functions

- estts::Status validate_response_code (int code)

  Validates response code returned by telemetry object.

7.1.1 Function Documentation

7.1.1.1 validate_response_code()

```
estts::Status validate_response_code (
            int code )
```

Validates response code returned by telemetry object.

Parameters

code	integer response code.

Returns

Status translation

Definition at line 22 of file command_handler.cpp.

7.2 command_handler.cpp

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002 //
00003 // Created by Hayden Roszell on 12/21/21.
00004 //
00005
00006 #include <chrono>
00007 #include <thread>
00008 #include <iostream>
00009 #include <vector>
00010 #include <sstream>
00011
00012 #include "command_handler.h"
00013
00014 using namespace std::this_thread; // sleep_for, sleep_until
00015 using namespace std::chrono; // nanoseconds, system_clock, seconds
00016
00022 estts::Status validate_response_code(int code) {
00023     if (code == estts::estts_response_code::SUCCESS)
00024         return estts::ES_SUCCESS;
00025     else if (code == estts::estts_response_code::UNRECOGNIZED_REQUEST)
00026         return estts::ES_BAD_OPTION;
00027     else if (code == estts::estts_response_code::OBC_FAILURE)
00028         return estts::ES_SERVER_ERROR;
00029     else return estts::ES_UNINITIALIZED;
00030 }
00031
00032 command_handler::command_handler() {
00033     this->ti = nullptr;
00034 }
00035
00036 estts::Status command_handler::execute(estts::waiting_command *command) {
00037     if (ti == nullptr) {
00038         SPDLOG_ERROR("Transmission interface not initialized. Was init_command_handler() called?");
00039         return estts::ES_UNINITIALIZED;
00040     }
00041     /*
00042     if (command->command != nullptr && command->frame.empty())
00043         return execute_obj(command);
00044     else if (command->command == nullptr && !command->frame.empty())
00045
00046     else {
00047         SPDLOG_ERROR("Command object not initialized properly. Please see documentation.");
00048         return estts::ES_UNINITIALIZED;
00049     }
00050      */
00051     return execute_str(command);
00052 }
00053
00054 estts::Status command_handler::execute_str(estts::waiting_command *command) {
00055     try {
00056         SPDLOG_INFO("Sending command");
00057         bool retry = true;
00058         int retries = 0;
00059         // Try to transmit frame
00060         while (ti->transmit(command->frame) != estts::ES_OK) {
00061             spdlog::error("Failed to transmit frame. Waiting {} seconds",
      estts::ESTTS_RETRY_WAIT_SEC);
00062             sleep_until(system_clock::now() + seconds(estts::ESTTS_RETRY_WAIT_SEC));
00063             retries++;
00064             if (retries > estts::endurosat::MAX_RETRIES) return estts::ES_UNSUCCESSFUL;
00065             SPDLOG_INFO("Retrying transmit ({}/{})", retries, estts::ESTTS_MAX_RETRIES);
00066         }
00067         // If we got this far,
00068         SPDLOG_DEBUG("Successfully transmitted command");
00069     }
00070     catch (const std::exception &e) {
00071         // TODO catch exceptions & do something smart with them
00072         spdlog::error("We failed somewhere");
00073         return estts::ES_UNSUCCESSFUL;
00074     }
```

```
00075
00076     SPDLOG_INFO("Waiting for a response from EagleSat II");
00077     sleep_until(system_clock::now() + milliseconds (100));
00078     auto telem = ti->receive();
00079     if (telem.empty())
00080         return estts::ES_UNSUCCESSFUL;
00081     SPDLOG_DEBUG("Got response from EagleSat II");
00082
00083     // todo there are likely error cases here that aren't accounted for. Find & fix them
00084
00085     if (command->str_callback != nullptr)
00086         if (estts::ES_OK != command->str_callback(telem))
00087             return estts::ES_UNSUCCESSFUL;
00088
00089     auto temp_completed = new completed;
00090     temp_completed->serial_number = command->serial_number;
00091     temp_completed->response_code = estts::ES_OK;
00092     completed_cache.push_back(temp_completed);
00093
00094     return estts::ES_OK;
00095 }
00096
00097 command_handler::~command_handler() = default;
00098
00099 estts::Status command_handler::init_command_handler(transmission_interface *ti) {
00100     this->ti = ti;
00101     return estts::ES_OK;
00102 }
00103
00104 estts::Status command_handler::map_telemetry_to_dispatched(const std::vector<estts::telemetry_object
      *> &telem) {
00105     // This function requires some intelligence. The basic premise is that the decoded telemetry
      objects
00106     // need to be mapped back to their associated dispatched command objects. There are a couple of
      ways we
00107     // can handle this. For this version, we're just going to handle each response individually and in
      real-time.
00108
00109     while (!dispatched.empty()) {
00110         auto current = dispatched.back();
00111         auto command_id = current->command->commandID;
00112         auto command_address = current->command->address;
00113         std::vector<estts::telemetry_object *> temp_telem;
00114         bool associated_frame_found;
00115         for (auto j : telem) {
00116             if (command_id == j->commandID && command_address == j->address) {
00117                 // We have a match!
00118                 associated_frame_found = true;
00119                 temp_telem.push_back(j);
00120             }
00121         }
00122         if (associated_frame_found) {
00123             current->telem_obj = temp_telem;
00124             current->response_code = validate_response_code(temp_telem[0]->response_code);
00125             if (current->obj_callback != nullptr)
00126                 if (estts::ES_OK != current->obj_callback(current->telem_obj)) // Call the
      obj_callback with the telemetry object
00127                     return estts::ES_UNSUCCESSFUL;
00128             // Todo this shoudn't return, it should schedule a new retry, or at least notify something
      that can retry
00129
00130             auto temp_completed = new completed;
00131             temp_completed->serial_number = current->serial_number;
00132             temp_completed->response_code = current->response_code;
00133             completed_cache.push_back(temp_completed);
00134         } else {
00135             SPDLOG_WARN("Didn't receive a telemetry response for command with SN {}",
      current->serial_number);
00136             current->response_code = estts::ES_UNSUCCESSFUL;
00137         }
00138         delete current;
00139         dispatched.pop_back();
00140     }
00141
00142     return estts::ES_OK;
00143 }
```

7.3 src/fapi/command_handler.h File Reference

```
#include <deque>
#include <functional>
```

```
#include <vector>
#include "transmission_interface.h"
#include "constants.h"
```

Classes

- class command_handler

7.4 command_handler.h

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002 //
00003 // Created by Hayden Roszell on 12/21/21.
00004 //
00005
00006 #ifndef ESTTS_FAPI_COMMAND_HANDLER_H
00007 #define ESTTS_FAPI_COMMAND_HANDLER_H
00008
00009 #include <deque>
00010 #include <functional>
00011 #include <vector>
00012 #include "transmission_interface.h"
00013 #include "constants.h"
00014
00015 class command_handler {
00016 private:
00017     typedef struct {
00018         std::string serial_number;
00019         estts::Status response_code;
00020     } completed;
00021
00022     transmission_interface *ti;
00023
00030     estts::Status map_telemetry_to_dispatched(const std::vector<estts::telemetry_object *> &telem);
00031
00038     estts::Status execute_str(estts::waiting_command *command);
00039 protected:
00040     std::vector<estts::dispatched_command *> dispatched;
00041
00042     std::vector<completed *> completed_cache;
00043
00048     explicit command_handler();
00049
00055     estts::Status init_command_handler(transmission_interface *ti);
00056
00057     ~command_handler();
00058
00064     estts::Status execute(estts::waiting_command * command);
00065 };
00066
00067
00068 #endif //ESTTS_FAPI_COMMAND_HANDLER_H
```

7.5 src/fapi/command_handler/acs_command.cpp File Reference

```
#include "acs_command.h"
```

7.6 acs_command.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/21/21.
00003 //
00004
00005 #include "acs_command.h"
00006
00007 estts::Status acs_command::get_current_position() {
00008
00009
00010     return estts::ES_UNINITIALIZED;
00011 }
00012
00013 std::string acs_command::deploy_magnetometer_boom(const estts::dispatch_fct &dispatch) {
00014
00015     auto temp = new estts::command_object;
00016
00017     temp->address = estts::es2_endpoint::ES_ACS;
00018     temp->commandID = estts::es2_commands::acs::ACS_DEP_MAG_BOOM;
00019     temp->method = estts::es2_commands::method::ES_WRITE;
00020     temp->sequence = 01;
00021     temp->timeStamp = 1729;
00022
00023     SPDLOG_INFO("Attempting to Deploy Magnetometer Boom (in theory)");
00024
00025     auto acs_dep_magnet_boom_callback = [] (const std::vector<estts::telemetry_object *>& telem) ->
      estts::Status {
00026         if (telem.empty()) {
00027             return estts::ES_UNINITIALIZED;
00028         }
00029         spdlog::info("Deployed Magnetometer Boom (hypothetically)!");
00030         return estts::ES_OK;
00031     };
00032
00033     return dispatch(temp, acs_dep_magnet_boom_callback);
00034 }
00035
00036 std::string acs_command::enable_acs(const estts::dispatch_fct &dispatch) {
00037
00038     auto temp = new estts::command_object;
00039
00040     temp->address = estts::es2_endpoint::ES_ACS;
00041     temp->commandID = estts::es2_commands::acs::ACS_ENABLE;
00042     temp->method = estts::es2_commands::method::ES_WRITE;
00043     temp->sequence = 01;
00044     temp->timeStamp = 1729;
00045
00046     SPDLOG_INFO("Attempting to Enable ACS");
00047
00048
00049     auto acs_enable_callback = [] (const std::vector<estts::telemetry_object *>& telem) ->
      estts::Status {
00050         if (telem.empty()) {
00051             return estts::ES_UNINITIALIZED;
00052         }
00053         spdlog::info("ACS Enabled!");
00054         return estts::ES_OK;
00055     };
00056
00057     return dispatch(temp, acs_enable_callback);
00058 }
00059
00060 std::string acs_command::power_acs(const estts::dispatch_fct &dispatch) {
00061
00062     auto temp = new estts::command_object;
00063
00064     temp->address = estts::es2_endpoint::ES_ACS;
00065     temp->commandID = estts::es2_commands::acs::ACS_POWER;
00066     temp->method = estts::es2_commands::method::ES_WRITE;
00067     temp->sequence = 01;
00068     temp->timeStamp = 1729;
00069
00070     SPDLOG_INFO("Attempting to Power ACS");
00071
00072     auto acs_power_callback = [] (const std::vector<estts::telemetry_object *>& telem) ->
      estts::Status {
00073         if (telem.empty()) {
00074             return estts::ES_UNINITIALIZED;
00075         }
00076         spdlog::info("ACS Powered!");
00077         return estts::ES_OK;
00078     };
00079
```

```
00080     return dispatch(temp, acs_power_callback);
00081 }
00082
00083 std::string acs_command::set_ctrl_mode(const estts::dispatch_fct &dispatch) {
00084
00085     auto temp = new estts::command_object;
00086
00087     temp->address = estts::es2_endpoint::ES_ACS;
00088     temp->commandID = estts::es2_commands::acs::ACS_SET_CTRL_MODE;
00089     temp->method = estts::es2_commands::method::ES_WRITE;
00090     temp->sequence = 01;
00091     temp->timeStamp = 1729;
00092
00093     SPDLOG_INFO("Attempting to Set Control Mode");
00094
00095     auto acs_power_callback = [] (const std::vector<estts::telemetry_object *>& telem) ->
      estts::Status {
00096         if (telem.empty()) {
00097             return estts::ES_UNINITIALIZED;
00098         }
00099         spdlog::info("Control Mode Set!");
00100         return estts::ES_OK;
00101     };
00102
00103     return dispatch(temp, acs_power_callback);
00104 }
00105
00106 std::string acs_command::set_est_mode(const estts::dispatch_fct &dispatch) {
00107
00108     auto temp = new estts::command_object;
00109
00110     temp->address = estts::es2_endpoint::ES_ACS;
00111     temp->commandID = estts::es2_commands::acs::ACS_SET_EST_MODE;
00112     temp->method = estts::es2_commands::method::ES_WRITE;
00113     temp->sequence = 01;
00114     temp->timeStamp = 1729;
00115
00116     SPDLOG_INFO("Attempting to Set Estimation Mode");
00117
00118     auto acs_power_callback = [] (const std::vector<estts::telemetry_object *>& telem) ->
      estts::Status {
00119         if (telem.empty()) {
00120             return estts::ES_UNINITIALIZED;
00121         }
00122         spdlog::info("Estimation Mode Set!");
00123         return estts::ES_OK;
00124     };
00125
00126     return dispatch(temp, acs_power_callback);
00127 }
```

7.7 src/fapi/command_handler/acs_command.h File Reference

```
#include "constants.h"
```

Classes

- class acs_command

7.8 acs_command.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/21/21.
00003 //
00004
00005 #ifndef ESTTS_ACS_COMMAND_H
00006 #define ESTTS_ACS_COMMAND_H
00007
00008 #include "constants.h"
```

```
00009
00010 class acs_command {
00011 public:
00012     estts::Status get_current_position();
00013     std::string deploy_magnetometer_boom(const estts::dispatch_fct &dispatch);
00014     std::string enable_acs(const estts::dispatch_fct &dispatch);
00015     std::string power_acs(const estts::dispatch_fct &dispatch);
00016     std::string set_ctrl_mode(const estts::dispatch_fct &dispatch);
00017     std::string set_est_mode(const estts::dispatch_fct &dispatch);
00018 };
00019
00020
00021 #endif //ESTTS_ACS_COMMAND_H
```

7.9 src/fapi/command_handler/crp_command.cpp File Reference

```
#include "crp_command.h"
```

7.10 crp_command.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/21/21.
00003 //
00004
00005 #include "crp_command.h"
```

7.11 src/fapi/command_handler/crp_command.h File Reference

Classes

- class crp_command

7.12 crp_command.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/21/21.
00003 //
00004
00005 #ifndef ESTTS_CRP_COMMAND_H
00006 #define ESTTS_CRP_COMMAND_H
00007
00008
00009 class crp_command {
00010
00011 };
00012
00013
00014 #endif //ESTTS_CRP_COMMAND_H
```

7.13 src/fapi/command_handler/eps_command.cpp File Reference

```
#include <vector>
#include "eps_command.h"
```

7.14 eps_command.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/21/21.
00003 //
00004
00005 #include <vector>
00006 #include "eps_command.h"
00007
00008 std::string eps_command::get_eps_vitals(const estts::dispatch_fct& dispatch, const
       std::function<estts::Status(estts::es2_telemetry::eps::vitals *)>& telem_callback) {
00009     SPDLOG_INFO("Getting EagleSat II EPS Vitals");
00010     auto command = new estts::command_object;
00011
00012     command->address = estts::es2_endpoint::ES_EPS;
00013     command->commandID = estts::es2_commands::eps::EPS_GET_HEALTH;
00014     command->method = estts::es2_commands::method::ES_READ;
00015     command->sequence = 01;
00016     command->timeStamp = 8456;
00017
00018
00019
00020     // This obj_callback is expected by the dispatcher/command handler, which will filter and
       construct telemetry objects
00021     // and pass them to this function. Once this process is complete, this function should know how to
       decode
00022     // the telemetry object into the expected structure (in this case, an EPS Vitals structure), and
       pass the vitals to
00023     // the obj_callback function passed as an argument. This process keeps logic specific to EPS in
       the EPS
00024     // command function, and changes the execution of the function to the command handler, where it
       should be. However, from
00025     // the perspective of this EPS get vitals function, the specific time when the obj_callback is
       called doesn't matter in any way,
00026     // because now the responsibility of handling the telemetry is pushed to the future when it's
       actually available.
00027     auto eps_telem_decomposition_callback = [telem_callback] (const
       std::vector<estts::telemetry_object *>& telem) -> estts::Status {
00028         if (telem.empty()) {
00029             return estts::ES_UNINITIALIZED;
00030         }
00031         // TODO do something with the telem_obj vector passed to this function
00032         auto vitals = new estts::es2_telemetry::eps::vitals;
00033
00034         vitals->battery_voltage = 9.0;
00035         vitals->brownouts = 0;
00036         vitals->charge_time_mins = 24;
00037
00038         spdlog::info("EPS Vitals: battery voltage: {} - brownouts: {} - charge time (min) {}",
       vitals->battery_voltage, vitals->brownouts, vitals->charge_time_mins);
00039
00040         telem_callback(vitals);
00041     };
00042
00043     // We're expecting the dispatcher to call the telemetry decomposition function. As seen above, the
       telemetry
00044     // decomposition knows how to handle a telemetry object of the EPS vitals type (which is matched
       by the address/command
00045     // in the command configuration), and calls the obj_callback function passed in by the calling
       body. Note that when this function
00046     // returns, it is expected that the entire telemetry collection and storage process is handled by
       the callbacks.
00047     // This means that at no point does this function need to run again, because the context is
       implied by the lambdas.
00048
00049     // Note that our obj_callback model allows this function to return with no repercussions, and
       using the unique command
00050     // serial number, we can fetch the command status at any time.
00051     return dispatch(command, eps_telem_decomposition_callback);
00052 }
00053
00054 std::string eps_command::get_eps_batteryVoltage(const estts::dispatch_fct &dispatch) {
00055     auto temp = new estts::command_object;
00056
00057     temp->address = estts::es2_endpoint::ES_EPS;
00058     temp->commandID = estts::es2_commands::eps::EPS_GET_BATTERY_VOLTAGE;
00059     temp->method = estts::es2_commands::method::ES_READ;
00060     temp->sequence = 01;
00061     temp->timeStamp = 8456;
00062
00063     SPDLOG_INFO("Attempting to get EPS battery voltage");
00064
00065     auto eps_telem_decomposition_callback = [] (const std::vector<estts::telemetry_object *>& telem)
       -> estts::Status {
```

```
00066         if (telem.empty()) {
00067             return estts::ES_UNINITIALIZED;
00068         }
00069         auto eps_voltage = new estts::es2_telemetry::eps::eps_voltage;
00070 
00071         eps_voltage->battery_voltage = 9.0;
00072 
00073         spdlog::info("EPS battery voltage : {}", eps_voltage->battery_voltage);
00074         spdlog::info("Got back battery voltage - it worked");
00075         return estts::ES_OK;
00076     };
00077 
00078     return dispatch(temp, eps_telem_decomposition_callback);
00079 }
00080 
00081 
00082 std::string eps_command::get_eps_batteryCurrent(const estts::dispatch_fct &dispatch) {
00083     auto command = new estts::command_object;
00084 
00085     command->address = estts::es2_endpoint::ES_EPS;
00086     command->commandID = estts::es2_commands::eps::EPS_GET_BATTERY_CURRENT;
00087     command->method = estts::es2_commands::method::ES_READ;
00088     command->sequence = 01;
00089     command->timeStamp = 8456;
00090 
00091     SPDLOG_INFO("Attempting to get EPS battery current");
00092 
00093 
00094     auto eps_telem_decomposition_callback = [] (const std::vector<estts::telemetry_object *>& telem)
     -> estts::Status {
00095         if (telem.empty()) {
00096             return estts::ES_UNINITIALIZED;
00097         }
00098         // TODO do something with the telem_obj vector passed to this function
00099         auto eps_current = new estts::es2_telemetry::eps::eps_current;
00100 
00101         eps_current->battery_current = 9.0;
00102 
00103         spdlog::info("EPS battery current : {}", eps_current->battery_current);
00104 
00105         spdlog::info("Got back battery current - it worked");
00106     };
00107 
00108     return dispatch(command, eps_telem_decomposition_callback);
00109 }
00110 
00111 
00112 std::string eps_command::get_eps_5Vbus_current(const estts::dispatch_fct &dispatch) {
00113     auto temp = new estts::command_object;
00114 
00115     temp->address = estts::es2_endpoint::ES_EPS;
00116     temp->commandID = estts::es2_commands::eps::EPS_GET_5VBUS_CURRENT;
00117     temp->method = estts::es2_commands::method::ES_READ;
00118     temp->sequence = 01;
00119     temp->timeStamp = 8467;
00120 
00121     SPDLOG_INFO("Attempting to get EPS 5V bus current");
00122 
00123 
00124 
00125 
00126     auto eps_telem_decomposition_callback = [] (const std::vector<estts::telemetry_object *>& telem)
     -> estts::Status {
00127         if (telem.empty()) {
00128             return estts::ES_UNINITIALIZED;
00129         }
00130         // TODO do something with the telem_obj vector passed to this function
00131         auto eps_5Vbus_current = new estts::es2_telemetry::eps::eps_5Vbus_current;
00132 
00133         eps_5Vbus_current->bus_current = 9.0;
00134 
00135         spdlog::info("EPS 5V bus current : {}", eps_5Vbus_current->bus_current);
00136 
00137         spdlog::info("Got back 5V bus current - it worked");
00138     };
00139 
00140     return dispatch(temp, eps_telem_decomposition_callback);
00141 }
00142 
00143 
00144 std::string eps_command::get_eps_3Vbus_current(const estts::dispatch_fct &dispatch) {
00145 
00146     auto temp = new estts::command_object;
00147 
00148     temp->address = estts::es2_endpoint::ES_EPS;
00149     temp->commandID = estts::es2_commands::eps::EPS_GET_3VBUS_CURRENT;
00150     temp->method = estts::es2_commands::method::ES_READ;
```

```
00151     temp->sequence = 01;
00152     temp->timeStamp = 8489;
00153
00154     SPDLOG_INFO("Attempting to get EPS 3.3V bus current");
00155
00156
00157
00158
00159     auto eps_telem_decomposition_callback = [] (const std::vector<estts::telemetry_object *>& telem)
       -> estts::Status {
00160         if (telem.empty()) {
00161             return estts::ES_UNINITIALIZED;
00162         }
00163         // TODO do something with the telem_obj vector passed to this function
00164         auto eps_3Vbus_current = new estts::es2_telemetry::eps::eps_3Vbus_current;
00165
00166         eps_3Vbus_current->bus_current = 9.0;
00167
00168         spdlog::info("EPS 3.3V bus current : {}", eps_3Vbus_current->bus_current);
00169
00170         spdlog::info("Got back 3.3V bus current - it worked");
00171     };
00172
00173     return dispatch(temp, eps_telem_decomposition_callback);
00174 }
00175
00176
00177 std::string eps_command::get_eps_temp_sensor5(const estts::dispatch_fct &dispatch) {
00178
00179     auto temp = new estts::command_object;
00180
00181     temp->address = estts::es2_endpoint::ES_EPS;
00182     temp->commandID = estts::es2_commands::eps::EPS_GET_TEMP_SENSOR5;
00183     temp->method = estts::es2_commands::method::ES_READ;
00184     temp->sequence = 01;
00185     temp->timeStamp = 8330;
00186
00187     SPDLOG_INFO("Attempting to get EPS external temperature sensor 5");
00188
00189
00190
00191     auto eps_telem_decomposition_callback = [] (const std::vector<estts::telemetry_object *>& telem)
       -> estts::Status {
00192         if (telem.empty()) {
00193             return estts::ES_UNINITIALIZED;
00194         }
00195         auto eps_externalTemp_sensor5 = new estts::es2_telemetry::eps::eps_externalTemp_sensor5;
00196
00197         eps_externalTemp_sensor5->external_temperature = 1.5;
00198
00199         spdlog::info("EPS external temperature sensor #5 reads : {}",
       eps_externalTemp_sensor5->external_temperature);
00200         spdlog::info("Got back the external temperature sensor - it worked");
00201         return estts::ES_OK;
00202     };
00203
00204     return dispatch(temp, eps_telem_decomposition_callback);
00205 }
00206
00207
00208 std::string eps_command::get_eps_temp_sensor6(const estts::dispatch_fct &dispatch) {
00209
00210     auto temp = new estts::command_object;
00211
00212     temp->address = estts::es2_endpoint::ES_EPS;
00213     temp->commandID = estts::es2_commands::eps::EPS_GET_TEMP_SENSOR6;
00214     temp->method = estts::es2_commands::method::ES_READ;
00215     temp->sequence = 01;
00216     temp->timeStamp = 8987;
00217
00218     SPDLOG_INFO("Attempting to get EPS external temperature sensor 6");
00219
00220
00221
00222     auto eps_telem_decomposition_callback = [] (const std::vector<estts::telemetry_object *>& telem)
       -> estts::Status {
00223         if (telem.empty()) {
00224             return estts::ES_UNINITIALIZED;
00225         }
00226         auto eps_externalTemp_sensor6 = new estts::es2_telemetry::eps::eps_externalTemp_sensor6;
00227
00228         eps_externalTemp_sensor6->external_temperature = 1.5;
00229
00230         spdlog::info("EPS external temperature sensor #6 reads : {}",
       eps_externalTemp_sensor6->external_temperature);
00231         spdlog::info("Got back the external temperature sensor - it worked");
00232         return estts::ES_OK;
```

```
00233     };
00234
00235     return dispatch(temp, eps_telem_decomposition_callback);
00236 }
00237
00238
00239 std::string eps_command::get_eps_temp_sensor7(const estts::dispatch_fct &dispatch) {
00240
00241     auto temp = new estts::command_object;
00242
00243     temp->address = estts::es2_endpoint::ES_EPS;
00244     temp->commandID = estts::es2_commands::eps::EPS_GET_TEMP_SENSOR7;
00245     temp->method = estts::es2_commands::method::ES_READ;
00246     temp->sequence = 01;
00247     temp->timeStamp = 8987;
00248
00249     SPDLOG_INFO("Attempting to get EPS external temperature sensor 7");
00250
00251
00252
00253     auto eps_telem_decomposition_callback = [] (const std::vector<estts::telemetry_object *>& telem)
      -> estts::Status {
00254         if (telem.empty()) {
00255             return estts::ES_UNINITIALIZED;
00256         }
00257         auto eps_externalTemp_sensor7 = new estts::es2_telemetry::eps::eps_externalTemp_sensor7;
00258
00259         eps_externalTemp_sensor7->external_temperature = 1.5;
00260
00261         spdlog::info("EPS external temperature sensor #7 reads : {}",
      eps_externalTemp_sensor7->external_temperature);
00262         spdlog::info("Got back the external temperature sensor - it worked");
00263         return estts::ES_OK;
00264     };
00265
00266     return dispatch(temp, eps_telem_decomposition_callback);
00267 }
00268
00269
00270 std::string eps_command::get_eps_battery_temp_sensor1(const estts::dispatch_fct &dispatch) {
00271
00272     auto temp = new estts::command_object;
00273
00274     temp->address = estts::es2_endpoint::ES_EPS;
00275     temp->commandID = estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR1;
00276     temp->method = estts::es2_commands::method::ES_READ;
00277     temp->sequence = 01;
00278     temp->timeStamp = 8278;
00279
00280     SPDLOG_INFO("Attempting to get EPS battery temp sensor 1");
00281
00282
00283
00284     auto eps_telem_decomposition_callback = [] (const std::vector<estts::telemetry_object *>& telem)
      -> estts::Status {
00285         if (telem.empty()) {
00286             return estts::ES_UNINITIALIZED;
00287         }
00288         auto eps_batteryTemp_sensor1 = new estts::es2_telemetry::eps::eps_batteryTemp_sensor1;
00289
00290         eps_batteryTemp_sensor1->battery_temperature = 1.5;
00291
00292         spdlog::info("EPS battery temp sensor 1 reads : {}",
      eps_batteryTemp_sensor1->battery_temperature);
00293         spdlog::info("Got back the battery temp sensor - it worked");
00294         return estts::ES_OK;
00295     };
00296
00297     return dispatch(temp, eps_telem_decomposition_callback);
00298 }
00299
00300
00301 std::string eps_command::get_eps_battery_temp_sensor2(const estts::dispatch_fct &dispatch) {
00302
00303     auto temp = new estts::command_object;
00304
00305     temp->address = estts::es2_endpoint::ES_EPS;
00306     temp->commandID = estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR2;
00307     temp->method = estts::es2_commands::method::ES_READ;
00308     temp->sequence = 01;
00309     temp->timeStamp = 8672;
00310
00311     SPDLOG_INFO("Attempting to get EPS battery temp sensor 2");
00312
00313
00314
00315     auto eps_telem_decomposition_callback = [] (const std::vector<estts::telemetry_object *>& telem)
```

```
    -> estts::Status {
00316         if (telem.empty()) {
00317             return estts::ES_UNINITIALIZED;
00318         }
00319         auto eps_batteryTemp_sensor2 = new estts::es2_telemetry::eps::eps_batteryTemp_sensor2;
00320 
00321         eps_batteryTemp_sensor2->battery_temperature = 1.5;
00322 
00323         spdlog::info("EPS battery temp sensor 2 reads : {}",
    eps_batteryTemp_sensor2->battery_temperature);
00324         spdlog::info("Got back the battery temp sensor - it worked");
00325         return estts::ES_OK;
00326     };
00327 
00328     return dispatch(temp, eps_telem_decomposition_callback);
00329 }
00330 
00331 
00332 std::string eps_command::get_eps_battery_temp_sensor3(const estts::dispatch_fct &dispatch) {
00333 
00334     auto temp = new estts::command_object;
00335 
00336     temp->address = estts::es2_endpoint::ES_EPS;
00337     temp->commandID = estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR3;
00338     temp->method = estts::es2_commands::method::ES_READ;
00339     temp->sequence = 01;
00340     temp->timeStamp = 8782;
00341 
00342     SPDLOG_INFO("Attempting to get EPS battery temp sensor 3");
00343 
00344 
00345 
00346     auto eps_telem_decomposition_callback = [] (const std::vector<estts::telemetry_object *>& telem)
    -> estts::Status {
00347         if (telem.empty()) {
00348             return estts::ES_UNINITIALIZED;
00349         }
00350         auto eps_batteryTemp_sensor3 = new estts::es2_telemetry::eps::eps_batteryTemp_sensor3;
00351 
00352         eps_batteryTemp_sensor3->battery_temperature = 1.5;
00353 
00354         spdlog::info("EPS battery temp sensor 3 reads : {}",
    eps_batteryTemp_sensor3->battery_temperature);
00355         spdlog::info("Got back the battery temp sensor - it worked");
00356         return estts::ES_OK;
00357     };
00358 
00359     return dispatch(temp, eps_telem_decomposition_callback);
00360 }
00361 
00362 
00363 std::string eps_command::get_eps_battery_temp_sensor4(const estts::dispatch_fct &dispatch) {
00364 
00365     auto temp = new estts::command_object;
00366 
00367     temp->address = estts::es2_endpoint::ES_EPS;
00368     temp->commandID = estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR4;
00369     temp->method = estts::es2_commands::method::ES_READ;
00370     temp->sequence = 01;
00371     temp->timeStamp = 8674;
00372 
00373     SPDLOG_INFO("Attempting to get EPS battery temp sensor 4");
00374 
00375 
00376 
00377     auto eps_telem_decomposition_callback = [] (const std::vector<estts::telemetry_object *>& telem)
    -> estts::Status {
00378         if (telem.empty()) {
00379             return estts::ES_UNINITIALIZED;
00380         }
00381         auto eps_batteryTemp_sensor4 = new estts::es2_telemetry::eps::eps_batteryTemp_sensor4;
00382 
00383         eps_batteryTemp_sensor4->battery_temperature = 1.5;
00384 
00385         spdlog::info("EPS battery temp sensor 4 reads : {}",
    eps_batteryTemp_sensor4->battery_temperature);
00386         spdlog::info("Got back the battery temp sensor - it worked");
00387         return estts::ES_OK;
00388     };
00389 
00390     return dispatch(temp, eps_telem_decomposition_callback);
00391 }
00392 
00393 eps_command::eps_command() = default;
```

7.15 src/fapi/command_handler/eps_command.h File Reference

```
#include <functional>
#include <string>
#include <constants.h>
```

Classes

- class eps_command

7.16 eps_command.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/21/21.
00003 //
00004
00005 #ifndef ESTTS_EPS_COMMAND_H
00006 #define ESTTS_EPS_COMMAND_H
00007
00008 #include <functional>
00009 #include <string>
00010 #include <constants.h>
00011
00012 class eps_command {
00013 public:
00014     explicit eps_command();
00015
00016     std::string get_eps_vitals(const estts::dispatch_fct &dispatch, const std::function<estts::Status(
00017             estts::es2_telemetry::eps::vitals *)> &telem_callback);
00018
00019     std::string get_eps_batteryVoltage(const estts::dispatch_fct &dispatch);
00020
00021     std::string get_eps_batteryCurrent(const estts::dispatch_fct &dispatch);
00022
00023     std::string get_eps_5Vbus_current(const estts::dispatch_fct &dispatch);
00024
00025     std::string get_eps_3Vbus_current(const estts::dispatch_fct &dispatch);
00026
00027     std::string get_eps_temp_sensor5(const estts::dispatch_fct &dispatch);
00028
00029     std::string get_eps_temp_sensor6(const estts::dispatch_fct &dispatch);
00030
00031     std::string get_eps_temp_sensor7(const estts::dispatch_fct &dispatch);
00032
00033     std::string get_eps_battery_temp_sensor1(const estts::dispatch_fct &dispatch);
00034
00035     std::string get_eps_battery_temp_sensor2(const estts::dispatch_fct &dispatch);
00036
00037     std::string get_eps_battery_temp_sensor3(const estts::dispatch_fct &dispatch);
00038
00039     std::string get_eps_battery_temp_sensor4(const estts::dispatch_fct &dispatch);
00040
00041 };
00042
00043
00044 #endif //ESTTS_EPS_COMMAND_H
00045
```

7.17 src/fapi/command_handler/mde_command.cpp File Reference

```
#include "mde_command.h"
```

7.18 mde_command.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/21/21.
00003 //
00004
00005 #include "mde_command.h"
```

7.19 src/fapi/command_handler/mde_command.h File Reference

Classes

- class mde_command

7.20 mde_command.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/21/21.
00003 //
00004
00005 #ifndef ESTTS_MDE_COMMAND_H
00006 #define ESTTS_MDE_COMMAND_H
00007
00008
00009 class mde_command {
00010
00011 };
00012
00013
00014 #endif //ESTTS_MDE_COMMAND_H
```

7.21 src/fapi/command_handler/obc_command.cpp File Reference

```
#include "obc_command.h"
```

7.22 obc_command.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/21/21.
00003 //
00004
00005 #include "obc_command.h"
```

7.23 src/fapi/command_handler/obc_command.h File Reference

Classes

- class obc_command

7.24 obc_command.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/21/21.
00003 //
00004
00005 #ifndef ESTTS_OBC_COMMAND_H
00006 #define ESTTS_OBC_COMMAND_H
00007
00008
00009 class obc_command {
00010
00011 };
00012
00013
00014 #endif //ESTTS_OBC_COMMAND_H
```

7.25 src/fapi/cosmos_groundstation_handler.cpp File Reference

```
#include "sstream"
#include "cosmos_groundstation_handler.h"
#include "socket_handler.h"
#include "constants.h"
```

7.26 cosmos_groundstation_handler.cpp

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002
00003 #include "sstream"
00004 #include "cosmos_groundstation_handler.h"
00005 #include "socket_handler.h"
00006 #include "constants.h"
00007
00008 cosmos_groundstation_handler::cosmos_groundstation_handler() {
00009    sock = new socket_handler(estts::cosmos::COSMOS_SERVER_ADDR,
      estts::cosmos::COSMOS_GROUNDSTATION_CMD_TELEM_PORT);
00010 }
00011
00012 void cosmos_groundstation_handler::groundstation_cosmos_worker() {
00013     std::string command;
00014     for (;;) {
00015         command = sock->read_socket_s();
00016         if (not command.empty()) {
00017             groundstation_manager->schedule_command(command,
      get_generic_command_callback_lambda(command, sock));
00018         }
00019     }
00020 }
00021
00022 estts::Status cosmos_groundstation_handler::cosmos_groundstation_init(transmission_interface *ti) {
00023     sock->init_socket_handle();
00024     groundstation_manager = new groundstation_cmdtelem_manager(ti,
      get_generic_telemetry_callback_lambda(sock));
00025     cosmos_worker = std::thread(&cosmos_groundstation_handler::groundstation_cosmos_worker, this);
00026     SPDLOG_TRACE("Created groundstation COSMOS worker thread with ID {}",
      std::hash<std::thread::id>{}(cosmos_worker.get_id()));
00027
00028     return estts::ES_OK;
00029 }
00030
00031 std::function<estts::Status(std::string)>
00032 cosmos_groundstation_handler::get_generic_command_callback_lambda(std::string command, socket_handler
      *sock) {
00033     return [command, sock] (const std::string& telem) -> estts::Status {
00034         if (telem.empty() || sock == nullptr) {
00035             return estts::ES_UNINITIALIZED;
00036         }
00037         sock->write_socket_s(command);
```

```
00038         return estts::ES_OK;
00039     };
00040 }
00041
00042 std::function<estts::Status(std::string)>
00043 cosmos_groundstation_handler::get_generic_telemetry_callback_lambda(socket_handler *sock) {
00044     return [sock] (const std::string& telem) -> estts::Status {
00045         if (telem.empty() || sock == nullptr) {
00046             return estts::ES_UNINITIALIZED;
00047         }
00048         sock->write_socket_s(telem);
00049         return estts::ES_OK;
00050     };
00051 }
```

7.27 src/fapi/cosmos_groundstation_handler.h File Reference

```
#include <thread>
#include "constants.h"
#include "socket_handler.h"
#include "groundstation_cmdtelem_manager.h"
#include "transmission_interface.h"
```

Classes

- class cosmos_groundstation_handler

7.28 cosmos_groundstation_handler.h

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002
00003 #ifndef ESTTS_COSMOS_GROUNDSTATION_HANDLER_H
00004 #define ESTTS_COSMOS_GROUNDSTATION_HANDLER_H
00005
00006 #include <thread>
00007 #include "constants.h"
00008 #include "socket_handler.h"
00009 #include "groundstation_cmdtelem_manager.h"
00010 #include "transmission_interface.h"
00011
00012 class cosmos_groundstation_handler {
00013 private:
00014     socket_handler * sock;
00015     std::thread cosmos_worker;
00016     groundstation_cmdtelem_manager * groundstation_manager;
00017
00018     transmission_interface * ti;
00019
00024     [[noreturn]] void groundstation_cosmos_worker();
00025
00033     static std::function<estts::Status(std::string)> get_generic_command_callback_lambda(std::string
       command, socket_handler * sock);
00034
00041     static std::function<estts::Status(std::string)>
       get_generic_telemetry_callback_lambda(socket_handler * sock);
00042
00043 public:
00047     cosmos_groundstation_handler();
00048
00054     estts::Status cosmos_groundstation_init(transmission_interface *ti);
00055 };
00056
00057
00058 #endif //ESTTS_COSMOS_GROUNDSTATION_HANDLER_H
```

7.29 src/fapi/cosmos_handler.cpp File Reference

```
#include <iostream>
#include <unistd.h>
#include "cosmos_handler.h"
```

7.30 cosmos_handler.cpp

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002 //
00003 // Created by Hayden Roszell on 2/24/22.
00004 //
00005
00006 #include <iostream>
00007 #include <unistd.h>
00008 #include "cosmos_handler.h"
00009
00010 cosmos_handler::cosmos_handler() {
00011     ti = new transmission_interface();
00012     this->sock = new socket_handler(estts::cosmos::COSMOS_SERVER_ADDR,
      estts::cosmos::COSMOS_PRIMARY_CMD_TELEM_PORT);
00013     obc_session = nullptr;
00014 }
00015
00016 [[noreturn]] estts::Status cosmos_handler::primary_cosmos_worker() {
00017     std::string temp_string;
00018     for (;;) {
00019         temp_string = sock->read_socket_s();
00020         if (!temp_string.empty()) {
00021             auto sn = obc_session->schedule_command(temp_string,
      get_generic_command_callback_lambda(temp_string, sock));
00022         }
00023     }
00024 }
00025
00026 estts::Status cosmos_handler::cosmos_init() {
00027     if (sock->init_socket_handle() != estts::ES_OK)
00028         return estts::ES_UNSUCCESSFUL;
00029     obc_session = new obc_session_manager(ti, get_generic_telemetry_callback_lambda(sock));
00030
00031     cosmos_worker = std::thread(&cosmos_handler::primary_cosmos_worker, this);
00032     SPDLOG_TRACE("Created primary COSMOS worker thread with ID {}",
      std::hash<std::thread::id>{}(cosmos_worker.get_id()));
00033
00034     //this->cosmos_satellite_txvr_init();
00035     this->cosmos_groundstation_init(ti);
00036
00037     return estts::ES_OK;
00038 }
00039
00040 std::function<estts::Status(std::string)> cosmos_handler::get_generic_command_callback_lambda(const
      std::string& command, socket_handler * sock) {
00041     return [command, sock] (const std::string& telem) -> estts::Status {
00042         if (telem.empty() || sock == nullptr) {
00043             return estts::ES_UNINITIALIZED;
00044         }
00045         std::stringstream temp;
00046         for (char i : command) {
00047             if (i != '\r')
00048                 temp << i;
00049         }
00050         spdlog::info("COSMOS Command Callback Lambda --> Sent {} and got back: {}", temp.str(),
      telem);
00051         sock->write_socket_s(telem);
00052         return estts::ES_OK;
00053     };
00054 }
00055
00056 std::function<estts::Status(std::string)>
      cosmos_handler::get_generic_telemetry_callback_lambda(socket_handler * sock) {
00057     return [sock] (const std::string& telem) -> estts::Status {
00058         if (telem.empty() || sock == nullptr) {
00059             return estts::ES_UNINITIALIZED;
00060         }
00061         std::stringstream temp;
```

```
00062         for (char i : telem) {
00063             if (i != '\r')
00064                 temp « i;
00065         }
00066         spdlog::info("COSMOS Telemetry Callback Lambda --> Found: {}", temp.str());
00067         sock->write_socket_s(telem);
00068         return estts::ES_OK;
00069     };
00070 }
```

7.31 src/fapi/cosmos_handler.h File Reference

```
#include <thread>
#include "constants.h"
#include "cosmos_groundstation_handler.h"
#include "cosmos_satellite_txvr_handler.h"
#include "socket_handler.h"
#include "obc_session_manager.h"
```

Classes

- class cosmos_handler

7.32 cosmos_handler.h

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002 //
00003 // Created by Hayden Roszell on 2/24/22.
00004 //
00005
00006 #ifndef ESTTS_COSMOS_COSMOS_HANDLER_H
00007 #define ESTTS_COSMOS_COSMOS_HANDLER_H
00008
00009 #include <thread>
00010 #include "constants.h"
00011 #include "cosmos_groundstation_handler.h"
00012 #include "cosmos_satellite_txvr_handler.h"
00013 #include "socket_handler.h"
00014 #include "obc_session_manager.h"
00015
00022 class cosmos_handler : virtual public cosmos_groundstation_handler, virtual public
       cosmos_satellite_txvr_handler {
00023 private:
00024     transmission_interface *ti;
00025
00026     socket_handler * sock;
00027
00028     std::thread cosmos_worker;
00029
00030     obc_session_manager * obc_session;
00031
00036     [[noreturn]] estts::Status primary_cosmos_worker();
00037
00045     static std::function<estts::Status(std::string)> get_generic_command_callback_lambda(const
       std::string& command, socket_handler * sock);
00046
00053     static std::function<estts::Status(std::string)>
       get_generic_telemetry_callback_lambda(socket_handler * sock);
00054 public:
00058     cosmos_handler();
00059
00065     estts::Status cosmos_init();
00066
00071     void initialize_cosmos_daemon() { cosmos_worker.join(); }
00072 };
00073
00074
00075 #endif //ESTTS_COSMOS_COSMOS_HANDLER_H
```

7.33 src/fapi/cosmos_satellite_txvr_handler.cpp File Reference

```
#include "cosmos_satellite_txvr_handler.h"
```

7.34 cosmos_satellite_txvr_handler.cpp

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002
00003 #include "cosmos_satellite_txvr_handler.h"
00004
00005 cosmos_satellite_txvr_handler::cosmos_satellite_txvr_handler() {
00006     // todo Create a new socket handler instance with the COSMOS server address and
      COSMOS_GROUNDSTATION_CMD_TELEM_PORT
00007 }
00008
00009 estts::Status cosmos_satellite_txvr_handler::cosmos_satellite_txvr_init() {
00010     // todo Initialize the socket created in the constructor with the init_socket_handle() method
00011     // todo Create new groundstation cmdtelem handler and initialize it with the telemetry callback
00012
00013     cosmos_worker = std::thread(&cosmos_satellite_txvr_handler::satellite_txvr_cosmos_worker, this);
00014     SPDLOG_TRACE("Created satellite transceiver COSMOS worker thread with ID {}",
      std::hash<std::thread::id>{}(cosmos_worker.get_id()));
00015
00016     return estts::ES_OK;
00017 }
00018
00019 void cosmos_satellite_txvr_handler::satellite_txvr_cosmos_worker() {
00020     for (;;) {
00021         // todo Indefinitely read the open socket port, and schedule a new command with the associated
      cmdtelem manager when a command is received from COSMOS
00022     }
00023 }
00024
00025 std::function<estts::Status(std::string)>
00026 cosmos_satellite_txvr_handler::get_generic_command_callback_lambda(std::string command, socket_handler
      *sock) {
00027     return [command, sock] (const std::string& telem) -> estts::Status {
00028         if (telem.empty() || sock == nullptr) {
00029             return estts::ES_UNINITIALIZED;
00030         }
00031         // todo handle command response that is passed to this callback as argument (hint use the
      socket handler)
00032         return estts::ES_OK;
00033     };
00034 }
```

7.35 src/fapi/cosmos_satellite_txvr_handler.h File Reference

```
#include <thread>
#include "constants.h"
#include "socket_handler.h"
#include "satellite_txvr_cmdtelem_manager.h"
```

Classes

- class cosmos_satellite_txvr_handler

7.36 cosmos_satellite_txvr_handler.h

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002
00003 #ifndef ESTTS_COSMOS_SATELLITE_TXVR_HANDLER_H
00004 #define ESTTS_COSMOS_SATELLITE_TXVR_HANDLER_H
00005
00006 #include <thread>
00007 #include "constants.h"
00008 #include "socket_handler.h"
00009 #include "satellite_txvr_cmdtelem_manager.h"
00010
00011 class cosmos_satellite_txvr_handler {
00012     socket_handler * sock;
00013
00014     std::thread cosmos_worker;
00015
00016     satellite_txvr_cmdtelem_manager * satellite_txvr_manager;
00017
00022     [[noreturn]] void satellite_txvr_cosmos_worker();
00023
00031     static std::function<estts::Status(std::string)> get_generic_command_callback_lambda(std::string
       command, socket_handler * sock);
00032
00033 public:
00037     cosmos_satellite_txvr_handler();
00038
00044     estts::Status cosmos_satellite_txvr_init();
00045 };
00046
00047
00048 #endif //ESTTS_COSMOS_SATELLITE_TXVR_HANDLER_H
```

7.37 src/fapi/groundstation_cmdtelem_manager.cpp File Reference

```
#include "helper.h"
#include "groundstation_cmdtelem_manager.h"
```

7.38 groundstation_cmdtelem_manager.cpp

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002
00003 #include "helper.h"
00004 #include "groundstation_cmdtelem_manager.h"
00005
00006 using namespace estts;
00007
00008 std::string groundstation_cmdtelem_manager::schedule_command(std::string command, const
       std::function<Status(
00009         std::string)> &callback) {
00010     if (ti == nullptr) {
00011         SPDLOG_ERROR("Transmission interface not initialized. Was init_session_manager() called?");
00012     }
00013     auto new_command = new waiting_command;
00014     new_command->frame = std::move(command);
00015     new_command->serial_number = generate_serial_number();
00016     new_command->str_callback = callback;
00017     new_command->command = nullptr;
00018     new_command->obj_callback = nullptr;
00019
00020     waiting.push_back(new_command);
00021
00022     SPDLOG_DEBUG("Scheduled new command with serial number {}", new_command->serial_number);
00023     return new_command->serial_number;
00024 }
00025
00026 groundstation_cmdtelem_manager::groundstation_cmdtelem_manager(transmission_interface *ti,
       std::function<estts::Status(
00027         std::string)> telem_callback) {
```

```
00028     this->telem_callback = std::move(telem_callback);
00029     this->ti = ti;
00030
00031     // Create a new thread, pass in dispatch() function and current object
00032     cmdtelem_worker = std::thread(&groundstation_cmdtelem_manager::dispatch, this);
00033     SPDLOG_TRACE("Created dispatch worker thread with ID {}",
      std::hash<std::thread::id>{}(cmdtelem_worker.get_id()));
00034 }
00035
00036 void groundstation_cmdtelem_manager::dispatch() {
00037     std::string message;
00038     for (;;) {
00039 start:
00040         if (!waiting.empty()) {
00041             SPDLOG_TRACE("{} commands in queue", waiting.size());
00042
00043             if (!ti->gstxvr_session_active) {
00044                 // Request a new communication session from the ground station transceiver
00045                 if (ES_OK != this->ti->request_gstxvr_session()) {
00046                     SPDLOG_ERROR("Failed to request new session.");
00047                     goto start; // todo This should probably have a more elegant solution..
00048                 }
00049             }
00050
00051             if(this->ti->gs_transmit(waiting.front()->frame) != estts::ES_OK){
00052                 SPDLOG_TRACE("Failed to transmit.");
00053                 }
00054             else {
00055                 auto telem = this->ti->receive();
00056                 SPDLOG_INFO("Got response from ground station transceiver.");
00057                 if(waiting.front()->str_callback != nullptr){
00058                     waiting.front()->str_callback(telem);
00059                 }
00060                 SPDLOG_TRACE("Command with serial number {} executed successfully.",
      waiting.front()->serial_number);
00061             }
00062             waiting.pop_front();
00063
00064             if (waiting.empty() && ti->gstxvr_session_active) {
00065                 if (ES_OK != this->ti->end_gstxvr_session()) {
00066                     SPDLOG_WARN("Failed to end session rip");
00067                 }
00068                 SPDLOG_INFO("Waiting for more commands");
00069             }
00070         }
00071     }
00072 }
```

7.39 src/fapi/groundstation_cmdtelem_manager.h File Reference

```
#include <functional>
#include <mutex>
#include <thread>
#include <utility>
#include <vector>
#include <queue>
#include "transmission_interface.h"
#include "constants.h"
```

Classes

- class groundstation_cmdtelem_manager

7.40 groundstation_cmdtelem_manager.h

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002
00003 #ifndef ESTTS_GROUNDSTATION_CMDTELEM_MANAGER_H
00004 #define ESTTS_GROUNDSTATION_CMDTELEM_MANAGER_H
00005
00006 #include <functional>
00007 #include <mutex>
00008 #include <thread>
00009 #include <utility>
00010 #include <vector>
00011 #include <queue>
00012 #include "transmission_interface.h"
00013 #include "constants.h"
00014
00015 class groundstation_cmdtelem_manager {
00016 private:
00017     transmission_interface *ti;
00018     std::thread cmdtelem_worker;
00019     std::deque<estts::waiting_command *> waiting;
00020     std::function<estts::Status(std::string)> telem_callback = nullptr;
00021
00025     [[noreturn]] void dispatch();
00026
00027 public:
00032     groundstation_cmdtelem_manager(transmission_interface * ti,
      std::function<estts::Status(std::string)> telem_callback);
00033
00043     std::string schedule_command(std::string command, const std::function<estts::Status(std::string)>&
      callback);
00044 };
00045
00046
00047 #endif //ESTTS_GROUNDSTATION_CMDTELEM_MANAGER_H
```

7.41 src/fapi/obc_session_manager.cpp File Reference

```
#include "helper.h"
#include "obc_session_manager.h"
```

7.42 obc_session_manager.cpp

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002 //
00003 // Created by Hayden Roszell on 1/4/22.
00004 //
00005
00006 #include "helper.h"
00007 #include "obc_session_manager.h"
00008
00009 using namespace estts;
00010
00011 // https://stackoverflow.com/questions/15752659/thread-pooling-in-c11
00012
00013 std::string obc_session_manager::schedule_command(command_object * command,
      std::function<Status(std::vector<telemetry_object *>)> decomp_callback) {
00014     if (ti == nullptr) {
00015         SPDLOG_ERROR("Transmission interface not initialized. Was init_session_manager() called?");
00016     }
00017
00018     auto new_command = new waiting_command;
00019     new_command->command = command;
00020     new_command->serial_number = generate_serial_number();
00021     new_command->obj_callback = std::move(decomp_callback);
00022     new_command->str_callback = nullptr;
00023     new_command->frame = nullptr;
00024
00025     waiting.push_back(new_command);
00026
00027     SPDLOG_DEBUG("Scheduled new command with serial number {}", new_command->serial_number);
00028     return new_command->serial_number;
00029 }
00030
```

```
00031 std::string obc_session_manager::schedule_command(std::string command,
00032                                                   const std::function<estts::Status(std::string)>&
      callback) {
00033     if (ti == nullptr) {
00034         SPDLOG_ERROR("Transmission interface not initialized. Was init_session_manager() called?");
00035     }
00036     auto new_command = new waiting_command;
00037     new_command->frame = std::move(command);
00038     new_command->serial_number = generate_serial_number();
00039     new_command->str_callback = callback;
00040     new_command->command = nullptr;
00041     new_command->obj_callback = nullptr;
00042
00043     waiting.push_back(new_command);
00044
00045     SPDLOG_DEBUG("Scheduled new command with serial number {}", new_command->serial_number);
00046     return new_command->serial_number;
00047 }
00048
00049 obc_session_manager::obc_session_manager(transmission_interface * ti) {
00050     this->ti = ti;
00051     this->init_command_handler(ti);
00052     ti->register_dispatch_function( [this] () {this->dispatch();} );
00053 }
00054
00055 obc_session_manager::obc_session_manager(transmission_interface * ti,
       std::function<estts::Status(std::string)> telem_callback) {
00056     this->telem_callback = std::move(telem_callback);
00057
00058     this->ti = ti;
00059     this->init_command_handler(this->ti);
00060     ti->set_telem_callback(this->telem_callback);
00061     // Create a new thread, pass in dispatch() function and current object
00062     ti->register_dispatch_function( [this] () {this->dispatch();} );
00063 }
00064
00065 void obc_session_manager::await_completion() {
00066     // If the thread is joinable (IE it's active), join the thread
00067     // Join blocks until the thread returns.
00068     if (session_worker.joinable())
00069         session_worker.join();
00070 }
00071
00072 obc_session_manager::~obc_session_manager() {
00073     await_completion();
00074     delete ti;
00075     if (!dispatched.empty())
00076         for (auto &i : dispatched)
00077             delete i;
00078     if (!waiting.empty())
00079         for (auto &i : waiting)
00080             delete i;
00081 }
00082
00083 void obc_session_manager::dispatch() {
00084     using namespace std::this_thread; // sleep_for, sleep_until
00085     using namespace std::chrono; // nanoseconds, system_clock, seconds
00086     for (;;) {
00087 start:
00088         if (!waiting.empty()) {
00089             SPDLOG_TRACE("{} commands in queue", waiting.size());
00090             if (!ti->obc_session_active) {
00091                 // Request a new communication session from EagleSat II
00092                 if (ES_OK != this->ti->request_obc_session()) {
00093                     SPDLOG_ERROR("Failed to request new session.");
00094                     goto start; // todo This should probably have a more elegant solution..
00095                 }
00096             }
00097
00098             // After execute is called, the session is in progress. Set this state before, so that
       abstracted objects
00099             // stay up to date.
00100             SPDLOG_TRACE("Session status: {}", ti->obc_session_active);
00101             if (ES_OK != this->execute(waiting.front())) {
00102                 SPDLOG_WARN("Failed to execute command with serial number {}",
       waiting.front()->serial_number);
00103             } else {
00104                 SPDLOG_INFO("Command executed successfully");
00105             }
00106             waiting.pop_front();
00107
00108             if (waiting.empty() && ti->obc_session_active) {
00109                 if (ES_OK != this->ti->end_obc_session(ax25::END_SESSION_FRAME)) {
00110                     SPDLOG_WARN("Failed to end session rip");
00111                 }
00112                 SPDLOG_INFO("Waiting for more commands");
00113             }
```

```
00114
00115         } else {
00116
00117             // Handle stream
00118             auto stream = ti->nonblock_receive();
00119             if (!stream.empty() && this->telem_callback != nullptr)
00120                 telem_callback(stream);
00121         }
00122         if (!ti->satellite_in_range) {
00123             SPDLOG_DEBUG("OBC Dispatch worker - Detected satellite is outside range, exiting.");
00124             return;
00125         }
00126     }
00127 }
```

7.43 src/fapi/obc_session_manager.h File Reference

```
#include <functional>
#include <mutex>
#include <thread>
#include <utility>
#include <vector>
#include <queue>
#include "constants.h"
#include "command_handler.h"
```

Classes

- class obc_session_manager

7.44 obc_session_manager.h

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002 //
00003 // Created by Hayden Roszell on 1/4/22.
00004 //
00005
00006 #ifndef ESTTS_COMMAND_SCHEDULER_H
00007 #define ESTTS_COMMAND_SCHEDULER_H
00008
00009 #include <functional>
00010 #include <mutex>
00011 #include <thread>
00012 #include <utility>
00013 #include <vector>
00014 #include <queue>
00015 #include "constants.h"
00016 #include "command_handler.h"
00017
00018 // Idea: schedule command takes in a function pointer. Then, a new command scheduler
00019 // structure should be created, containing a serial number, the function pointer,
00020 // and an Status containing the completion status. When a command is scheduled to run with
00021 // the schedule_command() function, the unique serial number is returned. When a job is scheduled and
00022 // run, it is popped off the queue, and the status variable associated with the structure is
00023 // updated. Finally, the structure is pushed onto another vector containing completed jobs.
00024 // the number of completed jobs that are stored is limited to a predefined number.
00025 // Finally, there should be a function that takes argument for a serial number associated
00026 // with a command, and returns the completion status.
00027
00028 class obc_session_manager : virtual public command_handler {
00029 private:
00030     transmission_interface *ti;
00031     std::thread session_worker;
00032     std::deque<estts::waiting_command *> waiting;
00033     std::function<estts::Status(std::string)> telem_callback = nullptr;
```

```
00034
00040     void dispatch();
00041
00042 public:
00047     explicit obc_session_manager(transmission_interface * ti);
00048
00054     explicit obc_session_manager(transmission_interface * ti,
      std::function<estts::Status(std::string)> telem_callback);
00055
00059     ~obc_session_manager();
00060
00070     std::string schedule_command(std::string command, const std::function<estts::Status(std::string)>&
      callback);
00071
00082     std::string schedule_command(estts::command_object * command,
00083                                  std::function<estts::Status(std::vector<estts::telemetry_object *>)>
      decomp_callback);
00084
00089     void await_completion();
00090 };
00091
00092
00093 #endif //ESTTS_COMMAND_SCHEDULER_H
```

7.45 src/fapi/satellite_txvr_cmdtelem_manager.cpp File Reference

```
#include "helper.h"
#include "satellite_txvr_cmdtelem_manager.h"
```

7.46 satellite_txvr_cmdtelem_manager.cpp

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002
00003 #include "helper.h"
00004 #include "satellite_txvr_cmdtelem_manager.h"
00005
00006 using namespace estts;
00007
00008 satellite_txvr_cmdtelem_manager::satellite_txvr_cmdtelem_manager(transmission_interface *ti,
00009                                                                  std::function<Status(
00010                                                                         std::string)> telem_callback)
      {
00011     this->telem_callback = std::move(telem_callback);
00012     this->ti = ti;
00013
00014     // Create a new thread, pass in dispatch() function and current object
00015     cmdtelem_worker = std::thread(&satellite_txvr_cmdtelem_manager::dispatch, this);
00016     SPDLOG_TRACE("Created dispatch worker thread with ID {}",
      std::hash<std::thread::id>{}(cmdtelem_worker.get_id()));
00017 }
00018
00019 std::string satellite_txvr_cmdtelem_manager::schedule_command(std::string command, const
      std::function<estts::Status(
00020         std::string)> &callback) {
00021     if (ti == nullptr) {
00022         SPDLOG_ERROR("Transmission interface not initialized. Was init_session_manager() called?");
00023     }
00024     auto new_command = new waiting_command;
00025     new_command->frame = std::move(command);
00026     new_command->serial_number = generate_serial_number();
00027     new_command->str_callback = callback;
00028     new_command->command = nullptr;
00029     new_command->obj_callback = nullptr;
00030
00031     waiting.push_back(new_command);
00032
00033     SPDLOG_DEBUG("Scheduled new command with serial number {}", new_command->serial_number);
00034     return new_command->serial_number;
00035 }
00036
00037 void satellite_txvr_cmdtelem_manager::dispatch() {
00038     for (;;) {
```

```
00039         if (!waiting.empty()) {
00040             SPDLOG_TRACE("{} commands in queue", waiting.size());
00041
00042             // todo handle the command found at waiting.front()
00043             // todo make sure to call the callback with the response
00044         }
00045     }
00046 }
```

7.47 src/fapi/satellite_txvr_cmdtelem_manager.h File Reference

```
#include <functional>
#include <mutex>
#include <thread>
#include <utility>
#include <vector>
#include <queue>
#include "transmission_interface.h"
#include "constants.h"
```

Classes

- class satellite_txvr_cmdtelem_manager

7.48 satellite_txvr_cmdtelem_manager.h

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002
00003 #ifndef ESTTS_SATELLITE_TXVR_CMDTELEM_MANAGER_H
00004 #define ESTTS_SATELLITE_TXVR_CMDTELEM_MANAGER_H
00005
00006 #include <functional>
00007 #include <mutex>
00008 #include <thread>
00009 #include <utility>
00010 #include <vector>
00011 #include <queue>
00012 #include "transmission_interface.h"
00013 #include "constants.h"
00014
00015 class satellite_txvr_cmdtelem_manager {
00016 private:
00017     transmission_interface *ti;
00018     std::thread cmdtelem_worker;
00019     std::deque<estts::waiting_command *> waiting;
00020     std::function<estts::Status(std::string)> telem_callback = nullptr;
00021
00025     [[noreturn]] void dispatch();
00026
00027 public:
00032     satellite_txvr_cmdtelem_manager(transmission_interface * ti,
       std::function<estts::Status(std::string)> telem_callback);
00033
00043     std::string schedule_command(std::string command, const std::function<estts::Status(std::string)>&
       callback);
00044 };
00045
00046
00047 #endif //ESTTS_SATELLITE_TXVR_CMDTELEM_MANAGER_H
```

7.49 src/ti/esttc.cpp File Reference

```
#include <chrono>
#include <thread>
#include <string>
#include <sstream>
#include <algorithm>
#include <vector>
#include "esttc.h"
```

7.50 esttc.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/10/21.
00003 //
00004
00005 #include <chrono>
00006 #include <thread>
00007 #include <string>
00008 #include <sstream>
00009 #include <algorithm>
00010 #include <vector>
00011 #include "esttc.h"
00012
00013 using std::stringstream;
00014 using std::string;
00015 using namespace std::this_thread; // sleep_for, sleep_until
00016 using namespace std::chrono; // nanoseconds, system_clock, seconds
00017 using namespace estts;
00018 using namespace estts::endurosat;
00019
00024 esttc::esttc() {
00025     esttc_symbols = new estts::endurosat::esttc_const;
00026 }
00027
00028 // 10.1 - STATUS CONTROL WORD ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
00029
00034 estts::Status esttc::default_mode() {
00035     return write_scw(esttc_symbols->default_mode);
00036 }
00037
00043 estts::Status esttc::write_scw(uint16_t scw_command) {
00044     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00045     string response;
00046
00047     // TODO - Handle when the scw_command > stopper
00048     string command_body = esttc_symbols->scw_body[scw_command];
00049
00050     return_status = build_esttc_command(
00051             esttc_symbols->METHOD_WRITE,
00052             esttc_symbols->CMD_SCW,
00053             response,
00054             command_body);
00055     if (return_status == estts::ES_OK && response.length() >= 7) {
00056         std::string scw_resp;
00057         if (cache.str().find("+PIPE") == std::string::npos) {
00058             do{}
00059             while ((scw_resp = read_serial_s()).empty());
00060         } else {
00061             scw_resp = cache.str();
00062         }
00063
00064
00065         if (scw_resp.substr(3, 4) == "C3C3") {
00066             SPDLOG_INFO("Successfully commanded UHF to enter Bootloader mode from from Application
      mode");
00067         } else if (scw_resp.substr(3, 4) == "8787") {
00068             SPDLOG_INFO("Successfully commanded UHF to enter Application mode from from Bootloader
      mode");
00069         } else if (scw_resp.find("+PIPE") != string::npos) {
00070             SPDLOG_INFO("Successfully commanded UHF to enter PIPE mode");
00071         } else {
00072             SPDLOG_INFO("SCW write failure");
```

```
            return estts::ES_UNSUCCESSFUL;
        }
    }
    return return_status;
}

estts::Status esttc::read_scw(string &RSSI, string &dvc_addr, string &rst_ctr, string &scw) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;

    return_status = build_esttc_command(
            esttc_symbols->METHOD_READ,
            esttc_symbols->CMD_SCW,
            response);

    if (return_status == estts::ES_SUCCESS) {
        RSSI = response.substr(3, 2); // [RR]
        dvc_addr = response.substr(5, 2); // [AA]
        rst_ctr = response.substr(7, 2); // [BB]
        scw = response.substr(9, 4); // [WWWW]
    }

    return return_status;
}

// 10.2 - RADIO FREQUENCY CONFIGURATION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
estts::Status esttc:: esttc::write_radio_freq_config(const string& frac, const string& div) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;
    string command_body;

    command_body += frac;
    command_body += div;

    return_status = build_esttc_command(
            esttc_symbols->METHOD_WRITE,
            esttc_symbols->CMD_RADIO_FREQ_CONFIG,
            response,
            command_body);

    return return_status;
}

estts::Status esttc::read_radio_freq(string &RSSI, string &frac, string &div) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;

    return_status = build_esttc_command(
            esttc_symbols->METHOD_READ,
            esttc_symbols->CMD_RADIO_FREQ_CONFIG,
            response);

    if (return_status == estts::ES_SUCCESS) {
        RSSI = response.substr(3, 2);
        frac = response.substr(5, 6);
        div = response.substr(11, 2);
    }

    return return_status;
}

// 10.3 - READ UPTIME ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
estts::Status esttc::read_uptime(string &RSSI, string &uptime) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;

    return_status = build_esttc_command(
            esttc_symbols->METHOD_READ,
            esttc_symbols->CMD_READ_UPTIME,
            response);

    if (return_status == estts::ES_SUCCESS) {
        RSSI = response.substr(3, 2);
        uptime = response.substr(5, 8);
    }

    return return_status;
}

// 10.4 - READ NUMBER OF TRANSMITTED PACKETS ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
estts::Status esttc::read_trans_pckts(string &RSSI, string &pckt_num) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;

```

```
    return_status = build_esttc_command(
            esttc_symbols->METHOD_READ,
            esttc_symbols->CMD_READ_TRANS_PCKTS,
            response);

    if (return_status == estts::ES_SUCCESS) {
        RSSI = response.substr(3, 2);
        pckt_num = response.substr(5, 8);
    }

    return return_status;
}

// 10.6 - READ NUMBER OF TRANSMITTED PACKETS WITH A CRC ERROR
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

estts::Status esttc::read_trans_pckts_crc(string &RSSI, string &pckt_num) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;

    return_status = build_esttc_command(
            esttc_symbols->METHOD_READ,
            esttc_symbols->CMD_READ_TRANS_PCKTS_CRC,
            response);

    if (return_status == estts::ES_SUCCESS) {
        RSSI = response.substr(3, 2);
        pckt_num = response.substr(5, 8);
    }

    return return_status;
}

// 10.8 - BEACON MESSAGE TRANSMISSION PERIOD CONFIGURATION
 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

estts::Status esttc::write_bcn_trans_period(const string &period) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;
    string command_body;

    command_body += "0000";
    command_body += period;

    return_status = build_esttc_command(
            esttc_symbols->METHOD_WRITE,
            esttc_symbols->CMD_BCN_MSG_TRANS_CONFIG,
            response,
            command_body);

    return return_status;
}

estts::Status esttc::read_bcn_trans_period(string &RSSI, string &period) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;

    return_status = build_esttc_command(
            esttc_symbols->METHOD_READ,
            esttc_symbols->CMD_BCN_MSG_TRANS_CONFIG,
            response);

    if (return_status == estts::ES_SUCCESS) {
        RSSI = response.substr(3, 2);
        period = response.substr(9, 4);
    }

    return return_status;
}

// 10.10 - RESTORE DEFAULT VALUES ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

estts::Status esttc::write_res_default_vals() {
        estts::Status return_status = estts::ES_UNSUCCESSFUL;
        string response;
        string command_body;

        command_body += "";

        return_status = build_esttc_command(
                esttc_symbols->METHOD_WRITE,
                esttc_symbols->CMD_RESTORE,
                response,
                command_body);

        return return_status;
}
```

```
00303
00304 // 10.31 - I2C PULL-UP RESISTORS CONFIGURATION READ/WRITE
       ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
00305
00311 estts::Status esttc::write_i2c_resist_config(const string &resistor_config) {
00312     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00313     string response;
00314     string command_body;
00315
00316     command_body += resistor_config;
00317
00318     return_status = build_esttc_command(
00319             esttc_symbols->METHOD_WRITE,
00320             esttc_symbols->CMD_I2C_RESIST_CONFIG,
00321             response,
00322             command_body);
00323
00324     return return_status;
00325 }
00326
00332 estts::Status esttc::read_i2c_resist_config(string &selected_resistor) {
00333     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00334     string response;
00335
00336     return_status = build_esttc_command(
00337             esttc_symbols->METHOD_READ,
00338             esttc_symbols->CMD_I2C_RESIST_CONFIG,
00339             response);
00340
00341     if (return_status == estts::ES_SUCCESS) {
00342         selected_resistor = response.substr(3, 2);
00343     }
00344
00345     return return_status;
00346 }
00347
00348 // 10.12 - ENABLING/DISABLING RADIO PACKET CRC16 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
00349
00355 estts::Status esttc::write_radio_crc16(const string &mode) {
00356     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00357     string response;
00358     string command_body;
00359
00360     command_body += mode;
00361
00362     return_status = build_esttc_command(
00363             esttc_symbols->METHOD_WRITE,
00364             esttc_symbols->CMD_ENABLE_DISABLE_RADIO_CRC,
00365             response,
00366             command_body);
00367
00368     return return_status;
00369 }
00370
00376 estts::Status esttc::read_radio_crc16(string &mode) {
00377     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00378     string response;
00379
00380     return_status = build_esttc_command(
00381             esttc_symbols->METHOD_READ,
00382             esttc_symbols->CMD_ENABLE_DISABLE_RADIO_CRC,
00383             response);
00384
00385     if (return_status == estts::ES_SUCCESS) {
00386         mode = response.substr(4, 1);
00387     }
00388
00389     return return_status;
00390 }
00391
00392 // 10.14 - ENABLING/DISABLING AUTOMATIC AX.25 DECODING
       ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
00393
00399 estts::Status esttc::write_config_ax25_decode(const string &config_bit) {
00400     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00401     string response;
00402     string command_body;
00403
00404     command_body += '0';
00405     command_body += config_bit;
00406
00407     return_status = build_esttc_command(
00408             esttc_symbols->METHOD_WRITE,
00409             esttc_symbols->CMD_AUTO_AX25_DECODE,
00410             response,
00411             command_body);
00412
```

```
    return return_status;
}

estts::Status esttc::read_config_ax25_decode(string &config_bit) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;

    return_status = build_esttc_command(
            esttc_symbols->METHOD_READ,
            esttc_symbols->CMD_AUTO_AX25_DECODE,
            response);

    if (return_status == estts::ES_SUCCESS) {
        config_bit = response.substr(4, 1);
    }

    return return_status;
}

// 10.16 - UHF ANTENNA RELEASE CONFIGURATION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

estts::Status esttc::write_ant_release_config(const string &ant_config) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;
    string command_body;

    command_body += ant_config;

    return_status = build_esttc_command(
            esttc_symbols->METHOD_WRITE,
            esttc_symbols->CMD_ANT_RELEASE_CONFIG,
            response,
            command_body);

    return return_status;
}

estts::Status esttc::read_ant_release_config(string &ant_config) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;

    return_status = build_esttc_command(
            esttc_symbols->METHOD_READ,
            esttc_symbols->CMD_ANT_RELEASE_CONFIG,
            response);

    if (return_status == estts::ES_SUCCESS) {
        ant_config = response.substr(3, 4);
    }

    return return_status;
}

// 10.18 - LOW POWER MODE ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

estts::Status esttc::write_low_pwr_mode() {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;
    string command_body;

    command_body = "";

    return_status = build_esttc_command(
            esttc_symbols->METHOD_WRITE,
            esttc_symbols->CMD_LOW_PWR_MODE,
            response,
            command_body);

    return return_status;
}

estts::Status esttc::read_low_pwr_mode(string &power_mode) {
    estts::Status return_status = estts::ES_UNSUCCESSFUL;
    string response;

    return_status = build_esttc_command(
            esttc_symbols->METHOD_READ,
            esttc_symbols->CMD_LOW_PWR_MODE,
            response);

    if (return_status == estts::ES_SUCCESS) {
        power_mode = response.substr(3, 2);
    }

    return return_status;
}
```

```
00524 // 10.20 - SOURCE CALL SIGN ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
00525
00531 estts::Status esttc::write_src_call_sign(const string &call_sign) {
00532     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00533     string response;
00534     string command_body;
00535
00536     command_body += call_sign;
00537
00538     return_status = build_esttc_command(
00539             esttc_symbols->METHOD_WRITE,
00540             esttc_symbols->CMD_SRC_CALL_SIGN,
00541             response,
00542             command_body);
00543
00544     return return_status;
00545 }
00546
00552 estts::Status esttc::read_src_call_sign(string &call_sign) {
00553     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00554     string response;
00555
00556     return_status = build_esttc_command(
00557             esttc_symbols->METHOD_READ,
00558             esttc_symbols->CMD_SRC_CALL_SIGN,
00559             response);
00560
00561     if (return_status == estts::ES_SUCCESS) {
00562         call_sign = response.substr(3, 6);
00563     }
00564
00565     return return_status;
00566 }
00567
00568 // 10.24 - READ DEVICE PAYLOAD SIZE ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
00569
00575 estts::Status esttc::read_dvc_payload_size(string &payload_size) {
00576     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00577     string response;
00578
00579     return_status = build_esttc_command(
00580             esttc_symbols->METHOD_READ,
00581             esttc_symbols->CMD_READ_DVC_PAYLOAD,
00582             response);
00583
00584     if (return_status == estts::ES_SUCCESS) {
00585         payload_size = response.substr(3, 6);
00586     }
00587
00588     return return_status;
00589 }
00590
00591 // 10.26 - DEVICE ADDRESS CONFIGURATION ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
00592
00598 estts::Status esttc::write_dvc_addr_config(const string &new_addr) {
00599     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00600     string response;
00601     string command_body;
00602
00603     command_body += new_addr;
00604
00605     return_status = build_esttc_command(
00606             esttc_symbols->METHOD_WRITE,
00607             esttc_symbols->CMD_DVC_ADDR_CONFIG,
00608             response,
00609             command_body);
00610
00611     return return_status;
00612 }
00613
00614 // 10.28 - RADIO TRANSCEIVER PROPERTY CONFIGURATION
       ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
00615
00624 estts::Status esttc::write_radio_trans_prop_config(const string &prop_group, const string &bytes,
       const string &offset, const string &data) {
00625     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00626     string response;
00627     string command_body;
00628     uint16_t data_size;
00629     stringstream hexToDec;
00630
00631     // Convert hex string to decimal integer
00632     hexToDec « std::hex « bytes;
00633     hexToDec » data_size;
00634
00635     // If the data exceeds the number of bytes given in the "bytes" parameter, return
00636     if ((data_size * 2) < data.length()) {
```

```
00637         return estts::ES_BAD_OPTION;
00638     }
00639
00640     command_body += prop_group;
00641     command_body += bytes;
00642     command_body += offset;
00643     command_body += data;
00644
00645     return_status = build_esttc_command(
00646             esttc_symbols->METHOD_WRITE,
00647             esttc_symbols->CMD_RADIO_TRANS_PROP_CONFIG,
00648             response,
00649             command_body);
00650
00651     return return_status;
00652 }
00653
00662 estts::Status esttc::read_radio_trans_prop_config(const string &prop_group, const string &bytes, const
       string &offset, string &data) {
00663     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00664     string response;
00665     string command_body;
00666     uint16_t data_size;
00667     stringstream hexToDec;
00668
00669     command_body += prop_group;
00670     command_body += bytes;
00671     command_body += offset;
00672
00673     return_status = build_esttc_command(
00674             esttc_symbols->METHOD_READ,
00675             esttc_symbols->CMD_RADIO_TRANS_PROP_CONFIG,
00676             response,
00677             command_body);
00678
00679     // Convert hex string to decimal integer
00680     hexToDec « std::hex « bytes;
00681     hexToDec » data_size;
00682
00683     if (return_status == estts::ES_SUCCESS) {
00684         // Check if the number of given size of data is in range of response
00685         if (response.length() >= (3 + (data_size * 2))) {
00686             data = response.substr(3, (data_size * 2));
00687         } else {
00688             return_status == estts::ES_BAD_OPTION;
00689         }
00690     }
00691
00692     return return_status;
00693 }
00694
00695 // 10.30 - FIRMWARE UPDATE ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
00696
00702 estts::Status esttc::update_firmware(const string &all_lines) {
00703     estts::Status update_status = estts::ES_UNSUCCESSFUL;
00704
00705     // TODO - Write the first two steps of Firmware Update
00706     //Step 1
00707     // Read Secure Mode
00708
00709     //Step 2
00710     // seedkey = Down_seedkey XOR DOWNLINK_XOR
00711     // Up_seedkey = seedkey XOR UPLINK_XOR
00712     // Write secure mode (Up_seedkey)
00713     // if ok, continue, else stop
00714
00715     // prelim setup
00716     const size_t num_of_lines = std::count(all_lines.begin(), all_lines.end(), '\n');
00717
00718     if (num_of_lines <= 0) {
00719         return estts::ES_BAD_OPTION;
00720     }
00721
00722     const std::vector<string> lines = split_lines(all_lines, num_of_lines);
00723
00724     // Step 3
00725     update_status = update_firmware_sequence(lines.front());
00726
00727     if (update_status == estts::ES_MEMORY_ERROR) {
00728         return estts::ES_MEMORY_ERROR;
00729     }
00730     // Step 4
00731     if (num_of_lines > 2) {
00732         for (const string& line : lines) {
00733             if (line != lines.front() && line != lines.back()) {
00734                 update_status = update_firmware_sequence(line);
00735             }
```

```
00736         }
00737     }
00738
00739     //Step 5
00740     update_status = update_firmware_sequence(lines.back());
00741
00742     if (update_status == estts::ES_SUCCESS) {
00743         SPDLOG_INFO("Firmware Update Successful");
00744     } else {
00745         spdlog::error("Firmware Update Unsuccessful");
00746     }
00747
00748     // If "ERR" or "ERR+FA" was received throughout the process of the update, DANGER, contact
      endurosat support
00749     return update_status;
00750 }
00751
00757 estts::Status esttc::update_firmware_sequence(const string &one_line) {
00758     // TODO - Double check this whole function. Too tired to check now
00759
00760     estts::Status return_status = estts::ES_UNSUCCESSFUL;
00761     string response;
00762     string command_body;
00763
00764     command_body = one_line;
00765
00766     return_status = build_esttc_command(
00767             esttc_symbols->METHOD_FIRMWARE_UPDATE,
00768             esttc_symbols->CMD_FRMWR_UPDATE,
00769             response,
00770             command_body);
00771
00772     if (return_status == estts::ES_UNSUCCESSFUL && response.substr(0, 6) == "ERR+FB") {
00773         return_status = estts::ES_MEMORY_ERROR;
00774     }
00775
00776     return return_status;
00777 }
00778
00785 std::vector<string> esttc::split_lines(const string &all_lines, size_t num_of_lines) {
00786     string::size_type prev_line = 0;
00787     string::size_type new_line = -1;
00788     std::vector<string> lines(num_of_lines);
00789
00790     for (int i = 0; i < num_of_lines; ++i) {
00791         prev_line += new_line + 1;
00792         new_line = all_lines.substr(prev_line).find('\n');
00793         lines.push_back(all_lines.substr(prev_line, new_line));
00794     }
00795
00796     return lines;
00797 }
00798
00807 estts::Status esttc::build_esttc_command(const char method, const char *command_code, string
      &response, const string& body) {
00808     estts::Status return_status = estts::ES_UNINITIALIZED;
00809     estts::Status serial_status = estts::ES_UNSUCCESSFUL;
00810     stringstream command;
00811
00812     /* Build ESTTC command*/
00813     command « esttc_symbols->HEADER;
00814     command « method;
00815     command « esttc_symbols->ADDRESS;
00816     command « command_code;
00817
00818     if (method == esttc_symbols->METHOD_WRITE ||
00819         esttc_symbols->METHOD_FIRMWARE_UPDATE ||
00820         command_code == esttc_symbols->CMD_RADIO_TRANS_PROP_CONFIG) {
00821         command « body;
00822     }
00823     //command « ' ' « calculate_crc32(command.str());
00824     command « esttc_symbols->END;
00825
00826     /*Attempt to transmit ESTTC command */
00827     serial_status = this->write_serial_s(command.str());
00828
00829     if (serial_status == estts::ES_OK) {
00830         do {}
00831         while ((response = read_serial_s()).empty());
00832
00833         // TODO - Make sure this covers ALL cases
00834         if (response.length() >= 7 && response.substr(0, 7) == "ERR+VAL") {
00835             return_status = estts::ES_BAD_OPTION;
00836             spdlog::error("Invalid  ESTTC command input data: {}", command.str());
00837         } else if (response.length() >= 3 && response.substr(0, 3) == "ERR") {
00838             return_status = estts::ES_UNSUCCESSFUL;
00839             spdlog::error("Failed to transmit  ESTTC command: {}", command.str());
```

```
00840         } else if (response.length() >= 2 && response.substr(0, 2) == "OK") {
00841             return_status = estts::ES_OK;
00842             SPDLOG_INFO("Successfully transmitted ESTTC command: {}", command.str());
00843         } else {
00844             return_status = estts::ES_UNINITIALIZED;
00845             SPDLOG_INFO("Unhandled ESTTC command response ...\ncommand: {}\nanswer: {}",
      command.str(), response);
00846         }
00847     } else {
00848         return_status = serial_status;
00849     }
00850
00851     return return_status;
00852 }
00853
00859 std::string esttc::calculate_crc32(string string) {
00860     // TODO Taylor's task for sprint #3
00861     return string;
00862 }
00863
00864 esttc::~esttc() {
00865     delete esttc_symbols;
00866 }
```

7.51 src/ti/esttc.h File Reference

```
#include "serial_handler.h"
#include "constants.h"
```

Classes

- class esttc

7.52 esttc.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/10/21.
00003 //
00004
00005 #ifndef ESTTS_ESTTC_H
00006 #define ESTTS_ESTTC_H
00007
00008 #include "serial_handler.h"
00009 #include "constants.h"
00010
00011 class esttc : virtual public serial_handler {
00012 private:
00013     estts::endurosat::esttc_const *esttc_symbols;
00014
00015     estts::Status build_esttc_command(char method, const char *command_code,  std::string &response,
      const std::string& body = "");
00016
00017     static std::vector<std::string> split_lines(const std::string &all_lines, size_t num_of_lines);
00018 protected:
00019 public:
00020
00021     // 10.1 - STATUS CONTROL WORD (SCW)
00022     estts::Status default_mode();
00023     estts::Status write_scw(uint16_t scw_command);
00024     estts::Status read_scw(std::string &RSSI, std::string &dvc_addr, std::string &rst_ctr, std::string
     &scw);
00025     // 10.2 - RADIO FREQUENCY CONFIGURATION
00026     estts::Status write_radio_freq_config(const std::string& frac = "76620F41", const std::string& div
      = "41");
00027     estts::Status read_radio_freq(std::string &RSSI, std::string &frac, std::string &div);
00028     // 10.3 - READ UPTIME
00029     estts::Status read_uptime(std::string &RSSI, std::string &uptime);
00030     // 10.4 - READ NUMBER OF TRANSMITTED PACKETS
```

```
00031     estts::Status read_trans_pckts(std::string &RSSI, std::string &pckt_num);
00032     // 10.6 - READ NUMBER OF TRANSMITTED PACKETS WITH A CRC ERROR
00033     estts::Status read_trans_pckts_crc(std::string &RSSI, std::string &pckt_num);
00034     // 10.8 - BEACON MESSAGE TRANSMISSION PERIOD CONFIGURATION
00035     estts::Status write_bcn_trans_period(const std::string &period = "003C");
00036     estts::Status read_bcn_trans_period(std::string &RSSI, std::string &period);
00037     // 10.10 - RESTORE DEFAULT VALUES
00038     estts::Status write_res_default_vals();
00039     // 10.31 - I2C PULL-UP RESISTORS CONFIGURATION READ/WRITE
00040     estts::Status write_i2c_resist_config(const std::string &resistor_config);
00041     estts::Status read_i2c_resist_config(std::string &selected_resistor);
00042     // 10.12 - ENABLING/DISABLING RADIO PACKET CRC16
00043     estts::Status write_radio_crc16(const std::string &mode);
00044     estts::Status read_radio_crc16(std::string &mode);
00045     // 10.14 - ENABLING/DISABLING AUTOMATIC AX.25 DECODING
00046     estts::Status write_config_ax25_decode(const std::string &config_bit); // Consider replacing the
      param with a char type
00047     estts::Status read_config_ax25_decode(std::string &config_bit);
00048     // 10.16 - UHF ANTENNA RELEASE CONFIGURATION
00049     estts::Status write_ant_release_config(const std::string &ant_config);
00050     estts::Status read_ant_release_config(std::string &ant_config);
00051     // 10.18 - LOW POWER MODE
00052     estts::Status write_low_pwr_mode();
00053     estts::Status read_low_pwr_mode(std::string &mode);
00054     // 10.20 - SOURCE CALL SIGN
00055     estts::Status write_src_call_sign(const std::string &call_sign = "XX0UHF");
00056     estts::Status read_src_call_sign(std::string &call_sign);
00057     // 10.24 - READ DEVICE PAYLOAD SIZE
00058     estts::Status read_dvc_payload_size(std::string &payload_size);
00059     // 10.26 - DEVICE ADDRESS CONFIGURATION
00060     estts::Status write_dvc_addr_config(const std::string &new_addr = "22");
00061     // 10.28 - RADIO TRANSCEIVER PROPERTY CONFIGURATION
00062     estts::Status write_radio_trans_prop_config(const std::string &prop_group, const std::string
      &bytes,
00063
      const std::string &offset, const std::string &data);
00064     estts::Status read_radio_trans_prop_config(const std::string &prop_group, const std::string
      &bytes,
00065
      const std::string &offset, std::string &data);
00066     // 10.30 - FIRMWARE UPDATE
00067     estts::Status update_firmware(const std::string &all_lines);
00068     estts::Status update_firmware_sequence(const std::string &one_line);
00069
00070     static std::string calculate_crc32(std::string string);
00071
00072     esttc();
00073
00074     ~esttc();
00075 };
00076
00077
00078 #endif //ESTTS_ESTTC_H
```

7.53 src/ti/posix_serial.cpp File Reference

```
#include "posix_serial.h"
#include <fcntl.h>
#include <cerrno>
#include <unistd.h>
#include <sys/ioctl.h>
#include <termios.h>
#include <sys/types.h>
```

7.54 posix_serial.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 3/17/22.
00003 //
00004
```

```
00005 #include "posix_serial.h"
00006
00007 #include <fcntl.h> // Contains file controls like O_RDWR
00008 #include <cerrno> // Error integer and strerror() function
00009 #include <unistd.h> // write(), read(), close()
00010 #include <sys/ioctl.h>
00011 #include <termios.h> // Contains POSIX terminal control definitions
00012 #include <sys/types.h>
00013 #include "posix_serial.h"
00014
00015
00016 using namespace estts;
00017
00018 // Reference https://www.cmrr.umn.edu/~strupp/serial.html
00019
00027 posix_serial::posix_serial(const char *port, int baud) {
00028     this->baud = baud;
00029     this->port = port;
00030     serial_port = -1; // Serial port initialized in open_port() method
00031     // Attempt to open serial port
00032
00033 #ifdef __TI_DEV_MODE__
00034     SPDLOG_DEBUG("Initializing transmission interface in dev mode. No serial port will be opened.");
00035     serial_port = 0;
00036 #else
00037     SPDLOG_DEBUG("Opening serial port {} with {} baud", port, baud);
00038     if (ES_OK != open_port()) {
00039         SPDLOG_ERROR("Failed to open serial port {}", port);
00040         throw std::runtime_error("Failed to open serial port.");
00041     }
00042
00043     // Attempt to initialize serial port
00044     if (ES_OK != initialize_serial_port()) {
00045         SPDLOG_ERROR("Failed to open serial port {}", port);
00046         throw std::runtime_error("Failed to initialize serial port.");
00047     }
00048 #endif
00049 }
00050
00055 estts::Status posix_serial::initialize_serial_port() {
00056     struct termios tty{};
00057 #ifndef __ESTTS_OS_LINUX__
00058     if (tcgetattr(serial_port, &tty) != 0) {
00059         SPDLOG_ERROR("Error %i from tcgetattr: %s\n", errno, strerror(errno));
00060         SPDLOG_INFO("Did you mean to use TI Dev Mode? See README.md");
00061         return ES_UNSUCCESSFUL;
00062     }
00063     // Initialize Terminos structure
00064     tty.c_cflag &= ~PARENB;  // Disable parity bit (IE clear parity bit)
00065     tty.c_cflag &= ~CSTOPB;  // 1 stop bit (IE clear stop field)
00066     tty.c_cflag &= ~CSIZE;   // Clear size bit
00067     tty.c_cflag |= CS8;      // 8 data bits
00068     tty.c_cflag &= ~CRTSCTS; // Disable hardware flow control
00069     tty.c_lflag &= ~ICANON;  // Disable UNIX Canonical mode (\n != terminator)
00070     tty.c_lflag &= ~ECHO;    // Disable echo
00071     tty.c_lflag &= ~ECHOE;   // Disable erasure
00072     tty.c_lflag &= ~ECHONL;  // Disable new-line echo
00073     tty.c_lflag &= ~ISIG;    // Disable interpretation of INTR, QUIT and SUSP
00074     tty.c_iflag &= ~(IXON | IXOFF | IXANY); // Disable software flow control
00075     tty.c_iflag &= ~(IGNBRK | BRKINT | PARMRK | ISTRIP | INLCR | IGNCR |
00076                      ICRNL); // Disable any special handling of received bytes
00077
00078     // Set baud rate
00079     cfsetispeed(&tty, baud);
00080     cfsetospeed(&tty, baud);
00081
00082     // Set non-blocking
00083     // todo make separate constructor determine if serial should read blocking/nonblocking
00084     fcntl(serial_port, F_SETFL, FNDELAY);
00085
00086     // Save tty settings, also check for error
00087     if (tcsetattr(serial_port, TCSANOW, &tty) != 0) {
00088         SPDLOG_ERROR("Error {} from tcsetattr: {}", errno, strerror(errno));
00089         return ES_UNSUCCESSFUL;
00090     }
00091 #else
00092         SPDLOG_INFO("Initializing serial handler with compatibility for Linux kernel");
00093     if (tcgetattr(serial_port, &tty) != 0) {
00094         spdlog::error("Error %i from tcgetattr: %s\n", errno, strerror(errno));
00095         SPDLOG_INFO("Did you mean to use TI Dev Mode? See README.md");
00096         return ES_UNSUCCESSFUL;
00097     }
00098
00099     // Initialize Terminos structure
00100     tty.c_cflag &= ~PARENB;  // Disable parity bit (IE clear parity bit)
00101     tty.c_cflag &= ~CSTOPB;  // 1 stop bit (IE clear stop field)
00102     tty.c_cflag &= ~CSIZE;   // Clear size bit
```

```
00103     tty.c_cflag |= CS8;      // 8 data bits
00104     tty.c_cflag &= ~CRTSCTS; // Disable hardware flow control
00105     tty.c_cflag |= CREAD | CLOCAL; // Turn on READ and ignore control lines
00106
00107     tty.c_lflag &= ~ICANON;  // Disable UNIX Canonical mode (\n != terminator)
00108     tty.c_lflag &= ~ECHO;    // Disable echo
00109     tty.c_lflag &= ~ECHOE;   // Disable erasure
00110     tty.c_lflag &= ~ECHONL;  // Disable new-line echo
00111     tty.c_lflag &= ~ISIG;    // Disable interpretation of INTR, QUIT and SUSP
00112
00113     tty.c_iflag &= ~(IXON | IXOFF | IXANY); // Disable software flow control
00114     tty.c_iflag &= ~(IGNBRK | BRKINT | PARMRK | ISTRIP | INLCR | IGNCR |
00115                      ICRNL); // Disable any special handling of received bytes
00116
00117     tty.c_oflag &= ~OPOST;
00118     tty.c_oflag &= ~ONLCR;
00119
00120     tty.c_cc[VTIME] = 100; // Set timeout of 10.0 seconds
00121     tty.c_cc[VMIN] = 0;
00122
00123     // Set baud rate
00124     cfsetispeed(&tty, B115200);
00125     cfsetospeed(&tty, B115200);
00126
00127     cfmakeraw(&tty);
00128
00129     // Save tty settings, also check for error
00130     if (tcsetattr(serial_port, TCSANOW, &tty) != 0) {
00131         spdlog::error("Error {} from tcsetattr: {}", errno, strerror(errno));
00132         return ES_UNSUCCESSFUL;
00133     }
00134     ioctl(serial_port, TIOCEXCL); // Disallow other processes from using this serial port
00135 #endif
00136     SPDLOG_DEBUG("Successfully initialized port {} with fd {}", port, serial_port);
00137     return ES_OK;
00138 }
00139
00145 estts::Status posix_serial::open_port() {
00146 #ifndef __ESTTS_OS_LINUX__
00147     // Open port stored in object with read/write
00148     serial_port = open(this->port, O_RDWR | O_NOCTTY | O_NDELAY);
00149 #else
00150     // Open port stored in object with read/write
00151     serial_port = open(this->port, O_RDWR);
00152 #endif
00153     // Check for errors
00154     if (serial_port < 0) {
00155         SPDLOG_ERROR("Error {} from open: {}", errno, strerror(errno));
00156         return ES_UNSUCCESSFUL;
00157     }
00158     SPDLOG_DEBUG("Successfully opened port {} with fd {}", port, serial_port);
00159     return ES_OK;
00160 }
00161
00169 ssize_t posix_serial::write_serial_uc(unsigned char *data, int size) const {
00170     // If serial port isn't open, error out
00171     if (serial_port < 0) {
00172         return -1;
00173     }
00174     ssize_t written = write(serial_port, data, size);
00175     if (written < 1) {
00176         return -1;
00177     }
00178     if (data[size] == '\r')
00179         SPDLOG_TRACE("Wrote '{}' (size={}) to {}", data, written, port);
00180     else {
00181         std::stringstream temp;
00182         for (int i = 0; i < size; i ++) {
00183             if (data[i] != '\r')
00184                 temp << data[i];
00185         }
00186         SPDLOG_TRACE("Wrote '{}' (size={}) to {}", temp.str(), written, port);
00187     }
00188     return written;
00189 }
00190
00199 unsigned char *posix_serial::read_serial_uc() {
00200     // Clear cache buf
00201     cache.clear();
00202     SPDLOG_TRACE("Reading serial - uc");
00203
00204     // If serial port isn't open, error out
00205     if (serial_port < 0) {
00206         SPDLOG_ERROR("Serial port not open! - serial_port = {}", serial_port);
00207         return nullptr;
00208     }
00209     auto buf = new unsigned char[MAX_SERIAL_READ];
```

```
00210     ssize_t n = read(serial_port, buf, MAX_SERIAL_READ);
00211     if (n < 1) {
00212         return nullptr;
00213     }
00214     if (buf[n] == '\r')
00215         SPDLOG_TRACE("Read '{}' (size={}) from {}", buf, n, port);
00216     else {
00217         std::stringstream temp;
00218         for (int i = 0; i < n; i ++) {
00219             if (buf[i] != '\r')
00220                 temp << buf[i];
00221         }
00222         SPDLOG_TRACE("Read '{}' (size={}) from {}", temp.str(), n, port);
00223     }
00224     // Add null terminator at the end of transmission for easier processing by parent class(s)
00225     buf[n] = '\0';
00226     cache << buf;
00227     return buf;
00228 }
00229
00235 estts::Status posix_serial::write_serial_s(const std::string &data) const {
00236     // If serial port isn't open, error out
00237     if (serial_port < 0) return estts::ES_UNINITIALIZED;
00238     // Cast string to const unsigned char *, then cast away const to pass
00239     // to method that writes unsigned char
00240     auto csc_data = const_cast<unsigned char *>(reinterpret_cast<const unsigned char
       *>(data.c_str()));
00241     if (this->write_serial_uc(csc_data, (int) data.length()) < 0) return estts::ES_UNSUCCESSFUL;
00242     return estts::ES_OK;
00243 }
00244
00249 std::string posix_serial::read_serial_s() {
00250     SPDLOG_TRACE("Reading serial - s");
00251     // If serial port isn't open, error out
00252     if (serial_port < 0) {
00253         SPDLOG_ERROR("Serial port not open! - serial_port = {}", serial_port);
00254         return "";
00255     }
00256     // Read serial data
00257     auto read = this->read_serial_uc();
00258     if (read == nullptr) {
00259         return "";
00260     }
00261     // Type cast unsigned char (auto) to a char *
00262     // Then call std::string constructor
00263     std::string string_read(reinterpret_cast<char const *>(read));
00264     delete read;
00265     return string_read;
00266 }
00267
00268 posix_serial::~posix_serial() {
00269     close(serial_port);
00270 }
00271
00272 void posix_serial::clear_serial_fifo() {
00273     SPDLOG_TRACE("Clearing serial FIFO buffer");
00274     if (check_serial_bytes_avail() > 0)
00275         read_serial_s();
00276 }
00277
00278 int posix_serial::check_serial_bytes_avail() const {
00279     int bytes;
00280     ioctl(serial_port, FIONREAD, &bytes);
00281     return bytes;
00282 }
```

7.55 src/ti/posix_serial.h File Reference

```
#include <sstream>
#include "constants.h"
```

Classes

- class posix_serial

7.56 posix_serial.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 3/17/22.
00003 //
00004
00005 #ifndef ESTTS_TI_POSIX_SERIAL_H
00006 #define ESTTS_TI_POSIX_SERIAL_H
00007
00008 #include <sstream>
00009 #include "constants.h"
00010
00011
00012 class posix_serial {
00013     // Private variables
00014     int serial_port;
00015
00016     // Private functions
00017     estts::Status open_port();
00018
00019     estts::Status initialize_serial_port();
00020
00021     const char *port;
00022     int baud;
00023 protected:
00024     // Check here first, maybe what you're waiting for is already received..
00025     // Note - cleared every time read is called
00026     std::stringstream cache;
00027
00028     posix_serial(const char *port, int baud);
00029
00030     ~posix_serial();
00031
00032     virtual ssize_t write_serial_uc(unsigned char *data, int size) const;
00033
00034     virtual unsigned char *read_serial_uc();
00035
00036     virtual estts::Status write_serial_s(const std::string &data) const;
00037
00038     virtual std::string read_serial_s();
00039
00040     virtual void clear_serial_fifo();
00041
00042     virtual int check_serial_bytes_avail() const;
00043 };
00044
00045
00046 #endif //ESTTS_TI_POSIX_SERIAL_H
```

7.57 src/ti/serial_handler.cpp File Reference

```
#include <iostream>
#include <sstream>
#include <dirent.h>
#include <cstring>
#include <vector>
#include <termios.h>
#include <boost/asio.hpp>
#include <sys/ioctl.h>
#include "serial_handler.h"
#include "helper.h"
```

7.58 serial_handler.cpp

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
```

```
00002 //
00003 // Created by Hayden Roszell on 3/17/22.
00004 //
00005
00006 #include <iostream>
00007 #include <sstream>
00008 #include <dirent.h>
00009 #include <cstring>
00010 #include <vector>
00011 #include <termios.h>
00012 #include <boost/asio.hpp>
00013 #include <sys/ioctl.h>
00014 #include "serial_handler.h"
00015 #include "helper.h"
00016
00017 using namespace boost::asio;
00018 using namespace estts;
00019
00025 serial_handler::serial_handler() : io(), serial(io) {
00026 #ifndef __ESTTS_OS_LINUX__
00027     this->port = estts::ti_serial::TI_SERIAL_ADDRESS;
00028 #else
00029     if (ES_OK != find_serial_port())
00030         throw std::runtime_error("Couldn't find serial device to attach to.");
00031 #endif
00032     this->baud = 115200;
00033
00034     sync_buf = new unsigned char[MAX_SERIAL_READ];
00035
00036     SPDLOG_DEBUG("Opening serial port {} with {} baud", port, baud);
00037
00038     if (ES_OK != initialize_serial_port()) {
00039         SPDLOG_ERROR("Failed to initialize serial port {}", port);
00040         throw std::runtime_error("Failed to initialize serial port.");
00041     }
00042 }
00043
00048 estts::Status serial_handler::initialize_serial_port() {
00049     boost::system::error_code error;
00050
00051     if (serial.is_open())
00052         serial.close();
00053
00054     serial.open(port, error);
00055     if (error) {
00056         SPDLOG_ERROR("Failed to open serial port {} - {}", port, error.message());
00057         return ES_UNSUCCESSFUL;
00058     }
00059
00060     struct termios tty{};
00061     if (tcgetattr(serial.lowest_layer().native_handle(), &tty) != 0) {
00062         SPDLOG_ERROR("Error %i from tcgetattr: %s\n", errno, strerror(errno));
00063         SPDLOG_INFO("Did you mean to use TI Dev Mode? See README.md");
00064         return ES_UNSUCCESSFUL;
00065     }
00066
00067     // Initialize Terminos structure
00068     tty.c_lflag &= ~ICANON;  // Disable UNIX Canonical mode (\n != terminator)
00069     tty.c_lflag &= ~ECHO;    // Disable echo
00070     tty.c_lflag &= ~ECHOE;   // Disable erasure
00071     tty.c_lflag &= ~ECHONL;  // Disable new-line echo
00072     tty.c_lflag &= ~ISIG;    // Disable interpretation of INTR, QUIT and SUSP
00073     // tty.c_iflag &= ~(IXON | IXOFF | IXANY); // Disable software flow control
00074     // tty.c_iflag &= ~(IGNBRK | BRKINT | PARMRK | ISTRIP | INLCR | IGNCR | ICRNL); // Disable any
    special handling of received bytes
00075
00076     // Save tty settings, also check for error
00077     if (tcsetattr(serial.lowest_layer().native_handle(), TCSANOW, &tty) != 0) {
00078         SPDLOG_ERROR("Error {} from tcsetattr: {}", errno, strerror(errno));
00079         return ES_UNSUCCESSFUL;
00080     }
00081
00082     try {
00083         serial.set_option(boost::asio::serial_port_base::baud_rate(baud));
00084         serial.set_option(boost::asio::serial_port_base::character_size(8));
00085
    serial.set_option(boost::asio::serial_port_base::flow_control(boost::asio::serial_port_base::flow_control::none));
00086
    serial.set_option(boost::asio::serial_port_base::parity(boost::asio::serial_port_base::parity::none));
00087
    serial.set_option(boost::asio::serial_port_base::stop_bits(boost::asio::serial_port_base::stop_bits::one));
00088     } catch (boost::system::system_error::exception &e) {
00089         SPDLOG_ERROR("Failed to configure serial port with baud {} - {}", baud, e.what());
00090         return ES_UNSUCCESSFUL;
00091     }
00092
00093     return estts::ES_OK;
```

```
00094 }
00095
00103 size_t serial_handler::write_serial_uc(unsigned char *data, int size) {
00104     if (!serial.is_open()) {
00105         SPDLOG_ERROR("Serial port {} not open", port);
00106         return -1;
00107     }
00108     size_t written = 0;
00109     boost::system::error_code error;
00110     // written = serial.write_some(buffer(data,size), error);
00111     written = write(serial, buffer(data,size), error);
00112     if (error) {
00113         SPDLOG_ERROR("Failed to write to serial port - {}", error.message());
00114         if (error == boost::asio::error::eof)
00115             initialize_serial_port();
00116     }
00117     print_write_trace_msg(data, written, port);
00118
00119     return written;
00120 }
00121
00130 unsigned char *serial_handler::read_serial_uc() {
00131     // Clear cache buf
00132     cache.clear();
00133
00134     if (!serial.is_open()) {
00135         SPDLOG_ERROR("Serial port {} not open", port);
00136         return nullptr;
00137     }
00138     size_t n = 0;
00139     boost::system::error_code error;
00140     n = serial.read_some(buffer(sync_buf, MAX_SERIAL_READ), error);
00141     if (error) {
00142         SPDLOG_ERROR("Failed to read from serial port - {}", error.message());
00143         if (error == boost::asio::error::eof)
00144             initialize_serial_port();
00145     }
00146
00147     print_read_trace_msg(sync_buf, n, port);
00148
00149     sync_buf[n] = '\0';
00150     cache << sync_buf;
00151     return sync_buf;
00152 }
00153
00159 estts::Status serial_handler::write_serial_s(const std::string &data) {
00160     if (!serial.is_open()) {
00161         SPDLOG_ERROR("Serial port {} not open", port);
00162         return estts::ES_UNINITIALIZED;
00163     }
00164     // Cast string to const unsigned char *, then cast away const to pass
00165     // to method that writes unsigned char
00166     auto csc_data = const_cast<unsigned char *>(reinterpret_cast<const unsigned char
       *>(data.c_str()));
00167     if (this->write_serial_uc(csc_data, (int) data.length()) < 0) return estts::ES_UNSUCCESSFUL;
00168     return estts::ES_OK;
00169 }
00170
00175 std::string serial_handler::read_serial_s() {
00176     if (!serial.is_open()) {
00177         SPDLOG_ERROR("Serial port {} not open", port);
00178         return "";
00179     }
00180     // Read serial data
00181     auto read = this->read_serial_uc();
00182     if (read == nullptr) {
00183         return "";
00184     }
00185     // Type cast unsigned char (auto) to a char *
00186     // Then call std::string constructor
00187     std::string string_read(reinterpret_cast<char const *>(read));
00188     return string_read;
00189 }
00190
00191 serial_handler::~serial_handler() {
00192     serial.close();
00193 }
00194
00195 void serial_handler::clear_serial_fifo() {
00196     SPDLOG_TRACE("Clearing serial FIFO buffer");
00197     while (check_serial_bytes_avail() > 0)
00198         read_serial_s();
00199 }
00200
00201 void serial_handler::clear_serial_fifo(const std::function<estts::Status(std::string)> &cb) {
00202     SPDLOG_TRACE("Clearing serial FIFO buffer");
00203     while (check_serial_bytes_avail() > 0)
```

```
00204         cb(read_serial_s());
00205 }
00206
00207 int serial_handler::check_serial_bytes_avail() {
00208     int value = 0;
00209     if (0 != ioctl(serial.lowest_layer().native_handle(), FIONREAD, &value)) {
00210         SPDLOG_ERROR("Failed to get bytes available - {}",
00211                      boost::system::error_code(errno,
      boost::asio::error::get_system_category()).message());
00212     }
00213     return value;
00214 }
00215
00216 void serial_handler::read_serial_async(const std::function<estts::Status(char *, size_t)>& cb) {
00217     serial.async_read_some(buffer(async_buf, MAX_SERIAL_READ), get_generic_async_read_lambda(cb));
00218 }
00219
00220 std::function<void(boost::system::error_code, size_t)>
      serial_handler::get_generic_async_read_lambda(const std::function<Status(char *, size_t)>&
      estts_callback) {
00221     return [estts_callback, this] (const boost::system::error_code& error, std::size_t
      bytes_transferred) {
00222         if (error) {
00223             spdlog::error("Async read failed - {}", error.message().c_str());
00224         }
00225         std::stringstream temp;
00226         for (char i : async_buf) {
00227             if (i != '\r')
00228                 temp << i;
00229         }
00230         spdlog::info("Async callback lambda - Got back -> {}", temp.str());
00231
00232         estts_callback(this->async_buf, bytes_transferred);
00233
00234         return estts::ES_OK;
00235     };
00236 }
00237
00238 unsigned char *serial_handler::read_serial_uc(int bytes) {
00239     // Clear cache buf
00240     cache.clear();
00241
00242     if (!serial.is_open()) {
00243         SPDLOG_ERROR("Serial port {} not open", port);
00244         return nullptr;
00245     }
00246     SPDLOG_TRACE("Reading {} bytes from {}", bytes, port);
00247
00248     boost::system::error_code error;
00249     auto n = read(serial, buffer(sync_buf, bytes), error);
00250     if (error) {
00251         SPDLOG_ERROR("Failed to read from serial port - {}", error.message());
00252         if (error == boost::asio::error::eof)
00253             initialize_serial_port();
00254     }
00255     // Add null terminator at the end of transmission for easier processing by parent class(s)
00256     sync_buf[n] = '\0';
00257
00258     print_read_trace_msg(sync_buf, n, port);
00259
00260     cache << sync_buf;
00261     return sync_buf;
00262 }
00263
00264 std::string serial_handler::read_serial_s(int bytes) {
00265     if (!serial.is_open()) {
00266         SPDLOG_ERROR("Serial port {} not open", port);
00267         return "";
00268     }
00269     // Read serial data
00270     auto read = this->read_serial_uc(bytes);
00271     if (read == nullptr) {
00272         return "";
00273     }
00274     // Type cast unsigned char (auto) to a char *
00275     // Then call std::string constructor
00276     std::string string_read(reinterpret_cast<char const *>(read));
00277
00278     return string_read;
00279 }
00280
00281 Status serial_handler::find_serial_port() {
00282     std::string serial_dir = "/dev/serial/by-id";
00283     std::stringstream temp_sym;
00284     DIR * d = opendir(serial_dir.c_str()); // open the path
00285     if (d == nullptr) return estts::ES_UNSUCCESSFUL; // if was not able to open path, return
00286     struct dirent * dir;
```

```
00287     while ((dir = readdir(d)) != nullptr) {
00288         if (dir->d_type == DT_LNK && strcmp(dir->d_name,".")!=0 && strcmp(dir->d_name,"..")!=0 ) {
00289             // Basically if we get here whatever is inside by-id is a symlink pointing to the /dev
00290             // location we want to connect to. We need to get the absolute path.
00291             temp_sym « serial_dir « "/" « dir->d_name;
00292             SPDLOG_INFO("Found serial device mounted at {}", temp_sym.str());
00293             break;
00294         }
00295     }
00296     if (temp_sym.str().empty()) {
00297         SPDLOG_ERROR("Didn't find serial port to attach to. Ensure that device is plugged in and
      mounted.");
00298         return ES_NOTFOUND;
00299     }
00300     this->port = temp_sym.str();
00301     closedir(d);
00302     return ES_OK;
00303 }
```

7.59 src/ti/serial_handler.h File Reference

```
#include <sstream>
#include <boost/asio.hpp>
#include "constants.h"
#include "posix_serial.h"
```

Classes

- class serial_handler

7.60 serial_handler.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/10/21.
00003 //
00004
00005 #ifndef ESTTS_SERIAL_HANDLER_H
00006 #define ESTTS_SERIAL_HANDLER_H
00007
00008 #include <sstream>
00009 #include <boost/asio.hpp>
00010 #include "constants.h"
00011
00012 #include "posix_serial.h"
00013
00014 class serial_handler {
00015 private:
00016     boost::asio::io_service io;
00017     boost::asio::serial_port serial;
00018     std::string port;
00019     int baud, restarts;
00020     unsigned char * sync_buf;
00021
00022     estts::Status find_serial_port();
00023 protected:
00024     // Check here first, maybe what you're waiting for is already received..
00025     // Note - cleared every time read is called
00026     std::stringstream cache;
00027
00028     char async_buf[MAX_SERIAL_READ];
00029
00030     serial_handler();
00031
00032     estts::Status initialize_serial_port();
00033
00034     ~serial_handler();
00035
```

```
00036     size_t write_serial_uc(unsigned char *data, int size);
00037
00038     unsigned char *read_serial_uc();
00039
00040     estts::Status write_serial_s(const std::string &data);
00041
00042     std::string read_serial_s();
00043
00044     unsigned char *read_serial_uc(int bytes);
00045
00046     std::string read_serial_s(int bytes);
00047
00048     std::function<void(boost::system::error_code, size_t)> get_generic_async_read_lambda(const
       std::function<estts::Status(char *, size_t)>& estts_callback);
00049
00050     void clear_serial_fifo();
00051
00052     void clear_serial_fifo(const std::function<estts::Status(std::string)>& cb);
00053
00054     int check_serial_bytes_avail();
00055
00056 public:
00057     void read_serial_async(const std::function<estts::Status(char *, size_t)>& cb);
00058 };
00059
00060 #endif //ESTTS_SERIAL_HANDLER_H
```

7.61 src/ti/socket_handler.cpp File Reference

```
#include <sstream>
#include <chrono>
#include <thread>
#include <fcntl.h>
#include <sys/ioctl.h>
#include "socket_handler.h"
#include <arpa/inet.h>
#include <unistd.h>
#include "helper.h"
```

7.62 socket_handler.cpp

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002 //
00003 // Created by Hayden Roszell on 1/3/22.
00004 //
00005
00006 #include <sstream>
00007 #include <chrono>
00008 #include <thread>
00009 #include <fcntl.h>
00010 #include <sys/ioctl.h>
00011 #include "socket_handler.h"
00012 #include <arpa/inet.h>
00013 #include <unistd.h>
00014 #include "helper.h"
00015
00016
00024 socket_handler::socket_handler(const char *address, int port) {
00025     sock = -1;
00026     serv_addr = {0};
00027     this->port = port;
00028     this->address = address;
00029     std::stringstream temp;
00030     temp « address « ":" « port;
00031     this->endpoint = temp.str();
00032
00033     sync_buf = new unsigned char[estts::ti_socket::TI_SOCKET_BUF_SZ];
00034 }
00035
```

```
00036 estts::Status socket_handler::open_socket() {
00037     // Creating socket file descriptor
00038     if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
00039         spdlog::error("Error {} from socket(): {}", errno, strerror(errno));
00040         return estts::ES_UNSUCCESSFUL;
00041     }
00042     return estts::ES_OK;
00043 }
00044
00050 estts::Status socket_handler::configure_socket() {
00051     serv_addr.sin_family = AF_INET;
00052     serv_addr.sin_port = htons(port);
00053
00054     // Convert IPv4 and IPv6 addresses from text to binary form
00055     if (inet_pton(AF_INET, address, &serv_addr.sin_addr) <= 0) {
00056         spdlog::error("Invalid address / Address not supported");
00057         return estts::ES_UNSUCCESSFUL;
00058     }
00059
00060     SPDLOG_DEBUG("Attempting to connect to socket at address {}:{}", address, port);
00061
00062     if (connect(sock, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {
00063         spdlog::error("Error {} from connect(): {}", errno, strerror(errno));
00064         return estts::ES_UNSUCCESSFUL;
00065     }
00066     int flags = fcntl(sock, F_GETFL, 0);
00067     if (0 != fcntl(sock, F_SETFL, flags & ~O_NONBLOCK))
00068         return estts::ES_UNSUCCESSFUL;
00069     SPDLOG_TRACE("Connection succeeded.");
00070
00071     return estts::ES_OK;
00072 }
00073
00081 ssize_t socket_handler::write_socket_uc(unsigned char *data, int size) const {
00082     // If serial port isn't open, error out
00083     if (sock < 0) {
00084         return -1;
00085     }
00086     ssize_t written = send(sock, data, size, 0);
00087     if (written < 1) {
00088         return -1;
00089     }
00090
00091     print_write_trace_msg(data, written, endpoint);
00092
00093     return written;
00094 }
00095
00104 unsigned char *socket_handler::read_socket_uc() const {
00105     // If socket isn't open, error out
00106     if (sock < 0) {
00107         return nullptr;
00108     }
00109     // Allocate heap space for receive buffer
00110     // auto buf = new unsigned char[estts::ti_socket::TI_SOCKET_BUF_SZ];
00111
00112     // Use read system call to read data in sock to buf
00113     auto n = read(sock, sync_buf, estts::ti_socket::TI_SOCKET_BUF_SZ);
00114     if (n < 1) {
00115         // Can't receive a negative number of bytes ;)
00116         return nullptr;
00117     }
00118
00119     print_read_trace_msg(sync_buf, n, endpoint);
00120
00121     // Add null terminator at the end of transmission for easier processing by parent class(s)
00122     sync_buf[n] = '\0';
00123     return sync_buf;
00124 }
00125
00131 estts::Status socket_handler::write_socket_s(const std::string &data) const {
00132     // If serial port isn't open, error out
00133     if (sock < 0) return estts::ES_UNINITIALIZED;
00134     // Cast string to const unsigned char *, then cast away const to pass
00135     // to method that writes unsigned char
00136     auto csc_data = const_cast<unsigned char *>(reinterpret_cast<const unsigned char
       *>(data.c_str()));
00137     if (this->write_socket_uc(csc_data, (int) data.length()) < 0) return estts::ES_UNSUCCESSFUL;
00138     return estts::ES_OK;
00139 }
00140
00145 std::string socket_handler::read_socket_s() const {
00146     // If serial port isn't open, error out
00147     if (sock < 0) {
00148         return "";
00149     }
00150     // Read serial data
```

```
00151     auto read = this->read_socket_uc();
00152     if (read == nullptr) {
00153         return "";
00154     }
00155     // Type cast unsigned char (auto) to a char *
00156     // Then call std::string constructor
00157     std::string string_read(reinterpret_cast<char const *>(read));
00158
00159     return string_read;
00160 }
00161
00162 int socket_handler::check_sock_bytes_avail() const {
00163     int count;
00164     ioctl(sock, FIONREAD, &count);
00165     return count;
00166 }
00167
00168 estts::Status socket_handler::init_socket_handle() {
00169     SPDLOG_DEBUG("Opening socket at {}:{}", address, port);
00170     if (estts::ES_OK != open_socket()) {
00171         spdlog::error("Failed to open socket.");
00172         return estts::ES_UNINITIALIZED;
00173     }
00174     if (estts::ES_OK != configure_socket()) {
00175         spdlog::error("Failed to configure socket.");
00176         SPDLOG_WARN("Is the ESTTS server running? See documentation for more.");
00177         return estts::ES_UNINITIALIZED;
00178     }
00179     SPDLOG_DEBUG("Socket configuration complete.");
00180     return estts::ES_OK;
00181 }
00182
00183 socket_handler::~socket_handler() {
00184     if (sync_buf) delete sync_buf;
00185 }
```

7.63 src/ti/socket_handler.h File Reference

```
#include <string>
#include <sys/socket.h>
#include <netinet/in.h>
#include "constants.h"
```

Classes

- class socket_handler

7.64 socket_handler.h

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002 //
00003 // Created by Hayden Roszell on 1/3/22.
00004 //
00005
00006 #ifndef ESTTS_SOCKET_HANDLER_H
00007 #define ESTTS_SOCKET_HANDLER_H
00008
00009 #include <string>
00010 #include <sys/socket.h>
00011 #include <netinet/in.h>
00012 #include "constants.h"
00013
00014 class socket_handler {
00015 private:
00016     unsigned char * sync_buf;
00017
00018     const char *address;
```

```
00019
00020     std::string endpoint;
00021
00022     struct sockaddr_in serv_addr;
00023
00024     estts::Status open_socket();
00025
00026     estts::Status configure_socket();
00027
00028 protected:
00029
00030     int check_sock_bytes_avail() const;
00031
00032 public:
00033
00034     int sock, port;
00035
00036     socket_handler(const char *address, int port);
00037
00038     ~socket_handler();
00039
00040     std::string read_socket_s() const;
00041
00042     unsigned char * read_socket_uc() const;
00043
00044     estts::Status write_socket_s(const std::string &data) const;
00045
00046     ssize_t write_socket_uc(unsigned char *data, int size) const;
00047
00048     estts::Status init_socket_handle();
00049 };
00050
00051
00052 #endif //ESTTS_SOCKET_HANDLER_H
```

7.65 src/ti/transmission_interface.cpp File Reference

```
#include <chrono>
#include <thread>
#include "transmission_interface.h"
```

7.66 transmission_interface.cpp

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002 //
00003 // Created by Hayden Roszell on 12/21/21.
00004 //
00005
00006 #include <chrono>
00007 #include <thread>
00008
00009 #include "transmission_interface.h"
00010
00011 using namespace std::this_thread; // sleep_for, sleep_until
00012 using namespace std::chrono; // nanoseconds, system_clock, seconds
00013 using namespace estts;
00014 using namespace estts::endurosat;
00015
00016 transmission_interface::transmission_interface() : socket_handler(ti_socket::TI_SOCKET_ADDRESS,
00017
      ti_socket::TI_SOCKET_PORT) {
00018     primary_telem_cb = nullptr;
00019     obc_session_active = false;
00020     pipe_mode = PIPE_OFF;
00021     satellite_in_range = false;
00022     dispatch_threadpool_active = false;
00023
00024     inrange_checker = std::thread(&transmission_interface::detect_satellite_in_range, this);
00025     SPDLOG_TRACE("Created thread to detect if the satellite is in range with ID {}",
     std::hash<std::thread::id>{}(pipe_keeper.get_id()));
00026
00027     mtx.unlock();
```

```
00028 }
00029
00030 Status transmission_interface::transmit(const std::string &value) {
00031     using namespace std::this_thread; // sleep_for, sleep_until
00032     using namespace std::chrono; // nanoseconds, system_clock, seconds
00033     if (value.empty())
00034         return ES_MEMORY_ERROR;
00035     if (!obc_session_active)
00036         SPDLOG_WARN("Communication session not active, message may not get to satellite.");
00037     mtx.lock();
00038     if (primary_telem_cb)
00039         clear_serial_fifo(primary_telem_cb);
00040     else
00041         clear_serial_fifo();
00042 #ifndef __TI_DEV_MODE__
00043     clear_serial_fifo();
00044     if (this->write_serial_s(value) != ES_OK) {
00045         SPDLOG_ERROR("Failed to transmit.");
00046         mtx.unlock();
00047         return ES_UNSUCCESSFUL;
00048     }
00049 #endif
00050     mtx.unlock();
00051     return ES_OK;
00052 }
00053
00054 std::string transmission_interface::receive() {
00055     mtx.lock();
00056 #ifndef __TI_DEV_MODE__
00057     auto buf = internal_receive();
00058     mtx.unlock();
00059     return buf;
00060 #else
00061     std::string received;
00062     do {
00063
00064     }
00065     while ((received = this->read_socket_s()).empty());
00066     mtx.unlock();
00067     return received;
00068 #endif
00069 }
00070
00071 std::string transmission_interface::internal_receive() {
00072     int bytes_avail;
00073     for (int seconds_elapsed = 0; seconds_elapsed < ESTTS_AWAIT_RESPONSE_PERIOD_SEC * 50;
      seconds_elapsed++) {
00074         bytes_avail = check_serial_bytes_avail();
00075         if (bytes_avail > 0) {
00076             break;
00077         }
00078         sleep_until(system_clock::now() + milliseconds (100));
00079     }
00080     if (bytes_avail <= 0) {
00081         mtx.unlock();
00082         return "";
00083     }
00084     auto buf = this->read_serial_s(bytes_avail);
00085     return buf;
00086 }
00087
00088 transmission_interface::~transmission_interface() {
00089     mtx.lock();
00090
00091 #ifdef __TI_DEV_MODE__
00092     this->write_socket_s("close");
00093 #endif
00094
00095     mtx.unlock();
00096 }
00097
00098 bool transmission_interface::check_data_available() {
00099 #ifdef __TI_DEV_MODE__
00100     if (check_sock_bytes_avail() > 0)
00101         return true;
00102     else
00103         return false;
00104 #endif
00105     if (check_serial_bytes_avail() > 0)
00106         return true;
00107     else
00108         return false;
00109 }
00110
00111 Status transmission_interface::transmit(const unsigned char *value, int length) {
00112     using namespace std::this_thread; // sleep_for, sleep_until
00113     using namespace std::chrono; // nanoseconds, system_clock, seconds
```

```
00114     if (length <= 0)
00115         return ES_MEMORY_ERROR;
00116     int retries = 0;
00117     if (!obc_session_active)
00118         SPDLOG_WARN("Communication session not active, message may not get to satellite.");
00119     mtx.lock();
00120 #ifndef __TI_DEV_MODE__
00121     retries = 0;
00122     clear_serial_fifo();
00123     while (this->write_serial_uc((unsigned char *)value, length) < length) {
00124         spdlog::error("Failed to transmit. Waiting {} seconds", endurosat::WAIT_TIME_SEC);
00125         sleep_until(system_clock::now() + seconds(endurosat::WAIT_TIME_SEC));
00126         retries++;
00127         if (retries > endurosat::MAX_RETRIES) {
00128             mtx.unlock();
00129             return ES_UNSUCCESSFUL;
00130         }
00131         SPDLOG_INFO("Retrying transmit (retry {}/{})", retries, endurosat::MAX_RETRIES);
00132     }
00133 #else
00134     SPDLOG_DEBUG("Transmitting {}", value);
00135     while (this->write_socket_uc((unsigned char *)value, length) < length) {
00136         spdlog::error("Failed to transmit. Waiting {} seconds", ti_socket::WAIT_TIME_SEC);
00137         sleep_until(system_clock::now() + seconds(ti_socket::WAIT_TIME_SEC));
00138         retries++;
00139         if (retries > ti_socket::MAX_RETRIES) {
00140             mtx.unlock();
00141             return ES_UNSUCCESSFUL;
00142         }
00143         SPDLOG_INFO("Retrying transmit (retry {}/{})", retries, ti_socket::MAX_RETRIES);
00144     }
00145 #endif
00146     mtx.unlock();
00147     return ES_OK;
00148 }
00149
00150 unsigned char *transmission_interface::receive_uc() {
00151     mtx.lock();
00152 #ifndef __TI_DEV_MODE__
00153     auto buf = this->read_serial_uc();
00154     mtx.unlock();
00155     return buf;
00156 #else
00157     unsigned char * received;
00158     do {
00159     }
00160     while (!(received = this->read_socket_uc()));
00161     mtx.unlock();
00162     return received;
00163 #endif
00164 }
00165
00166 std::string transmission_interface::nonblock_receive() {
00167     if (!mtx.try_lock())
00168         return "";
00169     int bytes;
00170     sleep_until(system_clock::now() + milliseconds (20));
00171     if ((bytes = check_serial_bytes_avail()) > 0) {
00172         mtx.unlock();
00173         return this->read_serial_s(bytes);
00174     }
00175     mtx.unlock();
00176     return "";
00177 }
00178
00179 Status transmission_interface::gs_transmit(const std::string &value) {
00180     if (value.empty())
00181         return ES_MEMORY_ERROR;
00182     mtx.lock();
00183 #ifndef __TI_DEV_MODE__
00184     clear_serial_fifo();
00185     if (this->write_serial_s(value) != ES_OK) {
00186         SPDLOG_ERROR("Failed to transmit.");
00187         mtx.unlock();
00188         return ES_UNSUCCESSFUL;
00189     }
00190 #endif
00191     mtx.unlock();
00192     return ES_OK;
00193 }
00194
00199 Status transmission_interface::enable_pipe() {
00200     std::string pipe_en = "ES+W22003323\r";
00201     std::stringstream buf;
00202     pipe_mode = PIPE_OFF;
00203
00204     for (int retries = 0; retries < endurosat::MAX_RETRIES; retries++) {
```

```
00205         if (retries > endurosat::MAX_RETRIES) {
00206             return ES_UNSUCCESSFUL;
00207         }
00208         SPDLOG_TRACE("Attempting to enable PIPE on ground station");
00209         write_serial_s(pipe_en);
00210         sleep_until(system_clock::now() + milliseconds (50));
00211 
00212         // Try to get data 3 times
00213         for (int i = 0; i < 3; i++) {
00214             if (check_data_available()) {
00215                 buf « read_serial_s();
00216                 if (buf.str().find("OK+3323\r") != std::string::npos && buf.str().find("+PIPE\r") !=
      std::string::npos) {
00217                     pipe_mode = PIPE_ON;
00218                     break;
00219                 }
00220             }
00221         }
00222         if (pipe_mode == PIPE_ON)
00223             break;
00224         SPDLOG_WARN("PIPE enable unsuccessful. Retrying ({}/{} retries)", retries + 1,
      endurosat::MAX_RETRIES);
00225     }
00226 
00227     if (pipe_mode != PIPE_ON) {
00228         SPDLOG_ERROR("Failed to enable PIPE on ground station transceiver.");
00229         return ES_UNSUCCESSFUL;
00230     }
00231 
00232     pipe_keeper = std::thread(&transmission_interface::maintain_pipe, this);
00233     SPDLOG_TRACE("Created maintain PIPE thread with ID {}",
      std::hash<std::thread::id>{}(pipe_keeper.get_id()));
00234     SPDLOG_DEBUG("PIPE is active.");
00235 
00236     return ES_OK;
00237 }
00238 
00239 Status transmission_interface::disable_pipe() {
00240     SPDLOG_DEBUG("Exiting PIPE mode");
00241     int retries = endurosat::PIPE_DURATION_SEC * 2;
00242 
00243     if (pipe_mode != PIPE_ON)
00244         return ES_OK;
00245 
00246     pipe_mode = PIPE_WAITING;
00247     pipe_keeper.join();
00248     std::string resp;
00249     std::stringstream buf;
00250     while (true) {
00251         auto avail = this->check_serial_bytes_avail();
00252         if (avail > 0) {
00253             buf « read_serial_s();
00254         }
00255         if (buf.str().find("+ESTTC") != std::string::npos) {
00256             break;
00257         }
00258         sleep_until(system_clock::now() + seconds(1));
00259         retries--;
00260         if (retries <= 0) {
00261             SPDLOG_ERROR("Oof PIPE didn't exit properly..");
00262             pipe_mode = PIPE_OFF;
00263             return ES_UNSUCCESSFUL;
00264         }
00265     }
00266     SPDLOG_TRACE("PIPE successfully exited.");
00267     pipe_mode = PIPE_OFF;
00268     return ES_OK;
00269 }
00270 
00271 void transmission_interface::maintain_pipe() {
00272     int counter = 0;
00273     while (pipe_mode == PIPE_ON) {
00274         counter++;
00275         if ((counter / 10) > (endurosat::PIPE_DURATION_SEC - 4)) {
00276             this->write_serial_uc((unsigned char *) " ", 1);
00277             counter = 0;
00278         }
00279         sleep_until(system_clock::now() + milliseconds (100));
00280     }
00281 }
00282 
00283 Status transmission_interface::request_obc_session() {
00284     mtx.lock();
00285 
00286     // Wait up to ESTTS_SATELLITE_CONNECTION_TIMEOUT_MIN minutes for the satellite to come in range.
00287     // Check every 1 minute
00288     auto wait = 0;
```

```
00289     while (!satellite_in_range) {
00290         if (wait > ESTTS_SATELLITE_CONNECTION_TIMEOUT_MIN) {
00291             SPDLOG_INFO("Satellite not detected within {} minutes.");
00292             mtx.unlock();
00293             return estts::ES_UNSUCCESSFUL;
00294         }
00295         sleep_until(system_clock::now() + minutes(1));
00296         wait++;
00297     }
00298 
00299     SPDLOG_INFO("Requesting new session");
00300     std::string pipe_en = "ES+W22003323\r";
00301     int retries = 0;
00302     std::stringstream buf;
00303 
00304     if (primary_telem_cb)
00305         clear_serial_fifo(primary_telem_cb);
00306     else
00307         clear_serial_fifo();
00308 
00309     // Enable PIPE has built-in retries. Don't cascade retries, if this function failed
00310     // something is pretty messed up.
00311     if (ES_OK != enable_pipe()) {
00312         mtx.unlock();
00313         return ES_UNSUCCESSFUL;
00314     }
00315 
00316     // Sanity check - make sure PIPE is enabled
00317     if (PIPE_ON != pipe_mode) {
00318         SPDLOG_ERROR("enable_pipe() succeeded, but trace variable is not set to PIPE_ON");
00319         mtx.unlock();
00320         return ES_SERVER_ERROR;
00321     }
00322 
00323     // Clear FIFO buffer
00324     if (primary_telem_cb)
00325         clear_serial_fifo(primary_telem_cb);
00326     else
00327         clear_serial_fifo();
00328 
00329     // Now, try to enable PIPE on the satellite.
00330     while (true) {
00331         if (retries > endurosat::MAX_RETRIES) {
00332             SPDLOG_ERROR("Failed to enable PIPE on satellite transceiver. ({}/{} retries)", retries,
      endurosat::MAX_RETRIES);
00333             mtx.unlock();
00334             return ES_UNSUCCESSFUL;
00335         }
00336         write_serial_s(pipe_en);
00337         sleep_until(system_clock::now() + milliseconds (100));
00338         buf « internal_receive();
00339         if (buf.str().find("OK+3323\r") != std::string::npos) {
00340             SPDLOG_TRACE("PIPE is probably enabled on the satellite");
00341             obc_session_active = true;
00342             break;
00343         }
00344         retries++;
00345         SPDLOG_ERROR("Failed to enable PIPE on satellite. Waiting {} seconds (retry {}/{})",
      endurosat::WAIT_TIME_SEC, retries, endurosat::MAX_RETRIES);
00346         sleep_until(system_clock::now() + seconds(endurosat::WAIT_TIME_SEC));
00347         // Once again don't clear buf, maybe confirmation got lost in the weeds.
00348     }
00349 
00350     sleep_until(system_clock::now() + seconds(2));
00351 
00352     // At this point, there is already a thread maintaining the PIPE state.
00353     // Exit at this point.
00354 
00355     SPDLOG_INFO("Session active");
00356 
00357     if (primary_telem_cb)
00358         clear_serial_fifo(primary_telem_cb);
00359     else
00360         clear_serial_fifo();
00361 
00362     mtx.unlock();
00363     return ES_OK;
00364 }
00365 
00366 Status transmission_interface::end_obc_session(const std::string &end_frame) {
00367     mtx.lock();
00368     SPDLOG_INFO("Ending session");
00369 
00370     int retries = endurosat::PIPE_DURATION_SEC * 2;
00371     disable_pipe();
00372     sleep_until(system_clock::now() + seconds(1));
00373 
```

```
00374     SPDLOG_INFO("Successfully ended session");
00375     obc_session_active = false;
00376     mtx.unlock();
00377     return ES_OK;
00378 }
00379
00380 void transmission_interface::detect_satellite_in_range() {
00381     std::string get_scw = "ES+R2200\r";
00382     std::string enable_bcn = "ES+W22003340\r";
00383
00384     for (;;) {
00385         if (!obc_session_active && pipe_mode == PIPE_OFF && !gstxvr_session_active) {
00386             SPDLOG_TRACE("detect_satellite_in_range - locking mutex");
00387             mtx.lock();
00388             if (ES_OK == enable_pipe()) {
00389                 // Try to read some data from the satellite transceiver 3 times
00390                 std::stringstream buf;
00391                 for (int i = 0; i < 3; i++) {
00392                     write_serial_s(get_scw);
00393                     sleep_until(system_clock::now() + milliseconds (500));
00394                     if (check_data_available())
00395                         buf << read_serial_s();
00396                     if (buf.str().find("OK+") != std::string::npos) {
00397                         satellite_in_range = true;
00398                         start_dispatch_threads();
00399                         break;
00400                     } else {
00401                         satellite_in_range = false;
00402                         end_dispatch_threads();
00403                     }
00404                     sleep_until(system_clock::now() + seconds (ESTTS_RETRY_WAIT_SEC));
00405                 }
00406                 if (satellite_in_range)
00407                     SPDLOG_INFO("Satellite in range.");
00408                 else
00409                     SPDLOG_INFO("Satellite not in range");
00410             }
00411             disable_pipe();
00412             SPDLOG_TRACE("detect_satellite_in_range - unlocking mutex");
00413             mtx.unlock();
00414             sleep_until(system_clock::now() + seconds (ESTTS_CHECK_SATELLITE_INRANGE_INTERVAL_SEC));
00415         } else
00416             sleep_until(system_clock::now() + seconds (ESTTS_RETRY_WAIT_SEC));
00417     }
00418 }
00419
00420 void transmission_interface::register_dispatch_function(const std::function<void()>& fct) {
00421     dispatch_functions.push_back(fct);
00422 }
00423
00424 void transmission_interface::start_dispatch_threads() {
00425
00426     if (!dispatch_threadpool_active) {
00427         for (auto& i : dispatch_functions) {
00428             std::thread t(i);
00429             SPDLOG_TRACE("Created dispatch worker thread with ID {}",
      std::hash<std::thread::id>{}(t.get_id()));
00430             dispatch_threadpool.push_back(std::move(t));
00431         }
00432         dispatch_threadpool_active = true;
00433     }
00434 }
00435
00436 void transmission_interface::end_dispatch_threads() {
00437     if (dispatch_threadpool_active) {
00438         for (auto &i : dispatch_threadpool) {
00439             if (i.joinable())
00440                 i.join();
00441         }
00442         dispatch_threadpool_active = false;
00443     }
00444 }
00445
00446 estts::Status transmission_interface::request_gstxvr_session() {
00447     mtx.lock();
00448     int retries = 0;
00449     while (obc_session_active || pipe_mode != PIPE_OFF) {
00450         sleep_until(system_clock::now() + milliseconds (500));
00451         if (retries > ESTTS_REQUEST_SESSION_TIMEOUT_SECONDS * 2) {
00452             SPDLOG_WARN("Failed to request ground station transceiver session - timed out");
00453             return ES_UNSUCCESSFUL;
00454         }
00455     }
00456     gstxvr_session_active = true;
00457     mtx.unlock();
00458     return ES_OK;
00459 }
```

```
00460
00461 estts::Status transmission_interface::end_gstxvr_session() {
00462     mtx.lock();
00463     gstxvr_session_active = false;
00464     mtx.unlock();
00465     return ES_OK;
00466 }
```

7.67 src/ti/transmission_interface.h File Reference

```
#include <mutex>
#include <thread>
#include "esttc.h"
#include "socket_handler.h"
```

Classes

- class transmission_interface

7.68 transmission_interface.h

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002 //
00003 // Created by Hayden Roszell on 12/21/21.
00004 //
00005
00006 #ifndef ESTTS_TRANSMISSION_INTERFACE_H
00007 #define ESTTS_TRANSMISSION_INTERFACE_H
00008
00009 #include <mutex>
00010 #include <thread>
00011 #include "esttc.h"
00012 #include "socket_handler.h"
00013
00014 class transmission_interface : virtual public esttc, virtual public socket_handler {
00015 private:
00016     std::vector<std::function<void()» dispatch_functions;
00017
00018     std::vector<std::thread> dispatch_threadpool;
00019
00020     std::function<estts::Status(std::string)> primary_telem_cb;
00021
00022     std::mutex mtx;
00023
00024     std::thread pipe_keeper;
00025
00026     std::thread inrange_checker;
00027
00028     estts::endurosat::PIPE_State pipe_mode;
00029
00030     bool dispatch_threadpool_active;
00031
00037     void maintain_pipe();
00038
00044     [[noreturn]] void detect_satellite_in_range();
00045
00046     void start_dispatch_threads();
00047
00048     void end_dispatch_threads();
00049
00055     std::string internal_receive();
00056
00057 public:
00058     bool obc_session_active;
00059
00060     bool satellite_in_range;
00061
```

```
00062     bool gstxvr_session_active;
00063
00064     void register_dispatch_function(const std::function<void()>& fct);
00065
00070     explicit transmission_interface();
00071
00075     ~transmission_interface();
00076
00083     void set_telem_callback(const std::function<estts::Status(std::string)>& cb) { primary_telem_cb =
      cb;}
00084
00092     estts::Status transmit(const std::string &value);
00093
00102     estts::Status transmit(const unsigned char *value, int length);
00103
00109     std::string receive();
00110
00115     std::string nonblock_receive();
00116
00121     unsigned char * receive_uc();
00122
00128     estts::Status request_obc_session();
00129
00130     estts::Status request_gstxvr_session();
00131
00132     estts::Status end_gstxvr_session();
00133
00138     bool check_session_active() const { return obc_session_active; };
00139
00145     estts::Status end_obc_session(const std::string& end_frame);
00146
00151     bool check_data_available();
00152
00160     estts::Status gs_transmit(const std::string &value);
00161
00162     /*
00163      * CRITICAL NOTE - The function calling enable_pipe OR disable_pipe MUST take the mutex.
00164      * These functions execute REGARDLESS of the mutex state.
00165      */
00166
00167     estts::Status enable_pipe();
00168
00169     estts::Status disable_pipe();
00170 };
00171
00172
00173 #endif //ESTTS_TRANSMISSION_INTERFACE_H
```

7.69 src/tnc_emulator/ax25_ui_frame_constructor.cpp File Reference

```
#include <iostream>
#include <cstring>
#include <utility>
#include <spdlog/spdlog.h>
#include "ax25_ui_frame_constructor.h"
#include "constants.h"
#include "helper.h"
```

7.70 ax25_ui_frame_constructor.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Cody on 12/17/2021.
00003 //
00004
00005 #include <iostream>
00006 #include <cstring>
00007 #include <utility>
00008 #include <spdlog/spdlog.h>
00009 #include "ax25_ui_frame_constructor.h"
00010 #include "constants.h"
```

```
00011 #include "helper.h"
00012
00013 using std::cerr;
00014 using std::cout;
00015 using std::endl;
00016 using std::string;
00017 using std::stringstream;
00018 using std::strlen;
00019 using std::hex;
00020
00021 // ~~~~~~~~~~~ FRAME HEADER GETTERS  ~~~~~~~~~~~
00022
00027 string ax25_ui_frame_constructor::getFlag() {
00028     return reinterpret_cast<char const *>(estts::ax25::AX25_FLAG);
00029 }
00030
00035 string ax25_ui_frame_constructor::getDestAddr() {
00036     // Convert const char to HEX
00037     return ascii_to_hex(estts::ax25::AX25_DESTINATION_ADDRESS);
00038 }
00039
00044 string ax25_ui_frame_constructor::getSSID0() {
00045     return reinterpret_cast<char const *>(estts::ax25::AX25_SSID0);
00046 }
00047
00052 string ax25_ui_frame_constructor::getSrcAddr() {
00053     return ascii_to_hex(estts::ax25::AX25_SOURCE_ADDRESS);
00054 }
00055
00060 string ax25_ui_frame_constructor::getSSID1() {
00061     return reinterpret_cast<char const *>(estts::ax25::AX25_SSID1);
00062 }
00063
00068 string ax25_ui_frame_constructor::getControl() {
00069     return reinterpret_cast<char const *>(estts::ax25::AX25_CONTROL);
00070 }
00071
00076 string ax25_ui_frame_constructor::getPID() {
00077     return reinterpret_cast<char const *>(estts::ax25::AX25_PID);
00078 }
00079
00084 string ax25_ui_frame_constructor::getInfoField() {
00085     auto info_field = this->build_info_field();
00086     return info_field;
00087 }
00088
00089 // ~~~~~~~~~~~ ENCODING ~~~~~~~~~~~
00090
00095 string ax25_ui_frame_constructor::construct_ax25() {
00096     std::stringstream frameStream;
00097
00098     // Preamble
00099     for (int i = 0; i < 8; i++)
00100         frameStream << getFlag();
00101
00102     frameStream << getFlag();
00103     // Destination address (hex encoded)
00104     frameStream << getDestAddr();
00105     frameStream << getSSID0();
00106     // Source address (hex encoded)
00107     frameStream << getSrcAddr();
00108     frameStream << getSSID1();
00109     frameStream << getControl();
00110     frameStream << getPID();
00111     std::string info_field = getInfoField();
00112     frameStream << info_field;
00113     frameStream << calculate_crc16_ccit(info_field);
00114     frameStream << getFlag();
00115
00116     // Postamble
00117     for (int i = 0; i < 3; i++)
00118         frameStream << getFlag();
00119
00120     SPDLOG_TRACE("Built AX.25 frame with value {}", frameStream.str());
00121
00122     std::string encoded_frame = this->encode_ax25_frame(frameStream.str());
00123
00124     SPDLOG_TRACE("Encoded AX.25 frame to {}", encoded_frame);
00125
00126     return encoded_frame;
00127 }
00128
00134 std::string ax25_ui_frame_constructor::perform_nrzi_encoding(std::string raw) {
00135     // TODO ESTTS-145 Integrate NRZI encoding - Place NRZI encoding in this function
00136     return raw;
00137 }
00138
```

```
00144 std::string ax25_ui_frame_constructor::calculate_crc16_ccit(std::string value) {
00145     // TODO ESTTS-144 Integrate CRC16-CCIT calculation - Place code in this function.
00146     return "";
00147 }
00148
00154 std::string ax25_ui_frame_constructor::scramble_frame(std::string raw) {
00155     // TODO ESTTS-146 Integrate EnduroSat scrambler - Place scrambling code in this function
00156     return raw;
00157 }
00158
00164 std::string ax25_ui_frame_constructor::encode_ax25_frame(std::string raw) {
00165     std::string encoded = this->scramble_frame(this->perform_nrzi_encoding(std::move(raw)));
00166     // TODO Add logic to perform bit stuffing in accordance with ISO AX.25 Frame Specification
00167     return encoded;
00168 }
```

7.71 src/tnc_emulator/ax25_ui_frame_constructor.h File Reference

```
#include <string>
#include <sstream>
#include <unordered_map>
#include "info_field.h"
#include "bin_converter.h"
```

Classes

- class ax25_ui_frame_constructor

7.72 ax25_ui_frame_constructor.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Cody on 12/17/2021.
00003 //
00004
00005 #ifndef ES2_ESTTS_CPP_FRAME_CONSTRUCTOR_H
00006 #define ES2_ESTTS_CPP_FRAME_CONSTRUCTOR_H
00007
00008 #include <string>
00009 #include <sstream>
00010 #include <unordered_map>
00011 #include "info_field.h"
00012 #include "bin_converter.h"
00013
00014 class ax25_ui_frame_constructor : virtual public info_field {
00015 private:
00016
00017     /* Getters for Header Field */
00018     static std::string getFlag();
00019
00020     static std::string getDestAddr();
00021
00022     static std::string getSSID0();
00023
00024     static std::string getSrcAddr();
00025
00026     static std::string getSSID1();
00027
00028     static std::string getControl();
00029
00030     static std::string getPID();
00031
00032     std::string getInfoField();
00033
00034     static std::string perform_nrzi_encoding(std::string raw);
00035
00036     static std::string calculate_crc16_ccit(std::string value);
00037
```

```
00038     static std::string scramble_frame(std::string raw);
00039
00040 protected:
00041
00042     std::string encode_ax25_frame(std::string raw);
00043
00044 public:
00045     /* Constructors */
00046     explicit ax25_ui_frame_constructor(estts::command_object *command) : info_field(command) {}
00047
00048     /* Encoded AX.25 Frame Constructor */
00049     std::string construct_ax25();
00050 };
00051
00052 #endif
```

7.73 src/tnc_emulator/ax25_ui_frame_destructor.cpp File Reference

```
#include <iostream>
#include <sstream>
#include "ax25_ui_frame_destructor.h"
#include "helper.h"
```

7.74 ax25_ui_frame_destructor.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/28/21.
00003 //
00004
00005 #include <iostream>
00006 #include <sstream>
00007 #include "ax25_ui_frame_destructor.h"
00008 #include "helper.h"
00009
00010 estts::Status ax25_ui_frame_destructor::decode_frame(const std::string &frame) {
00011     SPDLOG_DEBUG("Found frame: {}", frame);
00012
00013     if (estts::ES_OK != validate_header(frame)) {
00014         spdlog::error("Frame validation failed");
00015         return estts::ES_UNSUCCESSFUL;
00016     }
00017
00018     int crc_size = 0; // Usually 16 bits todo when CRC is checked and tests are added including a CRC,
      subtract it here
00019
00020     auto resp_object = build_telemetry_object(frame.substr(32));
00021
00022     telemetry.push_back(resp_object);
00023
00024     return estts::ES_OK;
00025 }
00026
00027 estts::Status ax25_ui_frame_destructor::check_source(const std::string &source) {
00028     if (hex_to_ascii(source) == estts::ax25::AX25_DESTINATION_ADDRESS)
00029         return estts::ES_OK;
00030     spdlog::error("Source: Expected {}; Got {}", estts::ax25::AX25_DESTINATION_ADDRESS,
      hex_to_ascii(source));
00031     return estts::ES_UNSUCCESSFUL;
00032 }
00033
00034 estts::Status ax25_ui_frame_destructor::check_destination(const std::string &dest) {
00035     if (hex_to_ascii(dest) == estts::ax25::AX25_SOURCE_ADDRESS)
00036         return estts::ES_OK;
00037     spdlog::error("Destination: Expected {}; Got {}", estts::ax25::AX25_SOURCE_ADDRESS,
      hex_to_ascii(dest));
00038     return estts::ES_UNSUCCESSFUL;
00039 }
00040
00041 estts::Status ax25_ui_frame_destructor::check_ssid0(const std::string &ssid) {
00042     if (ssid == estts::ax25::AX25_SSID0)
00043         return estts::ES_OK;
```

```
00044     spdlog::error("SSID0: Expected {}; Got {}", estts::ax25::AX25_SSID0, ssid);
00045     return estts::ES_UNSUCCESSFUL;
00046 }
00047
00048 estts::Status ax25_ui_frame_destructor::check_ssid1(const std::string &ssid) {
00049     if (ssid == estts::ax25::AX25_SSID1)
00050         return estts::ES_OK;
00051     spdlog::error("SSID1: Expected {}; Got {}", estts::ax25::AX25_SSID1, ssid);
00052     return estts::ES_UNSUCCESSFUL;
00053 }
00054
00055 estts::Status ax25_ui_frame_destructor::check_control(const std::string &control) {
00056     if (control == estts::ax25::AX25_CONTROL)
00057         return estts::ES_OK;
00058     spdlog::error("Control: Expected {}; Got {}", estts::ax25::AX25_CONTROL, control);
00059     return estts::ES_UNSUCCESSFUL;
00060 }
00061
00062 estts::Status ax25_ui_frame_destructor::check_pid(const std::string &pid) {
00063     if (pid == estts::ax25::AX25_PID)
00064         return estts::ES_OK;
00065     spdlog::error("PID: Expected {}; Got {}", estts::ax25::AX25_PID, pid);
00066     return estts::ES_UNSUCCESSFUL;
00067 }
00068
00069 estts::Status ax25_ui_frame_destructor::check_crc(const std::string &frame, const std::string &crc) {
00070     // TODO check the CRC against the calculated CRC of the entire frame
00071     return estts::ES_OK;
00072 }
00073
00074 estts::Status ax25_ui_frame_destructor::validate_header(const std::string &frame) {
00075     if (estts::ES_OK != check_destination(frame.substr(0, 12)))
00076         return estts::ES_UNSUCCESSFUL;
00077     if (estts::ES_OK != check_ssid0(frame.substr(12, 2)))
00078         return estts::ES_UNSUCCESSFUL;
00079     if (estts::ES_OK != check_source(frame.substr(14, 12)))
00080         return estts::ES_UNSUCCESSFUL;
00081     if (estts::ES_OK != check_ssid1(frame.substr(26, 2)))
00082         return estts::ES_UNSUCCESSFUL;
00083     if (estts::ES_OK != check_control(frame.substr(28, 2)))
00084         return estts::ES_UNSUCCESSFUL;
00085     if (estts::ES_OK != check_pid(frame.substr(30, 2)))
00086         return estts::ES_UNSUCCESSFUL;
00087     SPDLOG_TRACE("Frame header validated. Continuing");
00088     return estts::ES_OK;
00089 }
00090
00091 estts::Status ax25_ui_frame_destructor::build_telemetry_objects() {
00092     try {
00093         for (;;) {
00094             // First, find the start of the frame using the flag.
00095             // Assuming that the frame transmission had no loss, this method should work.
00096             unsigned long frame_start_flag;
00097             auto raw_length = raw_frame.length();
00098             if (raw_length < 1) {
00099                 spdlog::error("Trying to decode empty frame. Exiting");
00100                 return estts::ES_UNSUCCESSFUL;
00101             }
00102             for (int i = 0; i < raw_length; i++) {
00103                 if (raw_frame[i] == estts::ax25::AX25_FLAG[0] && raw_frame[i + 1] ==
      estts::ax25::AX25_FLAG[1]) {
00104                     i++;
00105                 } else {
00106                     frame_start_flag = i;
00107                     break;
00108                 }
00109             }
00110
00111             // Now, we need to find the end of the frame. We can do this using the flag, and safely
      assume that if the
00112             // trailing flag is missing, the frame is likely corrupted.
00113             unsigned long frame_end;
00114             for (int i = 0; i < raw_length; i++) {
00115                 if (i + 1 >= raw_length) {
00116                     SPDLOG_TRACE("No more frames found.");
00117                     return estts::ES_OK;
00118                 }
00119                 if (raw_frame[i] == estts::ax25::AX25_FLAG[0] && raw_frame[i + 1] ==
      estts::ax25::AX25_FLAG[1]) {
00120                     if (estts::ax25::AX25_FLAG[1] != raw_frame[i - 1] && i > 0) {
00121                         frame_end = i;
00122                         break;
00123                     }
00124                 }
00125             }
00126
00127             // Try decoding the frame.
```

```
00128             auto frame = raw_frame.substr(frame_start_flag, frame_end - frame_start_flag);
00129             if (estts::ES_OK == decode_frame(frame)) {
00130                 SPDLOG_TRACE("Trimming at index {} and looking for more frames", frame_end);
00131                 raw_frame.erase(0, frame_end);
00132             } else {
00133                 // Then, find the start of the frame using the src/dest callsigns. This is a failsafe
00134                 // method in case the transmission had loss
00135                 std::stringstream callsign_and_ssid;
00136                 // Remember that we're expecting the satellite to be the source and ground to be the
    destination
00137                 callsign_and_ssid << ascii_to_hex(estts::ax25::AX25_SOURCE_ADDRESS) <<
    estts::ax25::AX25_SSID0 <<
00138                                   ascii_to_hex(estts::ax25::AX25_DESTINATION_ADDRESS) <<
    estts::ax25::AX25_SSID1
00139                                   << std::endl;
00140                 auto temp = raw_frame.find(callsign_and_ssid.str());
00141                 auto frame_start_callsign = (temp != std::string::npos) ? -1 : temp;
00142                 break;
00143             }
00144         }
00145     }
00146     catch (const std::exception &e) {
00147         spdlog::error("Failed to find frame, found exception {}", e.what());
00148         return estts::ES_UNSUCCESSFUL;
00149     }
00150 }
00151
00152 std::vector<estts::telemetry_object *> ax25_ui_frame_destructor::destruct_ax25() {
00153     if (estts::ES_OK != build_telemetry_objects()) {
00154         return {};
00155     }
00156     return this->telemetry;
00157 }
```

7.75 src/tnc_emulator/ax25_ui_frame_destructor.h File Reference

```
#include <string>
#include <utility>
#include "constants.h"
#include "info_field.h"
```

Classes

- class ax25_ui_frame_destructor

7.76 ax25_ui_frame_destructor.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/28/21.
00003 //
00004
00005 #ifndef ESTTS_AX25_UI_FRAME_DESTRUCTOR_H
00006 #define ESTTS_AX25_UI_FRAME_DESTRUCTOR_H
00007
00008 #include <string>
00009 #include <utility>
00010 #include "constants.h"
00011 #include "info_field.h"
00012
00013 class ax25_ui_frame_destructor : virtual public info_field {
00014 private:
00015     std::vector<estts::telemetry_object *> telemetry;
00016
00017     estts::Status decode_frame(const std::string &frame);
00018
00019     static estts::Status validate_header(const std::string &frame);
00020
```

```
00021     static estts::Status check_source(const std::string &source);
00022
00023     static estts::Status check_destination(const std::string &dest);
00024
00025     static estts::Status check_ssid0(const std::string &ssid);
00026
00027     static estts::Status check_ssid1(const std::string &ssid);
00028
00029     static estts::Status check_control(const std::string &control);
00030
00031     static estts::Status check_pid(const std::string &pid);
00032
00033     static estts::Status check_crc(const std::string &frame, const std::string &crc);
00034
00035     estts::Status build_telemetry_objects();
00036
00037     std::string raw_frame;
00038
00039 public:
00040     explicit ax25_ui_frame_destructor(std::string raw) : info_field() { raw_frame = std::move(raw); }
00041
00042     std::vector<estts::telemetry_object *> destruct_ax25();
00043 };
00044
00045
00046 #endif //ESTTS_AX25_UI_FRAME_DESTRUCTOR_H
```

7.77 src/tnc_emulator/info_field.cpp File Reference

```
#include <iostream>
#include <string>
#include <sstream>
#include "info_field.h"
```

7.78 info_field.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Cody on 12/17/2021.
00003 //
00004
00005 #include <iostream>
00006 #include <string>
00007 #include <sstream>
00008 #include "info_field.h"
00009
00010 using std::cerr;
00011 using std::endl;
00012 using std::string;
00013
00018 string info_field::getAddress() {
00019     SPDLOG_TRACE("Setting info field address to {}", this->command->address);
00020     auto address = std::to_string(this->command->address);
00021     if (this->command->sequence < 10)
00022         address.insert(0, "0");
00023     return address;
00024 }
00025
00030 string info_field::getTimeStamp() {
00031     SPDLOG_TRACE("Setting info field timestamp to {}", this->command->timeStamp);
00032     return std::to_string(this->command->timeStamp);
00033 }
00034
00039 string info_field::getSequence() {
00040     SPDLOG_TRACE("Setting info field frame sequence to {}", this->command->sequence);
00041     auto sequence = std::to_string(this->command->sequence);
00042     if (this->command->sequence < 10)
00043         sequence.insert(0, "0");
00044     return sequence;
00045 }
00046
00051 string info_field::getCommandID() {
```

```
00052     SPDLOG_TRACE("Setting info field command ID to {}", this->command->commandID);
00053     auto command_id = std::to_string(this->command->commandID);
00054     if (this->command->commandID < 10)
00055         command_id.insert(0, "0");
00056     return command_id;
00057 }
00058
00063 string info_field::getMethod() {
00064     SPDLOG_TRACE("Setting info field method to {}", this->command->method);
00065     auto method = std::to_string(this->command->method);
00066     return method;
00067 }
00068
00073 string info_field::getData() {
00074     if (this->command->data != nullptr) {
00075         SPDLOG_TRACE("Setting info field data to {}", this->command->data);
00076         return reinterpret_cast<char const *>(this->command->data);
00077     } else return "";
00078 }
00079
00084 string info_field::build_info_field() {
00085     std::stringstream infoFieldStream;
00086     infoFieldStream « getAddress() « getTimeStamp() « getSequence() « getCommandID() « getMethod() «
      getData();
00087     SPDLOG_DEBUG("SAPI Info Field encoded to {}", infoFieldStream.str());
00088
00089     return infoFieldStream.str();
00090 }
00091
00092 estts::telemetry_object *info_field::build_telemetry_object(std::string info_field) {
00093     auto resp = new estts::telemetry_object;
00094
00095     int address;
00096     std::istringstream(info_field.substr(0, 2)) » address;
00097     resp->address = address;
00098
00099     int timestamp;
00100     std::istringstream(info_field.substr(2, 4)) » timestamp;
00101     resp->timeStamp = timestamp;
00102
00103     int sequence;
00104     std::istringstream(info_field.substr(6, 2)) » sequence;
00105     resp->sequence = sequence;
00106
00107     int command_id;
00108     std::istringstream(info_field.substr(8, 2)) » command_id;
00109     resp->commandID = command_id;
00110
00111     int response_code;
00112     std::istringstream(info_field.substr(10, 1)) » response_code;
00113     resp->response_code = response_code;
00114
00115     // resp->data = info_field.substr(12).c_str(); todo figure this shit out
00116
00117     SPDLOG_TRACE("Frame info field decoded successfully. Telemetry object stored at {}",
      static_cast<const void*>(resp));
00118
00119     return resp;
00120 }
```

7.79 src/tnc_emulator/info_field.h File Reference

```
#include <string>
#include <unordered_map>
#include <utility>
#include "bin_converter.h"
#include "constants.h"
```

Classes

- class info_field

7.80 info_field.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Cody on 12/17/2021.
00003 //
00004
00005 #ifndef ES2_ESTTS_CPP_INFOFIELD_H
00006 #define ES2_ESTTS_CPP_INFOFIELD_H
00007
00008 #include <string>
00009 #include <unordered_map>
00010 #include <utility>
00011 #include "bin_converter.h"
00012 #include "constants.h"
00013
00014 class info_field {
00015 private:
00016     /* Getters */
00017     std::string getData();
00018
00019     std::string getAddress();
00020
00021     std::string getTimeStamp();
00022
00023     std::string getSequence();
00024
00025     std::string getCommandID();
00026
00027     std::string getMethod();
00028
00029 protected:
00030     estts::command_object *command;
00031
00032     /* Encoded Information Field Getter */
00033     std::string build_info_field();
00034
00035     estts::telemetry_object *build_telemetry_object(std::string info_field);
00036
00037     explicit info_field() : command(nullptr) {}
00038
00039     explicit info_field(estts::command_object *esttsCommand) : command(esttsCommand) {}
00040 };
00041
00042 #endif
```

7.81 src/utils/bin_converter.cpp File Reference

```
#include <iostream>
#include <cstring>
#include <sstream>
#include "bin_converter.h"
```

7.82 bin_converter.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Cody on 12/17/2021.
00003 //
00004
00005 #include <iostream>
00006 #include <cstring>
00007 #include <sstream>
00008 #include "bin_converter.h"
00009
00010 using std::cerr;
00011 using std::endl;
00012 using std::stringstream;
00013 using std::string;
00014 using std::hex;
```

```
00015
00021 string bin_converter::toBinary(const short int size, const string &hexField) {
00022     string bitStream;
00023     int bitPadding = size - ((hexField.length() - 2) * 4);
00024
00025     /* If padding zeros are needed, prepend them to the string.
00026      * Else if there is an invalid number of hex values,
00027      * send an error and return zero's. */
00028     if (hexField.substr(0, 2) != "0x" || bitPadding < 0) {
00029         cerr « "Error [info_field] - Invalid hex value: " « hexField « "\nExpected size: " « size «
      endl;
00030
00031         return string(size, '0');
00032     } else if (bitPadding > 0) {
00033         bitStream += string(bitPadding, '0');
00034     }
00035
00036     /* Convert each hex character to binary */
00037     for (int i = 2; i < hexField.length(); ++i)
00038         bitStream += hexToBinMap.find(hexField[i])->second;
00039
00040     return bitStream;
00041 }
00042
00048 string bin_converter::toBinary(const string &hexField) {
00049     string bitStream;
00050
00051     /* Convert each hex character to binary */
00052     for (unsigned int i = 2; i < hexField.length(); ++i)
00053         bitStream += hexToBinMap.find(hexField[i])->second;
00054
00055     return bitStream;
00056 }
00057
00063 string bin_converter::toBinary(const unsigned char field[]) {
00064     stringstream hexStream;
00065
00066     /* Convert each character in the array into a hex string and append them together */
00067     hexStream « "0x" « hex;
00068     for (int i = 0; i < strlen((char *) field); ++i)
00069         hexStream « (int) field[i];
00070
00071     return toBinary(hexStream.str());
00072 }
00073
00079 string bin_converter::toBinary(const unsigned char field) {
00080     stringstream hexStream;
00081
00082     hexStream « "0x" « hex « (int) field;
00083
00084     return toBinary(hexStream.str());
00085 }
```

7.83 src/utils/bin_converter.h File Reference

```
#include <string>
#include <unordered_map>
```

Classes

- class bin_converter

7.84 bin_converter.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Cody on 12/17/2021.
00003 //
00004
```

```
00005 #ifndef ESTTS_BIN_CONVERTER_H
00006 #define ESTTS_BIN_CONVERTER_H
00007
00008 #include <string>
00009 #include <unordered_map>
00010
00011 class bin_converter {
00012 private:
00013     /* Container used to convert Hex characters (Key) to Binary strings (Value) */
00014     const std::unordered_map<char, std::string> hexToBinMap = {
00015             {'0', "0000"},
00016             {'1', "0001"},
00017             {'2', "0010"},
00018             {'3', "0011"},
00019             {'4', "0100"},
00020             {'5', "0101"},
00021             {'6', "0110"},
00022             {'7', "0111"},
00023             {'8', "1000"},
00024             {'9', "1001"},
00025             {'a', "1010"},
00026             {'b', "1011"},
00027             {'c', "1100"},
00028             {'d', "1101"},
00029             {'e', "1110"},
00030             {'f', "1111"}
00031     };
00032
00033     /* Container used to convert Binary strings (Key) to Hex characters (Value) */
00034     const std::unordered_map<std::string, char> BinToHexMap = {
00035             {"0000", '0'},
00036             {"0001", '1'},
00037             {"0010", '2'},
00038             {"0011", '3'},
00039             {"0100", '4'},
00040             {"0101", '5'},
00041             {"0110", '6'},
00042             {"0111", '7'},
00043             {"1000", '8'},
00044             {"1001", '9'},
00045             {"1010", 'a'},
00046             {"1011", 'b'},
00047             {"1100", 'c'},
00048             {"1101", 'd'},
00049             {"1110", 'e'},
00050             {"1111", 'f'}
00051     };
00052 public:
00053     // ~~~~~~~~~~ Convert To Binary  ~~~~~~~~~~
00054
00055     /* Converts hex string to binary and also checks size */
00056     std::string toBinary(short int size, const std::string &hexField);
00057
00058     /* Converts hex string to binary */
00059     std::string toBinary(const std::string &hexField);
00060
00061     /* Converts an unsigned char array to binary */
00062     std::string toBinary(const unsigned char field[]);
00063
00064     /* Converts an unsigned char to binary */
00065     std::string toBinary(unsigned char field);
00066 };
00067
00068 #endif //ESTTS_BIN_CONVERTER_H
```

7.85 src/utils/constants.h File Reference

`#include "spdlog/spdlog.h"`

Classes

- struct estts::es2_telemetry::eps::vitals
- struct estts::es2_telemetry::eps::eps_voltage
- struct estts::es2_telemetry::eps::eps_current

- struct estts::es2_telemetry::eps::eps_5Vbus_current
- struct estts::es2_telemetry::eps::eps_3Vbus_current
- struct estts::es2_telemetry::eps::eps_externalTemp_sensor5
- struct estts::es2_telemetry::eps::eps_externalTemp_sensor6
- struct estts::es2_telemetry::eps::eps_externalTemp_sensor7
- struct estts::es2_telemetry::eps::eps_batteryTemp_sensor1
- struct estts::es2_telemetry::eps::eps_batteryTemp_sensor2
- struct estts::es2_telemetry::eps::eps_batteryTemp_sensor3
- struct estts::es2_telemetry::eps::eps_batteryTemp_sensor4
- class estts::endurosat::esttc_const
- struct estts::command_object
- struct estts::telemetry_object
- struct estts::dispatched_command
- struct estts::waiting_command

Namespaces

- namespace estts
- namespace estts::cosmos
- namespace estts::ti_serial
- namespace estts::ti_socket
- namespace estts::ax25
- namespace estts::telem_handler
- namespace estts::estts_response_code
- namespace estts::es2_endpoint
- namespace estts::dispatcher
- namespace estts::es2_commands
- namespace estts::es2_commands::acs
- namespace estts::es2_commands::eps
- namespace estts::es2_commands::mde
- namespace estts::es2_commands::crp
- namespace estts::es2_commands::obc
- namespace estts::es2_commands::method
- namespace estts::es2_telemetry
- namespace estts::es2_telemetry::eps
- namespace estts::es2_telemetry::acs
- namespace estts::endurosat

Macros

- #define SPDLOG_ACTIVE_LEVEL 0
- #define MAX_SERIAL_READ 256

Typedefs

- typedef struct estts::command_object estts::command_object
- typedef struct estts::telemetry_object estts::telemetry_object
- typedef struct estts::dispatched_command estts::dispatched_command
- typedef struct estts::waiting_command estts::waiting_command
- typedef std::function< std::string(estts::command_object *, std::function< estts::Status(std::vector< estts::telemetry_object * >)>)> estts::dispatch_fct

Enumerations

- enum estts::Status {
  estts::ES_OK = 0 , estts::ES_SUCCESS = 0 , estts::ES_UNSUCCESSFUL = 1 , estts::ES_UNINITIALIZED = 2 ,
  estts::ES_MEMORY_ERROR = 3 , estts::ES_WAITING = 3 , estts::ES_BAD_OPTION = 405 , estts::ES_UNAUTHORIZED = 403 ,
  estts::ES_SERVER_ERROR = 500 , estts::ES_INPROGRESS = 300 , estts::ES_NOTFOUND = 404 }
- enum estts::endurosat::PIPE_State { estts::endurosat::PIPE_OFF = 0 , estts::endurosat::PIPE_WAITING = 1 , estts::endurosat::PIPE_ON = 2 }

Variables

- const char estts::REMOVABLE_STORAGE_NAME [ ] = "Samsung_T5"
- const int estts::ESTTS_MAX_RETRIES = 2
- const int estts::ESTTS_RETRY_WAIT_SEC = 1
- const int estts::ESTTS_AWAIT_RESPONSE_PERIOD_SEC = 5
- const int estts::ESTTS_SATELLITE_CONNECTION_TIMEOUT_MIN = 90
- const int estts::ESTTS_CHECK_SATELLITE_INRANGE_INTERVAL_SEC = 30
- const int estts::ESTTS_REQUEST_SESSION_TIMEOUT_SECONDS = 300
- const char estts::cosmos::COSMOS_SERVER_ADDR [ ] = "172.30.95.164"
- const int estts::cosmos::COSMOS_PRIMARY_CMD_TELEM_PORT = 65432
- const int estts::cosmos::COSMOS_GROUNDSTATION_CMD_TELEM_PORT = 8046
- const int estts::cosmos::COSMOS_SATELLITE_TXVR_CMD_TELEM_PORT = 55927
- const char estts::ti_serial::TI_SERIAL_ADDRESS [ ] = "/dev/cu.usbserial-A10JVB3P"
- const int estts::ti_socket::MAX_RETRIES = 2
- const int estts::ti_socket::WAIT_TIME_SEC = 2
- const int estts::ti_socket::TI_SOCKET_BUF_SZ = 1024
- const char estts::ti_socket::TI_SOCKET_ADDRESS [ ] = "127.0.0.1"
- const int estts::ti_socket::TI_SOCKET_PORT = 65548
- const char estts::ax25::AX25_FLAG [ ] = "7E"
- const char estts::ax25::AX25_DESTINATION_ADDRESS [ ] = "NABCDE"
- const char estts::ax25::AX25_SSID0 [ ] = "E0"
- const char estts::ax25::AX25_SOURCE_ADDRESS [ ] = "NEDCBA"
- const char estts::ax25::AX25_SSID1 [ ] = "E1"
- const char estts::ax25::AX25_CONTROL [ ] = "03"
- const char estts::ax25::AX25_PID [ ] = "F0"
- const char estts::ax25::NEW_SESSION_FRAME [ ] = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
- const char estts::ax25::END_SESSION_FRAME [ ] = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB"
- const char estts::telem_handler::TELEM_HANDLER_STATE_FILE [ ] = "es2_state.json"
- const int estts::estts_response_code::SUCCESS = 0
- const int estts::estts_response_code::UNRECOGNIZED_REQUEST = 1
- const int estts::estts_response_code::OBC_FAILURE = 2
- const int estts::es2_endpoint::ES_OBC = 01
- const int estts::es2_endpoint::ES_EPS = 02
- const int estts::es2_endpoint::ES_ACS = 03
- const int estts::es2_endpoint::ES_CRP = 05
- const int estts::es2_endpoint::ES_MDE = 04
- const int estts::es2_endpoint::ES_OFFLINE_LOG = 05
- const int estts::es2_endpoint::ES_TELEMETRY = 06
- const int estts::dispatcher::MAX_COMPLETED_CACHE = 20
- const int estts::es2_commands::acs::ACS_GET_GPS_LAT = 01
- const int estts::es2_commands::acs::ACS_GET_GPS_LONG = 02
- const int estts::es2_commands::acs::ACS_GET_POS = 03

- const int estts::es2_commands::acs::ACS_DEP_MAG_BOOM = 07
- const int estts::es2_commands::acs::ACS_ENABLE = 10
- const int estts::es2_commands::acs::ACS_POWER = 11
- const int estts::es2_commands::acs::ACS_SET_CTRL_MODE = 13
- const int estts::es2_commands::acs::ACS_SET_EST_MODE = 14
- const int estts::es2_commands::acs::ACS_SET_MAG_MNT = 33
- const int estts::es2_commands::acs::ACS_SET_MAG_MNT_MTRX = 34
- const int estts::es2_commands::acs::ACS_SET_INERTIA = 41
- const int estts::es2_commands::acs::ACS_SAVE_CONFIG = 63
- const int estts::es2_commands::acs::ACS_SET_ATT_ANG = 146
- const int estts::es2_commands::acs::ACS_SET_ANG_RATE = 147
- const int estts::es2_commands::acs::ACS_GET_MAGNET = 151
- const int estts::es2_commands::acs::ACS_RATE_SENSE_RATE = 155
- const int estts::es2_commands::acs::ACS_SET_MAGNETORQUER = 157
- const int estts::es2_commands::acs::ACS_GET_MAGNETO = 170
- const int estts::es2_commands::acs::ACS_GET_CC_CURRENT = 172
- const int estts::es2_commands::acs::ACS_EST_ANG_RATES_FINE = 201
- const int estts::es2_commands::eps::EPS_GET_HEALTH = 01
- const int estts::es2_commands::eps::EPS_GET_COMMAND_43 = 43
- const int estts::es2_commands::eps::EPS_GET_BATTERY_VOLTAGE = 1
- const int estts::es2_commands::eps::EPS_GET_BATTERY_CURRENT = 2
- const int estts::es2_commands::eps::EPS_GET_5VBUS_CURRENT = 15
- const int estts::es2_commands::eps::EPS_GET_3VBUS_CURRENT = 14
- const int estts::es2_commands::eps::EPS_GET_TEMP_SENSOR5 = 38
- const int estts::es2_commands::eps::EPS_GET_TEMP_SENSOR6 = 39
- const int estts::es2_commands::eps::EPS_GET_TEMP_SENSOR7 = 40
- const int estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR1 = 19
- const int estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR2 = 20
- const int estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR3 = 21
- const int estts::es2_commands::eps::EPS_GET_BATTERY_TEMP_SENSOR4 = 22
- const int estts::es2_commands::mde::MDE_GET_STATUS = 01
- const int estts::es2_commands::crp::CRP_GET_DATA = 01
- const int estts::es2_commands::obc::OBC_GET_HEALTH = 01
- const int estts::es2_commands::method::ES_READ = 0
- const int estts::es2_commands::method::ES_WRITE = 1
- const int estts::endurosat::PIPE_DURATION_SEC = 10
- const int estts::endurosat::MAX_RETRIES = 2
- const int estts::endurosat::WAIT_TIME_SEC = 2
- const int estts::endurosat::ES_BAUD = 115200
- const int estts::endurosat::MAX_ES_TXVR_TEMP = 50

7.85.1 Macro Definition Documentation

7.85.1.1 MAX_SERIAL_READ

```
#define MAX_SERIAL_READ 256
```

Definition at line 11 of file constants.h.

7.85.1.2 SPDLOG_ACTIVE_LEVEL

```
#define SPDLOG_ACTIVE_LEVEL 0
```

Definition at line 8 of file constants.h.

7.86 constants.h

Go to the documentation of this file.

```
00001 /* Copyright © EagleSat II - Embry Riddle Aeronautical University - All rights reserved - 2022 */
00002
00003 #ifndef ESTTS_CONSTANTS_H
00004 #define ESTTS_CONSTANTS_H
00005
00006 // Configure spdlog
00007 #undef SPDLOG_ACTIVE_LEVEL
00008 #define SPDLOG_ACTIVE_LEVEL 0
00009 #include "spdlog/spdlog.h"
00010
00011 #define MAX_SERIAL_READ 256
00012 namespace estts {
00013     const char REMOVABLE_STORAGE_NAME[] = "Samsung_T5";
00014
00015     const int ESTTS_MAX_RETRIES = 2;
00016     const int ESTTS_RETRY_WAIT_SEC = 1;
00017     const int ESTTS_AWAIT_RESPONSE_PERIOD_SEC = 5;
00018     const int ESTTS_SATELLITE_CONNECTION_TIMEOUT_MIN = 90;
00019     const int ESTTS_CHECK_SATELLITE_INRANGE_INTERVAL_SEC = 30;
00020     const int ESTTS_REQUEST_SESSION_TIMEOUT_SECONDS = 300;
00021
00022     namespace cosmos {
00023         const char COSMOS_SERVER_ADDR[] = "172.30.95.164"; // 172.30.95.164 172.19.35.160
00024         const int COSMOS_PRIMARY_CMD_TELEM_PORT = 65432;
00025         const int COSMOS_GROUNDSTATION_CMD_TELEM_PORT = 8046;
00026         const int COSMOS_SATELLITE_TXVR_CMD_TELEM_PORT = 55927;
00027     }
00028
00029     namespace ti_serial {
00030         const char TI_SERIAL_ADDRESS[] = "/dev/cu.usbserial-A10JVB3P";
00031     }
00032
00033     namespace ti_socket {
00034         const int MAX_RETRIES = 2;
00035         const int WAIT_TIME_SEC = 2;
00036         const int TI_SOCKET_BUF_SZ = 1024;
00037         const char TI_SOCKET_ADDRESS[] = "127.0.0.1";
00038         const int TI_SOCKET_PORT = 65548;
00039     }
00040
00041     /* AX.25 Related constants */
00042     namespace ax25 {
00043         const char AX25_FLAG[] = "7E"; // Flag is constant
00044         const char AX25_DESTINATION_ADDRESS[] = "NABCDE"; // Max 48-bit (6-byte)
00045         const char AX25_SSID0[] = "E0";
00046         const char AX25_SOURCE_ADDRESS[] = "NEDCBA"; // Max 48-bit (6-byte)
00047         const char AX25_SSID1[] = "E1";
00048         const char AX25_CONTROL[] = "03"; // 03 = Unnumbered Information
00049         const char AX25_PID[] = "F0"; // F0 = No layer 3 protocol implemented
00050
00051         const char NEW_SESSION_FRAME[] = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
00052         const char END_SESSION_FRAME[] = "BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB";
00053     }
00054
00055     namespace telem_handler {
00056         const char TELEM_HANDLER_STATE_FILE[] = "es2_state.json";
00057     }
00058
00059     namespace estts_response_code {
00060         const int SUCCESS = 0;
00061         const int UNRECOGNIZED_REQUEST = 1;
00062         const int OBC_FAILURE = 2;
00063     }
00064
00065     /* Endpoint names for all communication systems */
00066     namespace es2_endpoint {
00067         const int ES_OBC = 01;
00068         const int ES_EPS = 02;
00069         const int ES_ACS = 03;
```

```
00070         const int ES_CRP = 05;
00071         const int ES_MDE = 04;
00072         const int ES_OFFLINE_LOG = 05;
00073         const int ES_TELEMETRY = 06;
00074     }
00075
00076     /* Generic response code enumeration for return codes */
00077     enum Status {
00078         ES_OK = 0,
00079         ES_SUCCESS = 0,
00080         ES_UNSUCCESSFUL = 1,
00081         ES_UNINITIALIZED = 2,
00082         ES_MEMORY_ERROR = 3,
00083         ES_WAITING = 3,
00084         ES_BAD_OPTION = 405,
00085         ES_UNAUTHORIZED = 403,
00086         ES_SERVER_ERROR = 500,
00087         ES_INPROGRESS = 300,
00088         ES_NOTFOUND = 404
00089     };
00090
00091     namespace dispatcher {
00092         const int MAX_COMPLETED_CACHE = 20; // Maximum number of completed commands to remember
00093     }
00094
00095     namespace es2_commands {
00096         namespace acs {
00097             const int ACS_GET_GPS_LAT = 01;
00098             const int ACS_GET_GPS_LONG = 02;
00099             const int ACS_GET_POS = 03;
00100             const int ACS_DEP_MAG_BOOM = 07;
00101             const int ACS_ENABLE = 10;
00102             const int ACS_POWER = 11;
00103             const int ACS_SET_CTRL_MODE = 13;
00104             const int ACS_SET_EST_MODE = 14;
00105             const int ACS_SET_MAG_MNT = 33;
00106             const int ACS_SET_MAG_MNT_MTRX = 34;
00107             const int ACS_SET_INERTIA = 41;
00108             const int ACS_SAVE_CONFIG = 63;
00109             const int ACS_SET_ATT_ANG = 146;
00110             const int ACS_SET_ANG_RATE = 147;
00111             const int ACS_GET_MAGNET = 151;
00112             const int ACS_RATE_SENSE_RATE = 155;
00113             const int ACS_SET_MAGNETORQUER = 157;
00114             const int ACS_GET_MAGNETO = 170;
00115             const int ACS_GET_CC_CURRENT = 172;
00116             const int ACS_EST_ANG_RATES_FINE = 201;
00117         }
00118         namespace eps {
00119             const int EPS_GET_HEALTH = 01;
00120             const int EPS_GET_COMMAND_43 = 43;
00121             const int EPS_GET_BATTERY_VOLTAGE = 1;
00122             const int EPS_GET_BATTERY_CURRENT = 2;
00123             const int EPS_GET_5VBUS_CURRENT = 15;
00124             const int EPS_GET_3VBUS_CURRENT = 14;
00125             const int EPS_GET_TEMP_SENSOR5 = 38;
00126             const int EPS_GET_TEMP_SENSOR6 = 39;
00127             const int EPS_GET_TEMP_SENSOR7 = 40;
00128             const int EPS_GET_BATTERY_TEMP_SENSOR1 = 19;
00129             const int EPS_GET_BATTERY_TEMP_SENSOR2 = 20;
00130             const int EPS_GET_BATTERY_TEMP_SENSOR3 = 21;
00131             const int EPS_GET_BATTERY_TEMP_SENSOR4 = 22;
00132
00133         }
00134         namespace mde {
00135             const int MDE_GET_STATUS = 01;
00136         }
00137         namespace crp {
00138             const int CRP_GET_DATA = 01;
00139         }
00140         namespace obc {
00141             const int OBC_GET_HEALTH = 01;
00142         }
00143         namespace method {
00144             const int ES_READ = 0;
00145             const int ES_WRITE = 1;
00146         }
00147     }
00148
00149     namespace es2_telemetry {
00150         namespace eps {
00151             struct vitals {
00152                 double battery_voltage;
00153                 double brownouts;
00154                 double charge_time_mins;
00155             };
00156             struct eps_voltage {
```

```
00157                 double battery_voltage;
00158             };
00159             struct eps_current {
00160                 double battery_current;
00161             };
00162             struct eps_5Vbus_current {
00163                 double bus_current;
00164             };
00165             struct eps_3Vbus_current {
00166                 double bus_current;
00167             };
00168             struct eps_externalTemp_sensor5{
00169                 double external_temperature;
00170             };
00171             struct eps_externalTemp_sensor6{
00172                 double external_temperature;
00173             };
00174             struct eps_externalTemp_sensor7{
00175                 double external_temperature;
00176             };
00177             struct eps_batteryTemp_sensor1{
00178                 double battery_temperature;
00179             };
00180             struct eps_batteryTemp_sensor2{
00181                 double battery_temperature;
00182             };
00183             struct eps_batteryTemp_sensor3{
00184                 double battery_temperature;
00185             };
00186             struct eps_batteryTemp_sensor4{
00187                 double battery_temperature;
00188             };
00189         }
00190         namespace acs {
00191         }
00192     }
00193
00194     namespace endurosat {
00195         const int PIPE_DURATION_SEC = 10;
00196         const int MAX_RETRIES = 2;
00197         const int WAIT_TIME_SEC = 2;
00198         const int ES_BAUD = 115200;
00199         const int MAX_ES_TXVR_TEMP = 50;
00200         enum PIPE_State {
00201             PIPE_OFF = 0,
00202             PIPE_WAITING = 1,
00203             PIPE_ON = 2
00204         };
00205         class esttc_const {
00206         public:
00207             const uint8_t NUM_OF_RETRIES = 5;
00208             const char *HEADER = "ES+";
00209             const char METHOD_READ = 'R';
00210             const char METHOD_WRITE = 'W';
00211             const char METHOD_FIRMWARE_UPDATE = 'D';
00212             const char *ADDRESS = "22";
00213             const char *BLANK = " ";
00214             const char *END = "\r";
00215             const char* DOWNLINK_XOR = "AB7563CD";
00216             const char* UPLINK_XOR = "6ACD3B57";
00217             const char *CMD_SCW = "00"; // Status Control Word
00218             const char *CMD_RADIO_FREQ_CONFIG = "01"; //  Radio Frequency Configuration
00219             const char *CMD_READ_UPTIME = "02"; // Read Uptime E
00220             const char *CMD_READ_TRANS_PCKTS = "03"; // Read Number of Transmitted Packets C
00221             const char *CMD_READ_RECEIV_PCKTS = "04"; // Read Number of Received Packets E
00222             const char *CMD_READ_TRANS_PCKTS_CRC = "05"; // Read Number of Transmitted Packets With
    CRC Error C
00223             const char *CMD_PIPE_MODE_TMOUT_CONFIG = "06"; // Transparent (Pipe) Mode Timeout Period
    Configuration E
00224             const char *CMD_BCN_MSG_TRANS_CONFIG = "07"; //  Beacon Message Transmission Period
    Configuration C
00225             const char *CMD_AUDIO_BCN_P_TRANS = "08"; // Audio Beacon Period Between Transmissions E
00226             const char *CMD_RESTORE = "09"; // Restore Default Values C
00227             const char *CMD_TEMP_VAL = "0A"; // Internal Temperature Sensor Measurement Value E
00228             const char *CMD_I2C_RESIST_CONFIG = "0B"; // I2C Pull-Up Resistors Configuration
    Read/Write C
00229             const char *CMD_TERM_RESIST_CONFIG = "EC"; // Terminating Resistor Configuration
    Read/Write E
00230             const char *CMD_ENABLE_DISABLE_RADIO_CRC = "ED"; // Enabling/Disabling Radio Packet CRC16
    C
00231             const char *CMD_FORCE_BCN_CMD = "EE"; // Force Beacon Command E
00232             const char *CMD_AUTO_AX25_DECODE = "EF"; // Enabling/Disabling Automatic AX.25 Decoding C
00233             const char *CMD_READ_WRITE_I2C = "F1"; // Generic Write and/or Read From an I2C Device E
00234             const char *CMD_ANT_RELEASE_CONFIG = "F2"; // UHF Antenna Release Configuration C
00235             const char *CMD_ANT_READ_WRITE = "F3"; // UHF Antenna Read/Write E
00236             const char *CMD_LOW_PWR_MODE = "F4"; // Low Power Mode C
00237             const char *CMD_DEST_CALL_SIGN = "F5"; // Destination Call Sign E
```

```
00238             const char *CMD_SRC_CALL_SIGN = "F6"; // Source Call Sign C
00239             const char *CMD_READ_SFTWR_VER = "F9"; // Read Software Version Build E
00240             const char *CMD_READ_DVC_PAYLOAD = "FA"; // Read Device Payload Size C
00241             const char *CMD_BCN_MSG_CONFIG = "FB"; // Beacon Message Content Configuration E
00242             const char *CMD_DVC_ADDR_CONFIG = "FC"; // Device Address Configuration C
00243             const char *CMD_FRAM_MEM_READ_WRITE = "FD"; // FRAM Memory Read/Write E
00244             const char *CMD_RADIO_TRANS_PROP_CONFIG = "FE"; // Radio Transceiver Property
      Configuration C
00245             const char *CMD_SECURE_MODE = "FF"; // Secure Mode E
00246             const char *CMD_FRMWR_UPDATE = "AA"; // Firmware Update C
00247
00248             enum SCW_Commands {
00249                 default_mode,
00250                 enable_pipe,
00251                 scw_stopper
00252             };
00253
00254             const char *scw_body[scw_stopper] = {
00255                     "4343", // default_mode - 0011 0011 0000 0011
00256                     "3323" // enable_pipe - 0011 0011 0010 0011
00257             };
00258         };
00259     }
00260
00261     typedef struct command_object {
00262         int address{};
00263         int timeStamp{}; // deprecated
00264         int sequence{};
00265         int commandID{};
00266         int method{};
00267         const char *data{};
00268     } command_object;
00269
00270     typedef struct telemetry_object {
00271         int address{};
00272         int timeStamp{}; // deprecated
00273         int sequence{};
00274         int commandID{};
00275         int response_code{};
00276         const char *data{};
00277     } telemetry_object;
00278
00279     typedef struct dispatched_command {
00280         std::string frame;
00281         command_object * command;
00282         std::vector<telemetry_object *> telem_obj;
00283         std::string telem_str;
00284         Status response_code;
00285         std::string serial_number;
00286         std::function<estts::Status(std::vector<estts::telemetry_object *>)> obj_callback;
00287         std::function<estts::Status(std::string)> str_callback;
00288     } dispatched_command;
00289
00290     typedef struct waiting_command {
00291         std::string frame;
00292         command_object * command; // deprecated
00293         std::string serial_number;
00294         std::function<estts::Status(std::vector<estts::telemetry_object *>)> obj_callback; //
      deprecated
00295         std::function<estts::Status(std::string)> str_callback;
00296     } waiting_command;
00297
00298     typedef std::function<std::string(estts::command_object *,
      std::function<estts::Status(std::vector<estts::telemetry_object *>)>)> dispatch_fct;
00299 }
00300
00301 #endif //ESTTS_CONSTANTS_H
```

7.87 src/utils/helper.cpp File Reference

```
#include <algorithm>
#include <dirent.h>
#include <sstream>
#include <condition_variable>
#include <random>
#include <utility>
#include "helper.h"
```

Functions

- std::string ascii_to_hex (const std::string &in)
- std::string hex_to_ascii (const std::string &hex)
- std::string generate_serial_number ()

  Creates 16-character serial number using C++ random library.
- std::string find_removable_storage ()
- void print_write_trace_msg (unsigned char *message_uc, size_t bytes, const std::string &endpoint)
- void print_read_trace_msg (unsigned char *message_uc, size_t bytes, const std::string &endpoint)

7.87.1 Function Documentation

7.87.1.1 ascii_to_hex()

```
std::string ascii_to_hex (
            const std::string & in )
```

Convert ASCII string to HEX string. Be careful when using this function..

Parameters

in	ASCII string for conversion

Returns

HEX string characters

Definition at line 13 of file helper.cpp.

7.87.1.2 find_removable_storage()

```
std::string find_removable_storage ( )
```

Definition at line 62 of file helper.cpp.

7.87.1.3 generate_serial_number()

```
std::string generate_serial_number ( )
```

Creates 16-character serial number using C++ random library.

Returns

16-character serial number

Definition at line 43 of file helper.cpp.

7.87.1.4 hex_to_ascii()

```
std::string hex_to_ascii (
            const std::string & hex )
```

Convert HEX string to ASCII string. Borrowed from https://www.geeksforgeeks.org/convert-hexadecimal-valu

Parameters

hex	HEX string characters for conversion

Returns

ASCII string

Definition at line 21 of file helper.cpp.

7.87.1.5 print_read_trace_msg()

```
void print_read_trace_msg (
            unsigned char * message_uc,
            size_t bytes,
            const std::string & endpoint )
```

Definition at line 106 of file helper.cpp.

7.87.1.6 print_write_trace_msg()

```
void print_write_trace_msg (
            unsigned char * message_uc,
            size_t bytes,
            const std::string & endpoint )
```

Definition at line 99 of file helper.cpp.

7.88 helper.cpp

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/28/21.
00003 //
00004
00005 #include <algorithm>
00006 #include <dirent.h>
00007 #include <sstream>
00008 #include <condition_variable>
00009 #include <random>
00010 #include <utility>
00011 #include "helper.h"
00012
```

```
00013 std::string ascii_to_hex(const std::string& in) {
00014     std::stringstream ret;
00015     for (auto i : in) {
00016         ret « std::hex « (unsigned)i;
00017     }
00018     return ret.str();
00019 }
00020
00021 std::string hex_to_ascii(const std::string& hex) {
00022     // initialize the ASCII code string as empty.
00023     std::string ascii = "";
00024     for (size_t i = 0; i < hex.length(); i += 2)
00025     {
00026         // extract two characters from hex string
00027         std::string part = hex.substr(i, 2);
00028
00029         // change it into base 16 and
00030         // typecast as the character
00031         char ch = stoul(part, nullptr, 16);
00032
00033         // add this char to final ASCII string
00034         ascii += ch;
00035     }
00036     return ascii;
00037 }
00038
00043 std::string generate_serial_number() {
00044     auto len = 16;
00045     static const char alphanum[] =
00046             "0123456789"
00047             "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
00048             "abcdefghijklmnopqrstuvwxyz";
00049     std::string tmp_s;
00050     tmp_s.reserve(len);
00051     std::random_device dev;
00052     std::mt19937 rng(dev());
00053     std::uniform_int_distribution<std::mt19937::result_type> dist6(1,sizeof(alphanum));
00054
00055     for (int i = 0; i < len; ++i) {
00056         tmp_s += alphanum[dist6(rng) % (sizeof(alphanum) - 1)];
00057     }
00058
00059     return tmp_s;
00060 }
00061
00062 std::string find_removable_storage() {
00063     std::stringstream ssd_dir;
00064 #ifdef __ESTTS_OS_LINUX__
00065     ssd_dir « "/media/";
00066     auto path_found = false;
00067     DIR * d = opendir(ssd_dir.str().c_str());
00068     if (d == nullptr) return "";
00069     struct dirent * dir;
00070     while ((dir = readdir(d)) != nullptr) {
00071         ssd_dir.clear();
00072         if (strcmp(dir->d_name, ".")!=0 && strcmp(dir->d_name, "..")!=0 && !path_found) {
00073             std::stringstream temp_path;
00074             temp_path « ssd_dir.str() « dir->d_name;
00075             DIR * d1 = opendir(temp_path.str().c_str());
00076             if (d1 != nullptr) {
00077                 struct dirent * dir1;
00078                 while ((dir1 = readdir(d1)) != nullptr) {
00079                     if (dir->d_type == DT_DIR && strcmp(dir1->d_name,
      estts::REMOVABLE_STORAGE_NAME)==0) {
00080                         ssd_dir « dir->d_name « "/" « dir1->d_name;
00081                         SPDLOG_INFO("Constructed path to removable storage device - {}",
      ssd_dir.str());
00082                         path_found = true;
00083                     }
00084                 }
00085             }
00086             closedir(d1);
00087         }
00088     }
00089     closedir(d);
00090     if (path_found)
00091         return ssd_dir.str();
00092     else
00093         return "";
00094 #else
00095     return "";
00096 #endif
00097 }
00098
00099 void print_write_trace_msg(unsigned char *message_uc, size_t bytes, const std::string& endpoint) {
00100     std::string message(reinterpret_cast<char*>(message_uc));
00101     std::replace( message.begin(), message.end(), '\r', ' ');
```

```
00102     message.append("\0");
00103     SPDLOG_TRACE("Wrote '{}' (size={} bytes) to {}", message, bytes, endpoint);
00104 }
00105
00106 void print_read_trace_msg(unsigned char *message_uc, size_t bytes, const std::string& endpoint) {
00107     std::string message(reinterpret_cast<char*>(message_uc));
00108     std::replace( message.begin(), message.end(), '\r', ' ');
00109     message.append("\0");
00110     SPDLOG_TRACE("Read '{}' (size={} bytes) from {}", message, bytes, endpoint);
00111 }
```

7.89 src/utils/helper.h File Reference

```
#include <string>
#include "constants.h"
```

Functions

- std::string ascii_to_hex (const std::string &in)
- std::string hex_to_ascii (const std::string &hex)
- std::string generate_serial_number ()

  Creates 16-character serial number using C++ random library.
- std::string find_removable_storage ()
- void print_write_trace_msg (unsigned char *message_uc, size_t bytes, const std::string &endpoint)
- void print_read_trace_msg (unsigned char *message_uc, size_t bytes, const std::string &endpoint)

7.89.1 Function Documentation

7.89.1.1 ascii_to_hex()

```
std::string ascii_to_hex (
            const std::string & in )
```

Convert ASCII string to HEX string. Be careful when using this function..

Parameters

in	ASCII string for conversion

Returns

HEX string characters

Definition at line 13 of file helper.cpp.

7.89.1.2 find_removable_storage()

```
std::string find_removable_storage ( )
```

Definition at line 62 of file helper.cpp.

7.89.1.3 generate_serial_number()

```
std::string generate_serial_number ( )
```

Creates 16-character serial number using C++ random library.

Returns

16-character serial number

Definition at line 43 of file helper.cpp.

7.89.1.4 hex_to_ascii()

```
std::string hex_to_ascii (
            const std::string & hex )
```

Convert HEX string to ASCII string. Borrowed from https://www.geeksforgeeks.org/convert-hexadecimal-valu

Parameters

hex	HEX string characters for conversion

Returns

ASCII string

Definition at line 21 of file helper.cpp.

7.89.1.5 print_read_trace_msg()

```
void print_read_trace_msg (
            unsigned char * message_uc,
            size_t bytes,
            const std::string & endpoint )
```

Definition at line 106 of file helper.cpp.

7.89.1.6 print_write_trace_msg()

```
void print_write_trace_msg (
            unsigned char * message_uc,
            size_t bytes,
            const std::string & endpoint )
```

Definition at line 99 of file helper.cpp.

7.90 helper.h

Go to the documentation of this file.

```
00001 //
00002 // Created by Hayden Roszell on 12/28/21.
00003 //
00004
00005 #ifndef ESTTS_HELPER_H
00006 #define ESTTS_HELPER_H
00007
00008 #include <string>
00009 #include "constants.h"
00010
00016 std::string ascii_to_hex(const std::string& in);
00017
00024 std::string hex_to_ascii(const std::string& hex);
00025
00026 std::string generate_serial_number();
00027
00028 std::string find_removable_storage();
00029
00030 void print_write_trace_msg(unsigned char *message_uc, size_t bytes, const std::string& endpoint);
00031
00032 void print_read_trace_msg(unsigned char *message_uc, size_t bytes, const std::string& endpoint);
00033
00034 #endif //ESTTS_HELPER_H
```

Index