AS7 On-Board Software

Generated by Doxygen 1.9.5

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Namespace Documentation

4.1 AS7 Namespace Reference

Classes

- class Drone
- struct DroneCommand
- class Logger

Enumerations

- enum DroneCommandType { Blind , Guided , Landing , Arm }

4.1.1 Enumeration Type Documentation

4.1.1.1 DroneCommandType

`enum AS7::DroneCommandType`

Enumerator

Blind	
Guided	
Landing	
Arm	

Chapter 5

Class Documentation

5.1 AS7::Drone Class Reference

`#include <Drone.h>`

Public Member Functions

- Drone (Logger *logger, bfs::SbusRx *sbus_rx, bfs::SbusTx *sbus_tx)
- bool channelConfirm (int16_t channel=1, float threshold=0.7f)
- int droneStatus ()
- bool getEnableOperatorControl ()
- bool getEnableEmergencyStop ()
- bool getDroneArmComplete ()
- float getUsFront ()
- float getUsBack ()
- float getUsLeft ()
- float getUsRight ()
- float getUsUp ()
- float getUsDown ()
- float getCompassHeading ()
- void start (int core=1, int priority=3)
- void pause ()
- void resume ()
- void enqueueCommand (DroneCommand cmd)
- DroneCommand dequeueCommand ()
- void allowArming ()
- bool droneAllowedToFly () const
- bool droneHasArmed () const
- void enableOperatorControl ()
- void disableOperatorControl ()
- void emergencyStop ()
- void resetEmergencyStop ()
- void setDataGathering (bool value)
- bool recordingEnabled ()
- void generateEStopTx ()
- std::array< int16_t, NUM_CH > getEStopTx ()
- std::string getSbusRxArray ()
- std::string getSbusTxArray ()

Public Attributes

- float usFront = 200
- float usBack = 200
- float usLeft = 200
- float usRight = 200
- float usUp = 200
- float usDown = 200
- float compassHeading
- bool _dataGatheringEnabled = false

5.1.1 Constructor & Destructor Documentation

5.1.1.1 Drone()

```
AS7::Drone::Drone (
            Logger * logger,
            bfs::SbusRx * sbus_rx,
            bfs::SbusTx * sbus_tx )
```

5.1.2 Member Function Documentation

5.1.2.1 allowArming()

```
void AS7::Drone::allowArming ( )  [inline]
```

5.1.2.2 channelConfirm()

```
bool AS7::Drone::channelConfirm (
            int16_t channel = 1,
            float threshold = 0.7f )
```

5.1.2.3 dequeueCommand()

```
DroneCommand AS7::Drone::dequeueCommand ( )
```

5.1.2.4 disableOperatorControl()

```
void AS7::Drone::disableOperatorControl ( )
```

5.1.2.5 droneAllowedToFly()

```
bool AS7::Drone::droneAllowedToFly ( ) const  [inline]
```

5.1.2.6 droneHasArmed()

```
bool AS7::Drone::droneHasArmed ( ) const  [inline]
```

5.1.2.7 droneStatus()

```
int AS7::Drone::droneStatus ( )
```

5.1.2.8 emergencyStop()

```
void AS7::Drone::emergencyStop ( )
```

5.1.2.9 enableOperatorControl()

```
void AS7::Drone::enableOperatorControl ( )
```

5.1.2.10 enqueueCommand()

```
void AS7::Drone::enqueueCommand (
            DroneCommand cmd )
```

5.1.2.11 generateEStopTx()

```
void AS7::Drone::generateEStopTx ( )
```

5.1.2.12 getCompassHeading()

```
float AS7::Drone::getCompassHeading ( )  [inline]
```

5.1.2.13 getDroneArmComplete()

```
bool AS7::Drone::getDroneArmComplete ( )  [inline]
```

5.1.2.14 getEnableEmergencyStop()

```
bool AS7::Drone::getEnableEmergencyStop ( )  [inline]
```

5.1.2.15 getEnableOperatorControl()

```
bool AS7::Drone::getEnableOperatorControl ( )  [inline]
```

5.1.2.16 getEStopTx()

```
std::array< int16_t, NUM_CH > AS7::Drone::getEStopTx ( )
```

5.1.2.17 getSbusRxArray()

```
std::string AS7::Drone::getSbusRxArray ( )
```

5.1.2.18 getSbusTxArray()

```
std::string AS7::Drone::getSbusTxArray ( )
```

5.1.2.19 getUsBack()

```
float AS7::Drone::getUsBack ( )  [inline]
```

5.1.2.20 getUsDown()

```
float AS7::Drone::getUsDown ( )  [inline]
```

5.1.2.21 getUsFront()

```
float AS7::Drone::getUsFront ( )  [inline]
```

5.1.2.22 getUsLeft()

```
float AS7::Drone::getUsLeft ( )  [inline]
```

5.1.2.23 getUsRight()

```
float AS7::Drone::getUsRight ( )  [inline]
```

5.1.2.24 getUsUp()

```
float AS7::Drone::getUsUp ( )  [inline]
```

5.1.2.25 pause()

```
void AS7::Drone::pause ( )
```

5.1.2.26 recordingEnabled()

```
bool AS7::Drone::recordingEnabled ( )  [inline]
```

5.1.2.27 resetEmergencyStop()

```
void AS7::Drone::resetEmergencyStop ( )
```

5.1.2.28 resume()

```
void AS7::Drone::resume ( )
```

5.1.2.29 setDataGathering()

```
void AS7::Drone::setDataGathering (
            bool value )  [inline]
```

5.1.2.30 start()

```
void AS7::Drone::start (
            int core = 1,
            int priority = 3 )
```

5.1.3 Member Data Documentation

5.1.3.1 _dataGatheringEnabled

```
bool AS7::Drone::_dataGatheringEnabled = false
```

5.1.3.2 compassHeading

```
float AS7::Drone::compassHeading
```

5.1.3.3 usBack

```
float AS7::Drone::usBack = 200
```

5.1.3.4 usDown

```
float AS7::Drone::usDown = 200
```

5.1.3.5 usFront

```
float AS7::Drone::usFront = 200
```

5.1.3.6 usLeft

```
float AS7::Drone::usLeft = 200
```

5.1.3.7 usRight

```
float AS7::Drone::usRight = 200
```

5.1.3.8 usUp

```
float AS7::Drone::usUp = 200
```

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

- H:/repos/Argous/src/esp32/as7-gamma/Drone.h
- H:/repos/Argous/src/esp32/as7-gamma/Drone.cpp

5.2 AS7::DroneCommand Struct Reference

```
#include <Drone.h>
```

Public Attributes

- DroneCommandType type
- std::string desc
- std::array< float, NUM_CH > channels
- int duration
- float rateMultiplier = 1.0f
- float p_x
- float p_y
- float p_z
- float v_x
- float v_y
- float v_z
- float p_rl
- float p_pt
- float p_yw
- float v_rl
- float v_pt
- float v_yw
- bool dataRecording

5.2.1 Member Data Documentation

5.2.1.1 channels

```
std::array<float, NUM_CH> AS7::DroneCommand::channels
```

5.2.1.2 dataRecording

```
bool AS7::DroneCommand::dataRecording
```

5.2.1.3 desc

```
std::string AS7::DroneCommand::desc
```

5.2.1.4 duration

```
int AS7::DroneCommand::duration
```

5.2.1.5 p_pt

```
float AS7::DroneCommand::p_pt
```

5.2.1.6 p_rl

```
float AS7::DroneCommand::p_rl
```

5.2.1.7 p_x

```
float AS7::DroneCommand::p_x
```

5.2.1.8 p_y

```
float AS7::DroneCommand::p_y
```

5.2.1.9 p_yw

```
float AS7::DroneCommand::p_yw
```

5.2.1.10 p_z

```
float AS7::DroneCommand::p_z
```

5.2.1.11 rateMultiplier

```
float AS7::DroneCommand::rateMultiplier = 1.0f
```

5.2.1.12 type

```
DroneCommandType AS7::DroneCommand::type
```

5.2.1.13 v_pt

```
float AS7::DroneCommand::v_pt
```

5.2.1.14 v_rl

```
float AS7::DroneCommand::v_rl
```

5.2.1.15 v_x

```
float AS7::DroneCommand::v_x
```

5.2.1.16 v_y

```
float AS7::DroneCommand::v_y
```

5.2.1.17 v_yw

```
float AS7::DroneCommand::v_yw
```

5.2.1.18 v_z

```
float AS7::DroneCommand::v_z
```

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

- H:/repos/Argous/src/esp32/as7-gamma/Drone.h

5.3 AS7::Logger Class Reference

```
#include <SdLogger.h>
```

Public Member Functions

- Logger (Print *output)
- void start (int core=1, int priority=1, int verbosity=LOG_LEVEL_INFORM)
- void pause ()
- void resume ()
- bool running ()
- int verbosity ()
- void setVerbosity (int verbosity)
- void recordData (std::string key, float value)
- void pushData ()
- void inform (std::string message)
- void warn (std::string message)
- void error (std::string message)
- void fatal (std::string message)
- void verbose (std::string message)
- void plot (std::string message)
- void disableSDLogging ()

5.3.1 Constructor & Destructor Documentation

5.3.1.1 Logger()

```
AS7::Logger::Logger (
            Print * output )
```

5.3.2 Member Function Documentation

5.3.2.1 disableSDLogging()

```
void AS7::Logger::disableSDLogging ( )
```

5.3.2.2 error()

```
void AS7::Logger::error (
            std::string message )
```

5.3.2.3 fatal()

```
void AS7::Logger::fatal (
            std::string message )
```

5.3.2.4 inform()

```
void AS7::Logger::inform (
            std::string message )
```

5.3.2.5 pause()

```
void AS7::Logger::pause ( )
```

5.3.2.6 plot()

```
void AS7::Logger::plot (
            std::string message )
```

5.3.2.7 pushData()

```
void AS7::Logger::pushData ( )
```

5.3.2.8 recordData()

```
void AS7::Logger::recordData (
            std::string key,
            float value )
```

5.3.2.9 resume()

```
void AS7::Logger::resume ( )
```

5.3.2.10 running()

```
bool AS7::Logger::running ( )
```

5.3.2.11 setVerbosity()

```
void AS7::Logger::setVerbosity (
            int verbosity )
```

5.3.2.12 start()

```
void AS7::Logger::start (
            int core = 1,
            int priority = 1,
            int verbosity = LOG_LEVEL_INFORM )
```

5.3.2.13 verbose()

```
void AS7::Logger::verbose (
            std::string message )
```

5.3.2.14 verbosity()

```
int AS7::Logger::verbosity ( )
```

5.3.2.15 warn()

```
void AS7::Logger::warn (
            std::string message )
```

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

- H:/repos/Argous/src/esp32/as7-gamma/SdLogger.h
- H:/repos/Argous/src/esp32/as7-gamma/SdLogger.cpp

Chapter 6

File Documentation

6.1 H:/repos/Argous/src/esp32/as7-gamma/as7-gamma.ino File Reference

```
#include <sbus.h>
#include <FastLED.h>
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include <Adafruit_MMC5883.h>
#include <map>
#include "SdLogger.h"
#include "Drone.h"
```

Macros

- #define I2C_SDA 21
- #define I2C_SCL 22
- #define SIMULATION_ENABLE false
- #define LED_TYPE WS2811
- #define COLOR_ORDER GRB
- #define GRAVITY_CMS 980.6
- #define ACC_FREQ 200
- #define LED_TYPE WS2811
- #define COLOR_ORDER GRB

Enumerations

- enum DroneState {
  Initialise , Ready , Armed , Flying ,
  Landing , Stopped , Faulted , Debug }
- enum DroneFlightMode { OperatorControl , AutoStraightLine , ArmOnly }
- enum DroneDebugTest { SBUS_COMMS , ARMING_DISARMING , SD_READ_WRITE , LED_RESPONSE
  }

Functions

- DEFINE_GRADIENT_PALETTE (DEV_PALLETE)
- void readAccelerometer ()
- void taskUltrasonicSensor (void *parameters)
- void taskAccelerometer (void *parameters)
- void debug_switchModes (void *parameters)
- void taskStatusLeds (void *parameters)
- void setup ()
- void loop ()

Variables

- const int STATUS_FASTLED_PIN = 3
- const int STATUS_NUM_LEDS = 8
- const int STATUS_LED_BRIGHTNESS = 32
- const int NUM_US_SENSORS = 6
- const int NUM_US_POINTS = 3
- const int MAX_US_DISTANCE = 2000
- const int US_TRIGPIN_0 = 32
- const int US_ECHOPIN_0 = 35
- const int US_TRIGPIN_1 = 25
- const int US_ECHOPIN_1 = 33
- const int US_TRIGPIN_2 = 12
- const int US_ECHOPIN_2 = 13
- const int US_TRIGPIN_3 = 14
- const int US_ECHOPIN_3 = 27
- const int US_TRIGPIN_4 = 0
- const int US_ECHOPIN_4 = 4
- const int US_TRIGPIN_5 = 2
- const int US_ECHOPIN_5 = 15
- int US_TRIGPIN [NUM_US_SENSORS] = {}
- int US_ECHOPIN [NUM_US_SENSORS] = {}
- const int FL_LEDPIN = 3
- const int SBUS_RXPIN = 16
- const int SBUS_TXPIN = 17
- const int TEMP_PIN = 34
- const TickType_t TICK_US_DELAY = 80 / portTICK_PERIOD_MS
- const TickType_t TICK_SHORT = 200 / portTICK_PERIOD_MS
- const TickType_t TICK_MEDIUM = 500 / portTICK_PERIOD_MS
- const TickType_t TICK_LONG = 1000 / portTICK_PERIOD_MS
- const TickType_t TICK_LONGLONG = 5000 / portTICK_PERIOD_MS
- const int FL_LEDNUM = 8
- const int FL_LEDBRIGHTNESS = 64
- CRGB FL_LED [FL_LEDNUM]
- CRGBPalette16 DEVPAL = DEV_PALLETE
- int8_t accel_x_int
- int8_t accel_y_int
- int8_t accel_z_int
- float accel_total_x
- float accel_total_y
- float accel_total_z
- const float DIV_I =16
- byte accelData [3]

- const int NUM_ACCEL_READINGS = 5
- unsigned long accel_prev_millis = 0
- unsigned long accel_curr_millis = 0
- float accel_delta_s = 0
- float accel_x_readings [NUM_ACCEL_READINGS]
- float accel_y_readings [NUM_ACCEL_READINGS]
- float accel_z_readings [NUM_ACCEL_READINGS]
- int accel_index = 0
- float accel_x
- float accel_y
- float accel_z
- float accel_filt_x
- float accel_filt_y
- float accel_filt_z
- float raw_vel_x = 0
- float raw_vel_y = 0
- float raw_vel_z = 0
- float filt_vel_x = 0
- float filt_vel_y = 0
- float filt_vel_z = 0
- float raw_pos_x = 0
- float raw_pos_y = 0
- float raw_pos_z = 0
- float filt_pos_x = 0
- float filt_pos_y = 0
- float filt_pos_z = 0
- float compass_x
- float compass_y
- float compass_z
- float compass_heading
- const float declinationAngle = 0.192228625
- float accel_x_offset = 0
- float accel_y_offset = 0
- float accel_z_offset = 0
- bool accelerometerCalibrated = false
- float estDronePos_x
- float estDronePos_y
- float estDronePos_z
- sensors_event_t compass_event
- bfs::SbusRx sbusRx & Serial1
- std::array< int16_t, NUM_CH > sbusTxData
- std::array< int16_t, NUM_CH > sbusRxData
- File config_file
- File output_file
- File log_file
- bool recordingEnabled = false
- int us_distance [NUM_US_SENSORS][NUM_US_POINTS]
- int us_rawDistance [NUM_US_SENSORS]
- int tmp_temperature
- SemaphoreHandle_t enable_usSemaphore
- SemaphoreHandle_t enable_pilotSemaphore
- SemaphoreHandle_t debug_switchModesSemaphore
- SemaphoreHandle_t enable_scribeSemaphore
- SemaphoreHandle_t scribe_logSemaphore
- SemaphoreHandle_t scribe_msgSemaphore

- SemaphoreHandle_t scribe_logStackMutex
- SemaphoreHandle_t scribe_msgStackMutex
- TaskHandle_t th_Ultrasonic
- TaskHandle_t th_Comms
- TaskHandle_t th_Accel
- TaskHandle_t th_Switch
- std::map< DroneState, std::string > droneStateMap
- DroneState currentState = Initialise
- DroneState nextState = Initialise
- DroneFlightMode currentFlightMode = AutoStraightLine
- AS7::Logger logger & Serial
- AS7::Drone drone & logger
- Adafruit_MMC5883 compass = Adafruit_MMC5883(11111)

6.1.1 Macro Definition Documentation

6.1.1.1 ACC_FREQ

```
#define ACC_FREQ 200
```

6.1.1.2 COLOR_ORDER [1/2]

```
#define COLOR_ORDER GRB
```

6.1.1.3 COLOR_ORDER [2/2]

```
#define COLOR_ORDER GRB
```

6.1.1.4 GRAVITY_CMS

```
#define GRAVITY_CMS 980.6
```

6.1.1.5 I2C_SCL

```
#define I2C_SCL 22
```

6.1.1.6 I2C_SDA

```
#define I2C_SDA 21
```

6.1.1.7 LED_TYPE [1/2]

```
#define LED_TYPE WS2811
```

6.1.1.8 LED_TYPE [2/2]

```
#define LED_TYPE WS2811
```

6.1.1.9 SIMULATION_ENABLE

```
#define SIMULATION_ENABLE false
```

6.1.2 Enumeration Type Documentation

6.1.2.1 DroneDebugTest

```
enum DroneDebugTest
```

Enumerator

SBUS_COMMS	
ARMING_DISARMING	
SD_READ_WRITE	
LED_RESPONSE	

6.1.2.2 DroneFlightMode

```
enum DroneFlightMode
```

Enumerator

OperatorControl	
AutoStraightLine	
ArmOnly	

6.1.2.3 DroneState

```
enum DroneState
```

Enumerator

Initialise	
Ready	
Armed	
Flying	
Landing	
Stopped	
Faulted	
Debug	

6.1.3 Function Documentation

6.1.3.1 debug_switchModes()

```
void debug_switchModes (
            void * parameters )
```

6.1.3.2 DEFINE_GRADIENT_PALETTE()

```
DEFINE_GRADIENT_PALETTE (
            DEV_PALLETE  )
```

6.1.3.3 loop()

```
void loop ( )
```

6.1.3.4 readAccelerometer()

```
void readAccelerometer ( )
```

6.1.3.5 setup()

```
void setup ( )
```

6.1.3.6 taskAccelerometer()

```
void taskAccelerometer (
            void * parameters )
```

6.1.3.7 taskStatusLeds()

```
void taskStatusLeds (
            void * parameters )
```

6.1.3.8 taskUltrasonicSensor()

```
void taskUltrasonicSensor (
            void * parameters )
```

6.1.4 Variable Documentation

6.1.4.1 accel_curr_millis

```
unsigned long accel_curr_millis = 0
```

6.1.4.2 accel_delta_s

```
float accel_delta_s = 0
```

6.1.4.3 accel_filt_x

```
float accel_filt_x
```

6.1.4.4 accel_filt_y

```
float accel_filt_y
```

6.1.4.5 accel_filt_z

```
float accel_filt_z
```

6.1.4.6 accel_index

```
int accel_index = 0
```

6.1.4.7 accel_prev_millis

```
unsigned long accel_prev_millis = 0
```

6.1.4.8 accel_total_x

```
float accel_total_x
```

6.1.4.9 accel_total_y

```
float accel_total_y
```

6.1.4.10 accel_total_z

```
float accel_total_z
```

6.1.4.11 accel_x

```
float accel_x
```

6.1.4.12 accel_x_int

```
int8_t accel_x_int
```

6.1.4.13 accel_x_offset

```
float accel_x_offset = 0
```

6.1.4.14 accel_x_readings

```
float accel_x_readings[NUM_ACCEL_READINGS]
```

6.1.4.15 accel_y

```
float accel_y
```

6.1.4.16 accel_y_int

```
int8_t accel_y_int
```

6.1.4.17 accel_y_offset

```
float accel_y_offset = 0
```

6.1.4.18 accel_y_readings

```
float accel_y_readings[NUM_ACCEL_READINGS]
```

6.1.4.19 accel_z

```
float accel_z
```

6.1.4.20 accel_z_int

```
int8_t accel_z_int
```

6.1.4.21 accel_z_offset

```
float accel_z_offset = 0
```

6.1.4.22 accel_z_readings

```
float accel_z_readings[NUM_ACCEL_READINGS]
```

6.1.4.23 accelData

```
byte accelData[3]
```

6.1.4.24 accelerometerCalibrated

```
bool accelerometerCalibrated = false
```

6.1.4.25 compass

```
Adafruit_MMC5883 compass = Adafruit_MMC5883(11111)
```

6.1.4.26 compass_event

```
sensors_event_t compass_event
```

6.1.4.27 compass_heading

```
float compass_heading
```

6.1.4.28 compass_x

```
float compass_x
```

6.1.4.29 compass_y

```
float compass_y
```

6.1.4.30 compass_z

```
float compass_z
```

6.1.4.31 config_file

```
File config_file
```

6.1.4.32 currentFlightMode

```
DroneFlightMode currentFlightMode = AutoStraightLine
```

6.1.4.33 currentState

```
DroneState currentState = Initialise
```

6.1.4.34 debug_switchModesSemaphore

```
SemaphoreHandle_t debug_switchModesSemaphore
```

6.1.4.35 declinationAngle

```
const float declinationAngle = 0.192228625
```

6.1.4.36 DEVPAL

```
CRGBPalette16 DEVPAL = DEV_PALLETE
```

6.1.4.37 DIV_I

```
const float DIV_I =16
```

6.1.4.38 droneStateMap

```
std::map<DroneState, std::string> droneStateMap
```

Initial value:

```
= {
  {Initialise, "Initialise"},
  {Ready, "Ready"},
  {Armed, "Armed"},
  {Flying, "Flying"},
  {Landing, "Landing"},
  {Stopped, "Stopped"},
  {Faulted, "Faulted"},
  {Debug, "Debug"}
 }
```

6.1.4.39 enable_pilotSemaphore

```
SemaphoreHandle_t enable_pilotSemaphore
```

6.1.4.40 enable_scribeSemaphore

```
SemaphoreHandle_t enable_scribeSemaphore
```

6.1.4.41 enable_usSemaphore

```
SemaphoreHandle_t enable_usSemaphore
```

6.1.4.42 estDronePos_x

```
float estDronePos_x
```

6.1.4.43 estDronePos_y

```
float estDronePos_y
```

6.1.4.44 estDronePos_z

```
float estDronePos_z
```

6.1.4.45 filt_pos_x

```
float filt_pos_x = 0
```

6.1.4.46 filt_pos_y

```
float filt_pos_y = 0
```

6.1.4.47 filt_pos_z

```
float filt_pos_z = 0
```

6.1.4.48 filt_vel_x

```
float filt_vel_x = 0
```

6.1.4.49 filt_vel_y

```
float filt_vel_y = 0
```

6.1.4.50 filt_vel_z

```
float filt_vel_z = 0
```

6.1.4.51 FL_LED

```
CRGB FL_LED[FL_LEDNUM]
```

6.1.4.52 FL_LEDBRIGHTNESS

```
const int FL_LEDBRIGHTNESS = 64
```

6.1.4.53 FL_LEDNUM

```
const int FL_LEDNUM = 8
```

6.1.4.54 FL_LEDPIN

```
const int FL_LEDPIN = 3
```

6.1.4.55 log_file

```
File log_file
```

6.1.4.56 logger

```
AS7::Drone drone& logger
```

6.1.4.57 MAX_US_DISTANCE

```
const int MAX_US_DISTANCE = 2000
```

6.1.4.58 nextState

```
DroneState nextState = Initialise
```

6.1.4.59 NUM_ACCEL_READINGS

```
const int NUM_ACCEL_READINGS = 5
```

6.1.4.60 NUM_US_POINTS

```
const int NUM_US_POINTS = 3
```

6.1.4.61 NUM_US_SENSORS

```
const int NUM_US_SENSORS = 6
```

6.1.4.62 output_file

```
File output_file
```

6.1.4.63 raw_pos_x

```
float raw_pos_x = 0
```

6.1.4.64 raw_pos_y

```
float raw_pos_y = 0
```

6.1.4.65 raw_pos_z

```
float raw_pos_z = 0
```

6.1.4.66 raw_vel_x

```
float raw_vel_x = 0
```

6.1.4.67 raw_vel_y

```
float raw_vel_y = 0
```

6.1.4.68 raw_vel_z

```
float raw_vel_z = 0
```

6.1.4.69 recordingEnabled

```
bool recordingEnabled = false
```

6.1.4.70 SBUS_RXPIN

```
const int SBUS_RXPIN = 16
```

6.1.4.71 SBUS_TXPIN

```
const int SBUS_TXPIN = 17
```

6.1.4.72 sbusRxData

```
std::array<int16_t, NUM_CH> sbusRxData
```

6.1.4.73 sbusTxData

```
std::array<int16_t, NUM_CH> sbusTxData
```

6.1.4.74 scribe_logSemaphore

```
SemaphoreHandle_t scribe_logSemaphore
```

6.1.4.75 scribe_logStackMutex

```
SemaphoreHandle_t scribe_logStackMutex
```

6.1.4.76 scribe_msgSemaphore

```
SemaphoreHandle_t scribe_msgSemaphore
```

6.1.4.77 scribe_msgStackMutex

```
SemaphoreHandle_t scribe_msgStackMutex
```

6.1.4.78 Serial

```
AS7::Logger logger& Serial
```

6.1.4.79 Serial1

```
bfs::SbusTx sbusTx & Serial1
```

6.1.4.80 STATUS_FASTLED_PIN

```
const int STATUS_FASTLED_PIN = 3
```

6.1.4.81 STATUS_LED_BRIGHTNESS

```
const int STATUS_LED_BRIGHTNESS = 32
```

6.1.4.82 STATUS_NUM_LEDS

```
const int STATUS_NUM_LEDS = 8
```

6.1.4.83 TEMP_PIN

```
const int TEMP_PIN = 34
```

6.1.4.84 th_Accel

```
TaskHandle_t th_Accel
```

6.1.4.85 th_Comms

```
TaskHandle_t th_Comms
```

6.1.4.86 th_Switch

```
TaskHandle_t th_Switch
```

6.1.4.87 th_Ultrasonic

```
TaskHandle_t th_Ultrasonic
```

6.1.4.88 TICK_LONG

```
const TickType_t TICK_LONG = 1000 / portTICK_PERIOD_MS
```

6.1.4.89 TICK_LONGLONG

```
const TickType_t TICK_LONGLONG = 5000 / portTICK_PERIOD_MS
```

6.1.4.90 TICK_MEDIUM

```
const TickType_t TICK_MEDIUM = 500 / portTICK_PERIOD_MS
```

6.1.4.91 TICK_SHORT

```
const TickType_t TICK_SHORT = 200 / portTICK_PERIOD_MS
```

6.1.4.92 TICK_US_DELAY

```
const TickType_t TICK_US_DELAY = 80 / portTICK_PERIOD_MS
```

6.1.4.93 tmp_temperature

```
int tmp_temperature
```

6.1.4.94 us_distance

```
int us_distance[NUM_US_SENSORS][NUM_US_POINTS]
```

6.1.4.95 US_ECHOPIN

```
int US_ECHOPIN[NUM_US_SENSORS] = {}
```

6.1.4.96 US_ECHOPIN_0

```
const int US_ECHOPIN_0 = 35
```

6.1.4.97 US_ECHOPIN_1

```
const int US_ECHOPIN_1 = 33
```

6.1.4.98 US_ECHOPIN_2

```
const int US_ECHOPIN_2 = 13
```

6.1.4.99 US_ECHOPIN_3

```
const int US_ECHOPIN_3 = 27
```

6.1.4.100 US_ECHOPIN_4

```
const int US_ECHOPIN_4 = 4
```

6.1.4.101 US_ECHOPIN_5

```
const int US_ECHOPIN_5 = 15
```

6.1.4.102 us_rawDistance

```
int us_rawDistance[NUM_US_SENSORS]
```

6.1.4.103 US_TRIGPIN

```
int US_TRIGPIN[NUM_US_SENSORS] = {}
```

6.1.4.104 US_TRIGPIN_0

```
const int US_TRIGPIN_0 = 32
```

6.1.4.105 US_TRIGPIN_1

```
const int US_TRIGPIN_1 = 25
```

6.1.4.106 US_TRIGPIN_2

```
const int US_TRIGPIN_2 = 12
```

6.1.4.107 US_TRIGPIN_3

```
const int US_TRIGPIN_3 = 14
```

6.1.4.108 US_TRIGPIN_4

```
const int US_TRIGPIN_4 = 0
```

6.1.4.109 US_TRIGPIN_5

```
const int US_TRIGPIN_5 = 2
```

6.2 H:/repos/Argous/src/esp32/as7-gamma/Drone.cpp File Reference

```
#include "Drone.h"
```

Namespaces

- namespace AS7

6.3 H:/repos/Argous/src/esp32/as7-gamma/Drone.h File Reference

```
#include <Arduino.h>
#include <sbus.h>
#include <vector>
#include "SdLogger.h"
```

Classes

- struct AS7::DroneCommand
- class AS7::Drone

Namespaces

- namespace AS7

Macros

- #define AS7DRONE
- #define SBUS_CHANNEL_LOWER 0
- #define SBUS_CHANNEL_UPPER 2056
- #define NUM_CH 16
- #define DOF 6
- #define NAV_FREQ 250
- #define CTL_FREQ 1000
- #define CH_THROTTLE 2
- #define CH_YAW 3
- #define CH_STRAIGHT 1
- #define CH_STRAFE 0
- #define CH_SW1 7
- #define CH_FLIGHTMODE 4
- #define CH_ESTOP 6
- #define STATUS_UPDATE_DELAY 250
- #define THROTTLE_LIMIT 0.5f
- #define RAMPRATE_NONE 0
- #define RAMPRATE_LINEAR 1
- #define RAMPRATE_PROP 2

Enumerations

- enum AS7::DroneCommandType { AS7::Blind , AS7::Guided , AS7::Landing , AS7::Arm }

6.3.1 Macro Definition Documentation

6.3.1.1 AS7DRONE

```
#define AS7DRONE
```

TODO: THIS BANNER

AS7 Drone This class encapsulates talking to AS7

The main program controls the state that AS7 is in This class represents the inputs/outputs for AS7

6.3.1.2 CH_ESTOP

```
#define CH_ESTOP 6
```

6.3.1.3 CH_FLIGHTMODE

```
#define CH_FLIGHTMODE 4
```

6.3.1.4 CH_STRAFE

```
#define CH_STRAFE 0
```

6.3.1.5 CH_STRAIGHT

```
#define CH_STRAIGHT 1
```

6.3.1.6 CH_SW1

```
#define CH_SW1 7
```

6.3.1.7 CH_THROTTLE

```
#define CH_THROTTLE 2
```

6.3.1.8 CH_YAW

```
#define CH_YAW 3
```

6.3.1.9 CTL_FREQ

```
#define CTL_FREQ 1000
```

6.3.1.10 DOF

```
#define DOF 6
```

6.3.1.11 NAV_FREQ

```
#define NAV_FREQ 250
```

6.3.1.12 NUM_CH

```
#define NUM_CH 16
```

6.3.1.13 RAMPRATE_LINEAR

```
#define RAMPRATE_LINEAR 1
```

6.3.1.14 RAMPRATE_NONE

```
#define RAMPRATE_NONE 0
```

6.3.1.15 RAMPRATE_PROP

```
#define RAMPRATE_PROP 2
```

6.3.1.16 SBUS_CHANNEL_LOWER

```
#define SBUS_CHANNEL_LOWER 0
```

6.3.1.17 SBUS_CHANNEL_UPPER

```
#define SBUS_CHANNEL_UPPER 2056
```

6.3.1.18 STATUS_UPDATE_DELAY

```
#define STATUS_UPDATE_DELAY 250
```

6.3.1.19 THROTTLE_LIMIT

```
#define THROTTLE_LIMIT 0.5f
```

6.4 Drone.h

Go to the documentation of this file.

```
1
14 #pragma once
15 #ifndef AS7DRONE
16 #define AS7DRONE
17
18 #include <Arduino.h>
19 #include <sbus.h>     // SBUS Communication Library with FC
20 #include <vector>
21
22 #include "SdLogger.h" // Use the SD Logger to share messages
23
24 #define SBUS_CHANNEL_LOWER 0        // Default lower bound for sbus channels
25 #define SBUS_CHANNEL_UPPER 2056     // Default upper bound for sbus channels
26 #define NUM_CH 16                   // Number of SBUS channels.  Always 16.  Equivalent to
       bfs::SbusRx::NUM_CH()
27 #define DOF 6                       // Degrees of freedom for the drone.  0-5 represent x, y, z, roll
       (rl), pitch (pt), yaw (yw) (Euler ZYX Convention)
28
29 // Thread update frequencies (to reduce starvation of the watchdog)
30 //  Feed your watchdogs, people!  They do important household chores!
31 #define NAV_FREQ 250    // Navigation update rate, Hz (Default:  250 Hz)
32 #define CTL_FREQ 1000   // Controller update rate, Hz (Default:  250 Hz)
33
34 // Channel definitions
35 //  These channels index from ZERO. Ch[0] = CH1!
36 #define CH_THROTTLE     2   // Left stick y axis (starts from 0)
37 #define CH_YAW          3   // Left stick x axis
38 #define CH_STRAIGHT     1   // Right stick y axis
39 #define CH_STRAFE       0   // Right stick x axis
40 //#define CH_BUTTON1      6   // Button on middle of controller
41 #define CH_SW1          7   // Right toggle switch
42 #define CH_FLIGHTMODE   4   // Left toggle switch
43 #define CH_ESTOP        6   // Other button?
44
45 #define STATUS_UPDATE_DELAY 250 // Number of updates to wait before sending controller status
46
47 #define THROTTLE_LIMIT 0.5f
48
49 // Cv = Control Value, Pv = Present Value.  Use CV to control PV
50 #define RAMPRATE_NONE       0   // No ramp rate.                    Pv = Cv
51 #define RAMPRATE_LINEAR     1   // Linear ramp rate,        Pv += Constant until Pv > Cv
52 #define RAMPRATE_PROP       2   // Proportional Ramp Rate   Pv = (Cv - Pv) * Constant
53
54 namespace AS7
55 {
56     enum DroneCommandType {Blind, Guided, Landing, Arm};
57
58     // Drone Command Structure/Format
59     //  Two types:  Blind and Guided, set by enum DroneCommandType
60     //  Blind commands only refer to the channel array and duration
61     //  Guided commands will attempt to use on-board sensors to control the drone
62     //
63     // Landing is a special type where the drone will lower thrust just below its known weight to land
64     //  If possible, it will use the bottom sensors to guide landing
65     //  Landing is equivalent to setting v_y to some pre-defined value in blind mode.
66     //
67     // Arm is a special type that will send the arming command to the drone.
68     //  As SBUS is one-way at the moment, it's not possible to *know* if the drone is armed but we can
       generally assume
69     //   that after a certain duration, the drone is armed.
```

```
70      //  Because of this, there is no way to know if the drone is disarmed as it will disarm automatically
        from the FC
71      //  More advanced implementations using two-way SBUS or ideally MAVLink can get around this
        limmitation.
72      //
73      // Drone commands are enqueued to the drone, and will be executed FIFO.
74      //
75      // Note:  Blind Commands can also be used as buttons and inputs to the FC, not just for navigation
76      //
77      typedef struct {
78          DroneCommandType type;              // Blind = Purely a drone command, Guided = Assisted with
        sensors
79          std::string desc;                   // A description for the logger
80          std::array<float, NUM_CH> channels; // A float for each channel from (-1, 1)
81          int duration;                       // in ms
82          float rateMultiplier = 1.0f;          // Can be thought of as "aggressiveness" of controls
83
84          // Prefix P = Position; V = Velocity.  Units on per-member basis and are *convention* (not
        checked)
85          float p_x;         // Position to hold (some distance unit tbc)
86          float p_y;
87          float p_z;
88          float v_x;       // Velocity to hold (-1 to 1) floating point value
89          float v_y;       // e.g.  0.1 is equivalent to 10% forward thrust
90          float v_z;
91          float p_rl;        // Not directly controllable on a drone
92          float p_pt;        // Not directly controllable on a drone
93          float p_yw;
94          float v_rl;      // Not directly controllable on a drone
95          float v_pt;      // Not directly controllable on a drone
96          float v_yw;
97          bool dataRecording; // indicates if the drone should record data, used to synchronise sensors and
        current drone state
98      } DroneCommand;
99
100     class Drone
101     {
102     private:
103         TaskHandle_t thDrone;
104         TaskHandle_t thRemote;
105         static void startNavTask(void*);   // Task implementation for classes
106         static void startCtlTask(void*);
107         void navigationTask(void* parameters);    // The threaded task
108         void controllerTask(void* parameters);
109
110         std::queue<DroneCommand> _droneCommandQueue;
111
112         SemaphoreHandle_t _semDroneEnableMutex;          // Enables/Disables main drone task
113         SemaphoreHandle_t _semControlEnableMutex;        // Enables/Disables main drone task
114         SemaphoreHandle_t getSemDroneEnableMutex();      // Returns the enable mutex
115         SemaphoreHandle_t getSemControlEnableMutex();    // Returns the enable mutex
116
117         SemaphoreHandle_t _semTxChMutex;  // Mutex Lock for the tx data array
118         SemaphoreHandle_t getTxChMutex();   // Returns the write mutex for threading implementation
119
120         SemaphoreHandle_t _semRxChMutex;  // Mutex Lock for the rx data array
121         SemaphoreHandle_t getRxChMutex();   // Returns the write mutex for threading implementation
122
123         SemaphoreHandle_t _semCommandQueueMutex;    // Mutex lock for drone command queue
124         SemaphoreHandle_t getCommandQueueMutex();
125
126         Logger* _logger;
127         Logger* getLogger();
128
129         bfs::SbusRx* _sbusRx;   // SBUS Receive Channel Object
130         bfs::SbusTx* _sbusTx;   // SBUS Transmit Channel Object
131
132         std::array<int16_t, NUM_CH> _sbusRxData;    // Array of data received from the Radio Control
133         std::array<int16_t, NUM_CH> _sbusTxData;    // Array of data to transmit to the Flight
        Controller
134
135         std::array<bool, NUM_CH> _sbusAbsChannels;      // When true, the channel is (0, 1).  Otherwise
        channels default to (-1, 1).
136         std::array<bool, NUM_CH> getSbusAbsChannels();  // Returns an array which defines if a channel
        is absolute (true) or not (false).
137         void generateAbsChannels();                     // Sets the default Absolute Channels
138
139         int _controllerStatusCount = 0;
140         bool getControllerStatusCount();
141
142
143         // Channels that will be transmitted to the drone
144         std::array<int16_t, NUM_CH> _sbusTxChLower;    // Lower bounds for SBUS Transmit channels
145         std::array<int16_t, NUM_CH> _sbusTxChUpper;    // Upper bounds for SBUS Transmit Channels
146
147         // Channels that are received from the transmitter
148         std::array<int16_t, NUM_CH> _sbusRxChLower;    // Lower bounds for SBUS Receiver channels
```

```
149         std::array<int16_t, NUM_CH> _sbusRxChUpper;    // Upper bounds for SBUS Receiver Channels
150
151         std::array<int16_t, NUM_CH> _sbusEStopTx;      // Data to be written in case of an E-Stop.  Not
    all channels are to be zeroed!
152
153         // Methods for getting and setting data for task theads
154         inline std::array<int16_t, NUM_CH> getSbusRxData() {return _sbusRxData; }
155         inline std::array<int16_t, NUM_CH> getSbusTxData() {return _sbusTxData; }
156         void setSbusRxData(std::array<int16_t, NUM_CH> data);
157         void setSbusTxData(std::array<int16_t, NUM_CH> data);
158
159         void initUpperLowerBoundArrays();   // Sets UBound and LBound array to default
160
161         void writeChannel(int16_t value, int8_t ch);    // writes the value into the sbus transmit
    channel
162         void writeRxChannel_f(float value, int8_t ch);    // writes the value into the sbus transmit
    channel
163         void writeTxChannel_f(float value, int8_t ch);    // writes the value into the sbus transmit
    channel
164
165         int16_t convRxChannel_i(float value, int8_t ch);  // Returns the adjusted int16_t value for that
    channel
166         float convRxChannel_f(int16_t value, int8_t ch);  // Returns the adjusted int16_t value for that
    channel
167
168         int16_t convTxChannel_i(float value, int8_t ch);  // Returns the adjusted int16_t value for that
    channel
169         float convTxChannel_f(int16_t value, int8_t ch);  // Returns the adjusted int16_t value for that
    channel
170
171         int16_t readRxChannel(int16_t ch);              // Reads the value from the channel
172         float readRxChannel_f(int16_t ch);              // Reads the floating point value from the
    channel, adjusted for upper and lower bounds
173
174         int16_t readTxChannel(int16_t ch);              // Reads the current value being written to the
    controller
175         float readTxChannel_f(int16_t ch);              // Returns the current value as a floating point
    number
176
177         // Helper/Utility functions
178         float clamp(float value, float lbound, float ubound);                           // Returns
    values inside of upper bound and lower bound.
179         std::string formatSbusArray(std::array<int16_t, NUM_CH> chData);                // Returns the
    channels in a formatted string
180         float rampValue(float value, float target = 0, float rate = 0, int rampRateType =
    RAMPRATE_LINEAR);   // Returns the next ramped value depending on ramp type
181
182         // Combines rampValue with current channel data for ramping.
183         void rampChannel(float target, int8_t ch, float rate, int rampRateType = RAMPRATE_LINEAR);
184
185         bfs::SbusRx* getSbusRx();   // Returns SBUS RX object for task implementation
186         bfs::SbusTx* getSbusTx();   // Returns SBUS TX object for task implementation
187
188
189         bool _hasArmed = false;         // Remembers if the drone has undergone an arming process
190         bool _armingAllowed = false;    // Set by main program.  Once allowed, drone will start
    processing instructions
191         inline void setDroneHasArmed() {_hasArmed = true;}
192         inline bool getDroneHasArmed() {return _hasArmed; }
193
194         bool _droneCommandsStarted = false;     // Indicates if the drone has started processing
    commands
195         bool _droneCommandsCompleted = false;   // Indicates that there are no commands left (queue is
    empty)
196         bool _droneHasActiveCommand = false;
197         inline bool getHasActiveComamnd()const {return _droneHasActiveCommand;}
198         inline void setHasActiveCommand(bool value) {_droneHasActiveCommand = value;}
199         inline void setDroneCommandsStarted() {_droneCommandsStarted = true;}           // Latching
    check that the drone has started commands
200         inline void setDroneCommandsCompleted() {_droneCommandsCompleted = true;}       // Latching
    check that the rone has completed commands.  If commands haven't stareted, then this indicates empty.
201         inline bool nextCommandAvailable() {return _droneCommandQueue.size() != 0;}     // Checks if
    command queue size is not equal to zero.
202
203         bool _running = false;                  // Indicates current state of main drone task
204         bool _enableOperatorControl = false;    // When enabled, remote control commands are passed
    directly to drone from RX to TX
205         bool _enableEmergencyStop = false;      // When enabled, all TX channels are set to 0
206         bool _armingComplete = false;
207
208       public:
209
210         Drone(Logger *logger, bfs::SbusRx* sbus_rx, bfs::SbusTx* sbus_tx);
211         bool channelConfirm(int16_t channel=1, float threshold=0.7f);    // Returns true if the channel
    above threshold.  e.g.  button press
212
213         // Drone status is indicated by an int, though it could be indicated by an enum later on with
```

```
    DEFINEs
214         //  Current statuses could be drone_starting, drone_waitin0gdrone_in_progress, drone_estop,
    drone_operator_over, drone_finished
215         //  Not sure how the internal mechanism could work -- this could be a bunmch of bools with
    increasnig preference for noe another another?  maybe we shouldn't even consider status inside the
    drone *command* class
216         int droneStatus();
217 
218         inline bool getEnableOperatorControl() {return _enableOperatorControl; }        // Returns
    operator control bit
219         inline bool getEnableEmergencyStop() {return _enableEmergencyStop; }          // Returns e-stop
    bit
220         inline bool getDroneArmComplete() {return _armingComplete; }
221 
222         // Assume everything is far so we don't shock the drone into a position
223 
224         float usFront = 200;      // Distance measurement from the front US sensor
225         float usBack = 200;       // Distance measurement from the back US sensor
226         float usLeft = 200;       // Distance measurement from the left US sensor
227         float usRight = 200;      // Distance measurement from the right US sensor
228         float usUp = 200;         // Distance measurement from the upwards US sensor
229         float usDown = 200;       // Distance measurement from the downwards US sensor
230 
231         inline float getUsFront() {return usFront; };
232         inline float getUsBack() {return usBack; };
233         inline float getUsLeft() {return usLeft; };
234         inline float getUsRight() {return usRight; };
235         inline float getUsUp() {return usUp; };
236         inline float getUsDown() {return usDown; };
237 
238         float compassHeading;
239         inline float getCompassHeading() {return compassHeading; };
240 
241 
242 
243         // Main thread control
244         //  Operates both the controller and navigator threads
245         void start(int core=1, int priority=3);
246         void pause();
247         void resume();
248 
249         void enqueueCommand(DroneCommand cmd);  // Adds command to drone queue
250         DroneCommand dequeueCommand();          // Remove drone command, returns command from queue
251 
252         inline void allowArming() {_armingAllowed = true;}  // Allows drone to start processing commands
253         inline bool droneAllowedToFly()const {return _armingAllowed;} // Returns _armingAllowed bit
254         inline bool droneHasArmed()const {return _hasArmed;}          // Returns if dorn has armed
    previously
255 
256         void enableOperatorControl();   // Enables pass-through from RX to TX. Latching
257         void disableOperatorControl();
258 
259         void emergencyStop();
260         void resetEmergencyStop();
261 
262         bool _dataGatheringEnabled = false;
263         inline void setDataGathering(bool value) {_dataGatheringEnabled = value; }
264         inline bool recordingEnabled() {return _dataGatheringEnabled;} // Returns true if the current
    drone command is requesting data gathering
265 
266         void generateEStopTx();
267         std::array<int16_t, NUM_CH> getEStopTx();
268 
269         std::string getSbusRxArray();
270         std::string getSbusTxArray();
271     };
272 }
273 
274 
275 #endif
```

6.5 H:/repos/Argous/src/esp32/as7-gamma/SdLogger.cpp File Reference

```
#include "SdLogger.h"
```

Namespaces

- namespace AS7

6.6 H:/repos/Argous/src/esp32/as7-gamma/SdLogger.h File Reference

```
#include <Arduino.h>
#include <queue>
#include <SPI.h>
#include <SD.h>
#include <map>
```

Classes

- class AS7::Logger

Namespaces

- namespace AS7

Macros

- #define AS7SDLOGGING_H
- #define LOG_LEVEL_SILENT 0
- #define LOG_LEVEL_FATAL 1
- #define LOG_LEVEL_ERROR 2
- #define LOG_LEVEL_WARNING 3
- #define LOG_LEVEL_INFORM 4
- #define LOG_LEVEL_VERBOSE 5
- #define CS_PIN 5
- #define LOGGER_FREQ 100
- #define PLOTTER_ENABLE false
- #define SD_DISABLED false

6.6.1 Macro Definition Documentation

6.6.1.1 AS7SDLOGGING_H

```
#define AS7SDLOGGING_H
```

TODO: THIS BANNER

This is AS7's Logger. It prints to serial and SD card. The goal of this logger is to tie logging and comms into one location and delegate to a thread Read this for getters and setters inside freertos https://forums.↩ freertos.org/t/freertos-task-in-a-c-class/13984

6.6.1.2 CS_PIN

```
#define CS_PIN 5
```

6.6.1.3 LOG_LEVEL_ERROR

```
#define LOG_LEVEL_ERROR 2
```

6.6.1.4 LOG_LEVEL_FATAL

```
#define LOG_LEVEL_FATAL 1
```

6.6.1.5 LOG_LEVEL_INFORM

```
#define LOG_LEVEL_INFORM 4
```

6.6.1.6 LOG_LEVEL_SILENT

```
#define LOG_LEVEL_SILENT 0
```

6.6.1.7 LOG_LEVEL_VERBOSE

```
#define LOG_LEVEL_VERBOSE 5
```

6.6.1.8 LOG_LEVEL_WARNING

```
#define LOG_LEVEL_WARNING 3
```

6.6.1.9 LOGGER_FREQ

```
#define LOGGER_FREQ 100
```

6.6.1.10 PLOTTER_ENABLE

```
#define PLOTTER_ENABLE false
```

6.6.1.11 SD_DISABLED

```
#define SD_DISABLED false
```

6.7 SdLogger.h

Go to the documentation of this file.

```
1
8 #pragma once
9 #ifndef AS7SDLOGGING_H
10 #define AS7SDLOGGING_H
11
12 #include <Arduino.h>
13 #include <queue>
14 #include <SPI.h>
15 #include <SD.h>
16
17 #include <map>
18
19 // Defines the maximum level of messages sent through the serial port
20 // e.g.  a level of WARNING (3) will only allow warnings, errors, and fatal issues to be sent to the
      serial port.
21 #define LOG_LEVEL_SILENT    0
22 #define LOG_LEVEL_FATAL     1
23 #define LOG_LEVEL_ERROR     2
24 #define LOG_LEVEL_WARNING   3
25 #define LOG_LEVEL_INFORM    4
26 #define LOG_LEVEL_VERBOSE   5
27 #define CS_PIN 5
28
29 #define LOGGER_FREQ 100      // Update rate in Hertz
30
31 #define PLOTTER_ENABLE false // only prints plots, for testing.
32 #define SD_DISABLED false
33
34 namespace AS7
35 {
36     class Logger
37     {
38     private:
39
40         std::queue<std::string> _msg_Queue;
41         std::queue<std::string> _log_Queue;
42         // Used for reading the message Queues for the scribe task
43         SemaphoreHandle_t _sem_log;
44         SemaphoreHandle_t _sem_msg;
45
46         // Mutex for the message Queues
47         SemaphoreHandle_t _sem_logQueueMutex;
48         SemaphoreHandle_t _sem_msgQueueMutex;
49
50         SemaphoreHandle_t _sem_dataMutex;
51         SemaphoreHandle_t _sem_dataEnqMutex;
52
53         // Sem for Enabling/Disabling Task
54         SemaphoreHandle_t _sem_enableMutex;
55         bool _running = false; // tracks if the thread is running or stopped
56         bool _sdEnabled = false;
57         bool _sdDetected = false;
58
59         std::map<std::string, float> _activeData;     // Data that is actively written to
60         std::map<std::string, float> _enqueuedData;   // Used as a buffer before being written
61         bool _hasEnqueuedData = false;
62
63         File _logFile;
64         File _dataFile;
65         File _configFile;
66
```

```
67          File getLogFile();
68          File getDataFile();
69          File getConfigFile();
70
71          void openLogFile();     // Opens Log file on SD in Append
72          void openDataFile();    // Opens Data file on SD in Append
73          void closeLogFile();    // Closes Log File
74          void closeDataFile();   // Closes Data File
75
76          const std::string _logFileLocation = "/as7.log"; // Location of the logging file, includes
        extension.  Use .c_str() to for SD library
77          const std::string _dataFileLocation = "/data.csv"; // Location of the data file, includes
        extension.  Use .c_str() to for SD library
78
79          Print* _printer;
80
81          Print* getPrinter();
82          int _verbosity = LOG_LEVEL_INFORM;
83
84          std::string getTestMessage();
85          std::queue<std::string> getMsgQueue();
86          std::queue<std::string> getLogQueue();
87
88          SemaphoreHandle_t getSemLog();
89          SemaphoreHandle_t getSemMsg();
90          SemaphoreHandle_t getSemLogQueueMutex();
91          SemaphoreHandle_t getSemMsgQueueMutex();
92
93          inline SemaphoreHandle_t getSemDataMutex() {return _sem_dataMutex; }
94          inline SemaphoreHandle_t getSemDataEnqMutex() {return _sem_dataEnqMutex; }
95
96          SemaphoreHandle_t getSemEnableMutex();
97
98
99          // Adds a message to be recorded to the SD card
100          // diagnotics probably?  will flush out later
101          void enqueueMsg(std::string message);
102
103          // Adds a message to be sent to the console and onto the SD card
104          void enqueueLog(std::string message, int verbosity);
105
106          // for PLY generation, will need to be flushed out.
107          // String since we can also send header information
108          void enqueuePnt(std::string points);
109
110          std::string dequeueLog();
111
112          // Implementation to start FreeRTOS tasks in classes
113          static void startTaskImpl(void*);
114          TaskHandle_t th_logger;
115          void mainTask(void* parameters);
116
117          void initialiseSD();
118
119      public:
120          Logger(Print* output);
121          void start(int core=1, int priority=1, int verbosity=LOG_LEVEL_INFORM);
122          void pause();
123          void resume();
124
125          bool running();
126          int  verbosity();
127
128          void setVerbosity(int verbosity);
129
130          void recordData(std::string key, float value);
131          void pushData();
132
133          // The main logging tasks
134          void inform(std::string message);
135          void warn(std::string message);
136          void error(std::string message);
137          void fatal(std::string message);
138          void verbose(std::string message);
139
140          void plot(std::string message);
141
142          void disableSDLogging();        // Disables SD Logging, even if SD Card is attached.
143
144      };
145  }
146
147
148
149  #endif
```

Index