Alpha CubeSat Ground

0.1

Generated by Doxygen 1.8.15

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all documented 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

Namespace Documentation

3.1 cubesat Namespace Reference

Namespaces

- binutil
- database
- telemetry

3.1.1 Detailed Description

Main namespace, containing ground station types

3.2 cubesat::binutil Namespace Reference

Functions

- int char_to_int (char input)
- void hex_to_bin (const char ∗src, unsigned char ∗target)
- std::vector< unsigned char > hex_str_to_bin (std::string const &encoded)

3.2.1 Detailed Description

Contains binary utilities

3.2.2 Function Documentation

3.2.2.1 char_to_int()

```
int cubesat::binutil::char_to_int (
             char input )  [inline]
```

Convert character to integer value

3.2.2.2 hex_str_to_bin()

```
std::vector<unsigned char> cubesat::binutil::hex_str_to_bin (
             std::string const & encoded )  [inline]
```

Convenience function - converts hex encoded string to byte array in a more C++-style way than the above methods

3.2.2.3 hex_to_bin()

```
void cubesat::binutil::hex_to_bin (
             const char * src,
             unsigned char * target )  [inline]
```

This function assumes src to be a zero terminated sanitized string with an even number of [0-9a-f] characters, and target to be sufficiently large

3.3 cubesat::database Namespace Reference

Classes

- class ElasticsearchDatabase
- class ImageDatabase

3.3.1 Detailed Description

Contains types and utilities for database management

3.4 cubesat::telemetry Namespace Reference

Classes

- class ImageAssembler
- class Packet
- struct RockBlockReport
- struct TelemetryConfig
- class TelemetryHandler

Typedefs

- using **HttpServer** = SimpleWeb::Server< SimpleWeb::HTTP >

Functions

- void write_rockblock_ok_response (HttpServer::Response &response)
- RockBlockReport parse_rockblock_request_params (HttpServer::Request &params)

3.4.1 Detailed Description

Contains types and utilities for handling satellite telemetry

3.4.2 Function Documentation

3.4.2.1 parse_rockblock_request_params()

```
RockBlockReport cubesat::telemetry::parse_rockblock_request_params (
            HttpServer::Request & params )
```

Reads a http parameter map `params` and returns a RockBlockReport object containing the API data

3.4.2.2 write_rockblock_ok_response()

```
void cubesat::telemetry::write_rockblock_ok_response (
            HttpServer::Response & response )
```

This function writes the required code 200 "OK" response required every time data is received from rockblock to `response`

Chapter 4

Class Documentation

4.1 cubesat::database::ElasticsearchDatabase Class Reference

```
#include <ElasticsearchDatabase.h>
```

Public Member Functions

- ElasticsearchDatabase (std::string const &pHost, int pPort)
- **ElasticsearchDatabase** (ElasticsearchDatabase const &other)=delete
- **ElasticsearchDatabase** (ElasticsearchDatabase const &&other)=delete
- void **operator=** (ElasticsearchDatabase const &other)=delete
- void **operator=** (ElasticsearchDatabase const &&other)=delete
- void index (std::string const &index_base_name, std::string const &document_type, boost::optional< uint64_t > const &doc_id, std::string const &data_json)

4.1.1 Detailed Description

An Elasticsearch database accessor used for submitting requests to the given database with json data. Automatically appends local date and time to index name to create an index pattern for sorting data. Note that this is not the actual timestamp information of any contained data.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 ElasticsearchDatabase()

```
cubesat::database::ElasticsearchDatabase::ElasticsearchDatabase (
            std::string const & pHost,
            int pPort )
```

Create a database representation that connects to elasticsearch using the provided ip and port

4.1.3 Member Function Documentation

4.1.3.1 index()

```
void cubesat::database::ElasticsearchDatabase::index (
            std::string const & index_base_name,
            std::string const & document_type,
            boost::optional< uint64_t > const & doc_id,
            std::string const & data_json )
```

Indexes a document synchronously, using `index_base_name` appended with local date for the index, `document_type` as the doc type, an optional document uid/serial number (if none is provided, elasticsearch generates one. This is the recommended approach), and json data as a string to be inserted into the database.

Example usage: A call to index("test_index", "test_type", {}, "{\"time" : "2019-03-16T11:45:50+02:00", "value" ↩ : 28.577}"); Will insert into the index test_index-<current time> a document of test_type, with json content: { "time" : "2019-03-16T11:45:50+02:0", "value" : 28.577 }

A note to future maintainers: document types will be deprecated in upcoming versions of elasticsearch - to ensure best compatibility, only use one type per index, or rework this function in the future to no longer accept document types.

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

- ElasticsearchDatabase.h
- ElasticsearchDatabase.cpp

4.2 cubesat::telemetry::ImageAssembler Class Reference

`#include <ImageAssembler.h>`

Public Member Functions

- **ImageAssembler** (uint32_t num_frags)
- void register_fragment (uint32_t fragment, std::vector< unsigned char > &&binary_data)
- bool is_full ()
- bool try_assemble_image (std::vector< unsigned char > &out_image_blob)

4.2.1 Detailed Description

Accumulates multiple binary data fragments and assembles them into a single binary blob containing image data

4.2.2 Member Function Documentation

4.2.2.1 is_full()

```
bool cubesat::telemetry::ImageAssembler::is_full ( )
```

Returns whether num_frags, the expected number of fragments for a full image, have been collected

4.2.2.2 register_fragment()

```
void cubesat::telemetry::ImageAssembler::register_fragment (
            uint32_t fragment,
            std::vector< unsigned char > && binary_data )
```

Adds an image fragment to be accumulated into a full image. `fragment` is the index of the particular fragment being registered (where in the sequence of fragments that form the image). `binary_data` contains the raw data, which is *moved* into the assembler.

4.2.2.3 try_assemble_image()

```
bool cubesat::telemetry::ImageAssembler::try_assemble_image (
            std::vector< unsigned char > & out_image_blob )
```

Returns whether num_frags, the expected number of fragments for a full image, have been collected If so, writes full image data to `out_image_blob`

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

- ImageAssembler.h
- ImageAssembler.cpp

4.3 cubesat::database::ImageDatabase::ImageData Struct Reference

Public Attributes

- std::string **name**
- ImageDatabase::format **format**
- std::vector< unsigned char > **binary**

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

- ImageDatabase.h

4.4 cubesat::database::ImageDatabase Class Reference

```
#include <ImageDatabase.h>
```

Classes

- struct ImageData

Public Types

- enum **format** { **jpeg**, **none** }

Public Member Functions

- ImageDatabase (boost::filesystem::path const &location)
- void write_image (ImageData const &image)

Static Public Member Functions

- static std::string get_format_extension (ImageDatabase::format format)
- static ImageDatabase::format get_format (std::string const &str_format)

4.4.1 Detailed Description

A store for images on the local filesystem

4.4.2 Constructor & Destructor Documentation

4.4.2.1 ImageDatabase()

```
cubesat::database::ImageDatabase::ImageDatabase (
            boost::filesystem::path const & location )
```

Create an image database in the supplied directory path `location`

4.4.3 Member Function Documentation

4.4.3.1 get_format()

```
ImageDatabase::format cubesat::database::ImageDatabase::get_format (
            std::string const & str_format ) [static]
```

takes a string of the image format and returns the corresponding enum type

4.4.3.2 get_format_extension()

```
std::string cubesat::database::ImageDatabase::get_format_extension (
             ImageDatabase::format format )  [static]
```

returns a string file extension of the supplied format

4.4.3.3 write_image()

```
void cubesat::database::ImageDatabase::write_image (
             ImageData const & image )
```

Store an `image` in the database (on the local filesystem, in the directory specified for the database instance)

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

- ImageDatabase.h
- ImageDatabase.cpp

4.5 cubesat::telemetry::Packet Class Reference

`#include <Packet.h>`

Public Member Functions

- **Packet** (std::vector< unsigned char > &&bytes)
- **Packet** (uint32_t initial_size=0)
- **Packet** (Packet &other)=delete
- **Packet** (Packet &&other)=delete
- void **operator=** (Packet &other)=delete
- void **operator=** (Packet &&other)=delete
- uint32_t available () const
- void **write_int8** (int8_t val)
- void **write_uint8** (uint8_t val)
- void **write_int16** (int16_t val)
- void **write_uint16** (uint16_t val)
- void **write_int32** (int32_t val)
- void **write_uint32** (uint32_t val)
- void **write_bytes** (std::vector< unsigned char >::iterator start, std::vector< unsigned char >::iterator end)
- void **write_bytes** (std::vector< unsigned char > &src)
- int8_t **read_int8** ()
- uint8_t **read_uint8** ()
- int16_t **read_int16** ()
- uint16_t **read_uint16** ()
- int32_t **read_int32** ()
- uint32_t **read_uint32** ()
- void **read_bytes** (std::vector< unsigned char >::iterator start, std::vector< unsigned char >::iterator end)
- std::vector< unsigned char > **read_bytes** (uint32_t length)
- uint32_t set_read_idx (uint32_t new_idx)
- uint32_t set_write_idx (uint32_t new_idx)
- uint32_t reset_read ()
- uint32_t reset_write ()

4.5.1 Detailed Description

Provides facilities for reading and writing primitive types to buffers of binary data, such as basic numeric types, or other buffers. Uses std::vector<unsigned char> as underlying buffer implementation, and maintains a reading and writing index as data is read/written linearly in the buffer.

4.5.2 Member Function Documentation

4.5.2.1 available()

```
uint32_t cubesat::telemetry::Packet::available ( ) const
```

Returns the number of bytes that can still be read

4.5.2.2 reset_read()

```
uint32_t cubesat::telemetry::Packet::reset_read ( )
```

Sets the reading index within the underlying buffer to zero

4.5.2.3 reset_write()

```
uint32_t cubesat::telemetry::Packet::reset_write ( )
```

Sets the writing index within the underlying buffer to zero

4.5.2.4 set_read_idx()

```
uint32_t cubesat::telemetry::Packet::set_read_idx (
             uint32_t new_idx )
```

Sets the reading index within the underlying buffer, from which future data is read

4.5.2.5 set_write_idx()

```
uint32_t cubesat::telemetry::Packet::set_write_idx (
             uint32_t new_idx )
```

Sets the writing index within the underlying buffer, to which future data is written

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

- Packet.h
- Packet.cpp

4.6 cubesat::telemetry::RockBlockReport Struct Reference

```
#include <TelemetryData.h>
```

Public Member Functions

- boost::property_tree::ptree get_property_tree () const

Public Attributes

- std::string **imei**
- uint32_t **momsn**
- boost::posix_time::ptime **transmit_time**
- double **iridium_latitude**
- double **iridium_longitude**
- double **iridium_cep**
- std::string **hex_encoded_cubesat_data**

4.6.1 Detailed Description

contains the data sent by rockblock API when reporting. For more info see: http://www.rock7mobile.↩ com/downloads/RockBLOCK-Web-Services-User-Guide.pdf

4.6.2 Member Function Documentation

4.6.2.1 get_property_tree()

```
boost::property_tree::ptree cubesat::telemetry::RockBlockReport::get_property_tree ( ) const
```

Returns a property tree containing this report's data

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

- TelemetryData.h
- TelemetryData.cpp

4.7 cubesat::telemetry::TelemetryConfig Struct Reference

```
#include <TelemetryConfig.h>
```

Static Public Member Functions

- static TelemetryConfig from_config (boost::property_tree::ptree &telemetry_conf)

Public Attributes

- std::string **reporting_index**
- std::string **reporting_doctype_name**
- uint32_t **image_frame_size**
- database::ImageDatabase::format **image_format**

4.7.1 Detailed Description

Configuration data for a telemetry handler module. Setting #reporting_index and #reporting_doctype_name affects indices and names of files stored to Elasticsearch Setting #image_format and #image_frame_size specifies the expected format and number of packet fragments for images sent from the satellite.

4.7.2 Member Function Documentation

4.7.2.1 from_config()

```
TelemetryConfig cubesat::telemetry::TelemetryConfig::from_config (
            boost::property_tree::ptree & telemetry_conf )  [static]
```

Returns a Telemetry Configuration object from a given boost property tree, `telemetry_conf` . The configuration must contain fields: "reporting doctype", "reporting index", "image size", and "image format".

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

- TelemetryConfig.h
- TelemetryConfig.cpp

4.8 cubesat::telemetry::TelemetryHandler Class Reference

```
#include <TelemetryHandler.h>
```

Public Member Functions

- **TelemetryHandler** (TelemetryConfig const &config, database::ElasticsearchDatabase &reporter, database::ImageDatabase &image_store)
- void on_request (std::shared_ptr< HttpServer::Response > response, std::shared_ptr< HttpServer::←
Request > request)

4.8.1 Detailed Description

Http "module" that handles telemetry data, received from the RockBlock API This module receives data, then processes the raw binary sent by a satellite, then archives the data to the given elasticsearch database and image store

4.8.2 Member Function Documentation

4.8.2.1 on_request()

```
void cubesat::telemetry::TelemetryHandler::on_request (
            std::shared_ptr< HttpServer::Response > response,
            std::shared_ptr< HttpServer::Request > request )
```

Handles http `request`; receives, processes RockBlock API data

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

- TelemetryHandler.h
- TelemetryHandler.cpp

Index