Gautebøye - File specification

Gaute Hope (gaute.hope@student.uib.no), 13.08.2012, Revision 1

Introduction

This document is a specification of how data is stored locally on the buoy and on the central logging point, versions **8 for buoy storage (DAT)** and **2 for central storage (DTT)**.

1 Overview

There are two file formats used: one binary and one ASCII format who both may fully represent the data. The binary format is used as local storage on the buoy SD-card while the ASCII format is used on the central logging point (Zero).

2 Data structure

2.1 Entity

The data consists of:

1. Sample: Values returned from AD converter (32 bit)
2. Timestamps
3. Position
4. Status (Validity of timing and position)

The formats are designed to uniquely represent these entities.

2.2 Structure

The data is organized in IDs with a number of batches, each batch has a reference and a set of corresponding values. The terms reference and batch are used interchangeably.

Store
- ID 1
  - Reference 0
  - Reference 1
    - Sample 0
    - Sample 1
    - ...
    - Sample 1023
  - Reference 2
  - ...
  - Reference 39
- ID 2
- ID 3
- ...

Table 1: Store structure

[1] TODO: Put some numbers on timing drift..

There is one set of data and index file for each ID, the index file contains information about the number of batches (references) and samples. The data file contains the references and the samples. The index file is not strictly needed for anything else than the data format version, but is used to check against data and as an possibility to read a data file faster.

2.2.1 Index

The index contains:

1. Version of file format
2. ID
3. Sample length (always 32 bit)
4. Number of samples (batch size · max references)
5. Number of samples per reference (batch size, 1024)
6. Number of references (40)

2.2.2 Reference: Timing, position and status

Each reference determines the position and status as well as a time reference for its batch of samples. That means there is not recorded a position and status, as well as timestamp, for each sample - but for a predefined number of samples. The time of each samples are calculated from the reference and the sample rate. The batch size is chosen based on the desired resolution of timestamps and position fixes and limited by the memory of the CPU unit, it is put to 1024 samples per batch, which results in little drift[1] between each reference and provides positioning every $1024/250\text{Hz} \approx 4\text{s}$.

The time is represented as microseconds since the UNIX epoch (1970-1-1 00:00:00), this is incidentally the same as the high precision time format used by MiniSEED [3].

Checksum As the data is sampled a 32-bit checksum is calculated for each batch, this is stored along with the reference and is used to check whether the stored data has not been corrupted and is the same that was recorded. The checksum is the XOR between all samples in the batch.

Entity A reference has the following entities:

1. Reference number
2. Time reference: microseconds since Unix epoch
3. Position (longitude and latitude)
4. Status
5. Checksum

Both formats use this structure, on the buoys local storage it is stored in the root folder of the SD card and on the central it is stored in a folder dedicated to its buoy.

2.2.3 Status

The status is a 16 bits wide with bits set for different conditions, the bit is set if the specified function is working, these are defined in buoy/gps.h:

Bit	Flag
0b0	NOTHING
0b1	HAS_TIME
0b10	HAS_SYNC
0b100	HAS_SYNC_REFERENCE
0b1000	POSITION

Table 2: Status flag bit mapping

NOTHING No flags set.

HAS_TIME A valid time is provided by the GPS unit.

HAS_SYNC There is a PPS signal being sent and the timing can be determined accurately.

HAS_SYNC_REFERENCE The reference in use was determined at a time when there was a PPS signal available. This can remain true after both time and sync is lost and the accuracy will deteriorate with the drift of the CPU clock.

POSITION The GPS is delivering a valid position fix. This will be true when HAS_TIME is true.

2.3 Sample

The sample is 32 bits wide and is an integer stored as two's complement. The least significant bit (LSB) indicates whether the full scale range was exceeded and the output has been clipped. If the value is at its maximum and the LSB is set (1) the output has been clipped, if the value is at its minimum and LSB is unset (0) the output has been clipped [2]. For correct interpretation of the sample the LSB should be set to 0 before attempting to read it as a 32 bit integer.

Throughout processing and storing the values are represented with 32 bits, even though the least significant bit may cleared after having converted the values to signed integers.

3 Formats

Since the files are stored in DAT format then read either directly from the SD card or sent over the network the DTT version of an ID is governed by the properties of its DAT file. The settings for size or batch length in the DAT will be seen in the DTT file.

3.1 Binary format: DAT

The binary format uses the extension DAT for its data files and IND for its index files, the buoy chooses the next free ID number on start up and as each data file is full continues. This puts a limit at the maximum number of IDs since the file system has a limit on the maximum number of characters.

Endianness The buoy runs on an ARM7 CPU and values are stored in little endian.

Version The version number, stored in the index, is updated whenever there is an upgrade of the buoy so that the data quality or integrity changes. The buoy version is not stored.

Implementation The implementation and latest version of the DAT file format is to be found in the source: buoy/store.h and buoy/store.cpp.

Limit of IDs The SD card is formatted using the FAT32 file system and the file names follow the 8.3 limit, that leaves 8 digits for the ID number and a maximum ID of $10^8 - 1 = 99999999$. With a maximum number of 40 references (batches) per data file and a batch size of 1024 values there will be 164 seconds of recording in each file. This means that for 14 days of continuous recording 7383 IDs are needed, in other words well below the limit [1].

Data file size The number of references (batches) and the data file size has been chosen to be in this order so that in case of data corruption in one file not too many samples will be lost, but still to be balanced against the overhead of more IDs and metadata. The more specific a batch of samples can be addressed the easier it is to retrieve them with the minimum disturbance of other batches or IDs. To avoid any interference with the current logging a data file is not transmitted before it is closed, that means that there is

a lag the length of the data file when watching the data in real time. The chosen length is a reasonable compromise between these.

3.1.1 Structure: Index

The index file for ID is named *ID.IND* and is when a data file is finished written as a binary file as:

Start	Field	Type	Size
0	Version	uint16_t	2
2	Id	uint32_t	4
6	Sample length	uint16_t	2
8	No of samples	uint32_t	4
12	Batch size	uint32_t	4
16	No of references	uint32_t	4
	Total length		**20**

Table 3: Binary index file, fields, start and size is given in bytes.

3.1.2 Structure: Data file

The data file is made up of batches, consisting of first one reference then a sequence of samples. The batches are concatenated after each other without spacing as described in table 1. The structure of the reference is:

Start	Field	Type	Size
0	Padding	0	12
12	Reference id	uint32_t	4
16	Time (reference)	uint64_t	8
24	Status	uint32_t	4
28	Latitude	char[12]	12
40	Longitude	char[12]	12
52	Checksum	uint32_t	4
56	Padding	0	12
	Total length		**68**

Table 4: Binary reference, fields, start and size is given in bytes.

The reference is padded with zeros ($3 \cdot \text{sample length} = 12$) in both ends so that it will be easier to recover parts of a corrupt data file as well as making it possible to detect a reference when none is expected.

Sample Each sample is an **uint32_t** with size 4.

Batch The structure of one batch is then:

Start	Field	Count	Size
0	Reference	1	68
68	Samples	1024	4096
	Total length		**4164**

Table 5: Binary batch, fields, start and size is given in bytes.

Such that a data file is given as:

Start	Field	Count	Size
0	Reference 0	1	68
68	Samples	1024	4096
4164	Reference 1	1	68
4232	Samples	1024	4096
...	...	...	...
162396	Reference 39	1	68
162464	Samples	1024	4096
	Total length		**166560**

Table 6: Binary data file, fields, start and size is given in bytes.

3.2 ASCII format: DTT

The ASCII format uses the extension DTT for its data file and ITT for its index files, they are created by the central logger (Zero) as they are transmitted from the buoy. A file may be incompletely downloaded and still be readable. The index file stores a list of all references and how much of it has been downloaded (chunks), normally a whole reference is downloaded at the time, but the logger may be configured to download only parts (chunks) of the batch at the time. All line endings are UNIX line endings '\n'.

The logger is written in Python so the file format has been influenced by this. It is read directly by MATLAB scripts which plot and do simple analyzes.

The central logger also keeps another index file, *indexes*, with a list of available IDs on the buoy and whether they are enabled (file exists and is valid on buoy).

The implementation and latest version is to be found in the source code: zero/data.py for index and data, and zero/index.py for the index of all available ids.

3.2.1 Structure: Index of ids

This is a list with one line per id of the format:

ID,enabled

1. ID: integer value id
2. enabled: 'True' or 'False', whether this ID exists on the buoy.

3.2.2 Structure: Index

There is one field per line with the following fields:

1. Local version (version of DTT format) (integer)

2. Remote version (version of DAT format on buoy) (integer)
3. ID (integer)
4. Number of samples (integer)
5. Number of references (integer)
6. HasFull, 'True' or 'False', whether a complete index has been received.

After that there is one line for each received reference, with a comma separated list of the following fields:

1. Reference number (integer)
2. Reference timestamp (integer), microseconds since Unix epoch.
3. Status (integer)
4. Latitude (string)
5. Longitude (string)
6. Checksum (integer)
7. Line (integer), specifying where this batch starts in data file.
8. Comma separated list of finished chunks (integers, variable length)

If the chunk size equals the batch size, there is only one chunk per reference. Meaning if the reference (batch) has been retrieved there should be one 0 in the last comma separated list. There cannot be a chunk size greater than the batch size and the batch size must be a multiple of the chunk size.

3.2.3 Structure: Data file

The data file is built up in a similar way as the binary data file, where all the batches follow each other, a batch consists of a reference followed by a sequence of samples. So that the timing of each sample can be determined from the previous reference.
The reference is one line of a comma separated list of the fields:

1. R (the character 'R' to indicate a reference)
2. Batch length (integer), value: 1024
3. Reference number (integer)
4. Reference timestamp (integer)
5. Status (integer)
6. Latitude (string)
7. Longitude (string)
8. Checksum (integer)

After this follows 1024 samples (integers) one on each line. The references are sorted descending, but there may be missing references (batches).

4 Implementation

4.1 Buoy: Local storage

The implementation is done in: buoy/store.cpp and buoy/store.h.

4.2 Zero: Central logging point

The implementation is done in: zero/data.py and zero/index.py. Reading is done with: zero/matlab/readdtt.m, where there exists various other scripts for working with the information.

4.3 Conversion and reading buoy local storage on a regular computer

4.3.1 dttreader

The program: zero/storetool/dttreader reads and print IND and DAT file pairs and can convert to DTT files as well as print the samples in a specified format.

4.3.2 dtttomseed

...

References

[1] Gaute Hope. "Gautebøye: Local data capacity budget". Local data capacity budget. 2012.

[2] Texas Instruments. *ADS1282 - High Resolution Analog-to-Digital Converter*. Revised 2010. 2007.

[3] Chad Trabant. *Libmseed manual and source code*. Version 2.5.1. IRIS Data Management Center.