# FDB (File-data-base) Documentation

# 1: Purpose of FDB, and why we chose it

FDB stands for File-Data-Base. Originally our storage method was called SS (after one of its’ creators, Steven Schenk) but this name was changed to FDB in order to make the name more generic and less offensive.

We created FDB for a very simple reason: It was our best option. We tried to integrate ORM (Object Relation Mapping) into our C++ code using ODB, but installing this framework on a Linux machine ,or any other machine for that matter, proved to be almost impossible. As it turned out, the ODB framework had had a bad reputation for quite some time.

For lack of any other real ORM system for C++, we decided to build our own storage method! Steven Schenk came up with the ambitious idea of creating a storage method ourselves.

# 2: Technical features and properties

FDB comes with the following technical features:

- Indexing of specified columns. By default, the CityGis CSV files have their DateTime and UnitID columns indexed since these are often used.

- Converting specified .CSV files to .FDB. The structure of the .CSV file is specified in a Tables.ss file in the same directory the .FDB files will be saved to. These Table files contain the structure of a .CSV file: Which columns it uses, what delimiters it has and which columns should be indexed.

- High speed saving of said files (+- 17 secs / 100mb).

- Retrieving column values, based upon date-ranges. I.E. “Give me all the car speeds between 2014 and 2015”.

FDB has some other properties:

- It is lightweight, consisting of only 3 small source files.

- It is written entirely in C++, so it is relatively fast compared to PHP- and Java- style filesavers.

In the next few chapters we will provide some more information about some of the aforementioned features and properties of FDB.

# 2.1: Indexing/Table files

The indexing of .FDB files is closely tied to the Tables.ss file that specifies the structure of a (or multiple) CSV files. A line from a Tables file would look something like this:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*table;rows;*

*--------;*

*positions;#DateTime,#UnitId,Rdx,Rdy,Speed,Course,NumSatellite,HDOP,Quality;*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The first line specifies that the name of the CSV file ends after a semicolon, and that after that semicolon the names of the columns are given until another semicolon is encountered.

The second line specifies that this is the end of what we call the “header” section: The section that specifies the delimiters between a CSV file and its’ columns.

The third line contains, in order: The name of the CSV file, the two columns that are to be indexed, and the rest of the columns.

Using this information, the create\_index() method creates and –index.fdi file that holds the indexed values. The create\_index() method also takes into account that bigger .CSV files are spread across multiple numbered .FDB files, and creates multiple –index.fdi files if the CSV file gets too large.

# 2.2: Conversion/Saving

The conversion from CSV to FDB also makes use of the tables file. For each row in the CSV file, the insert() method checks wether the given row belongs to a column designated as an index; if so, that row is indexed.

The insert() method creates a new .FDB file containing all the information that was in the original CSV file every 10.000 lines, in order to improve writing speeds. If a CSV file is larger than 10.000 lines, a new FDB file is created with an incremented number after the filename.

Originally, our insert() method and our csvreader() (This being a method that reads the content of a newly added .CSV file and passing it on to fdb.insert()) used Lists of maps of strings, which were relatively inefficient. Our FDB now uses lists of strings, which are much faster. This fact, combined with our constant refactoring and use of C++, makes FDB quite fast! Writing a 100-110mb file costs about 20 seconds or less.

# 2.3: Retrieving values

When we want to read values from an existing FDB files or set of files, we use the select() method. The select method takes as input three or more strings called indexes and a table file: The first two indexes specify on which date-range select() has to search, from 1-1-15 to 1-1-16 for example. The third index specifies which column values should be returned: \* will return all rows from all columns, “events” will return all rows from the events column that match the date-range and so on.