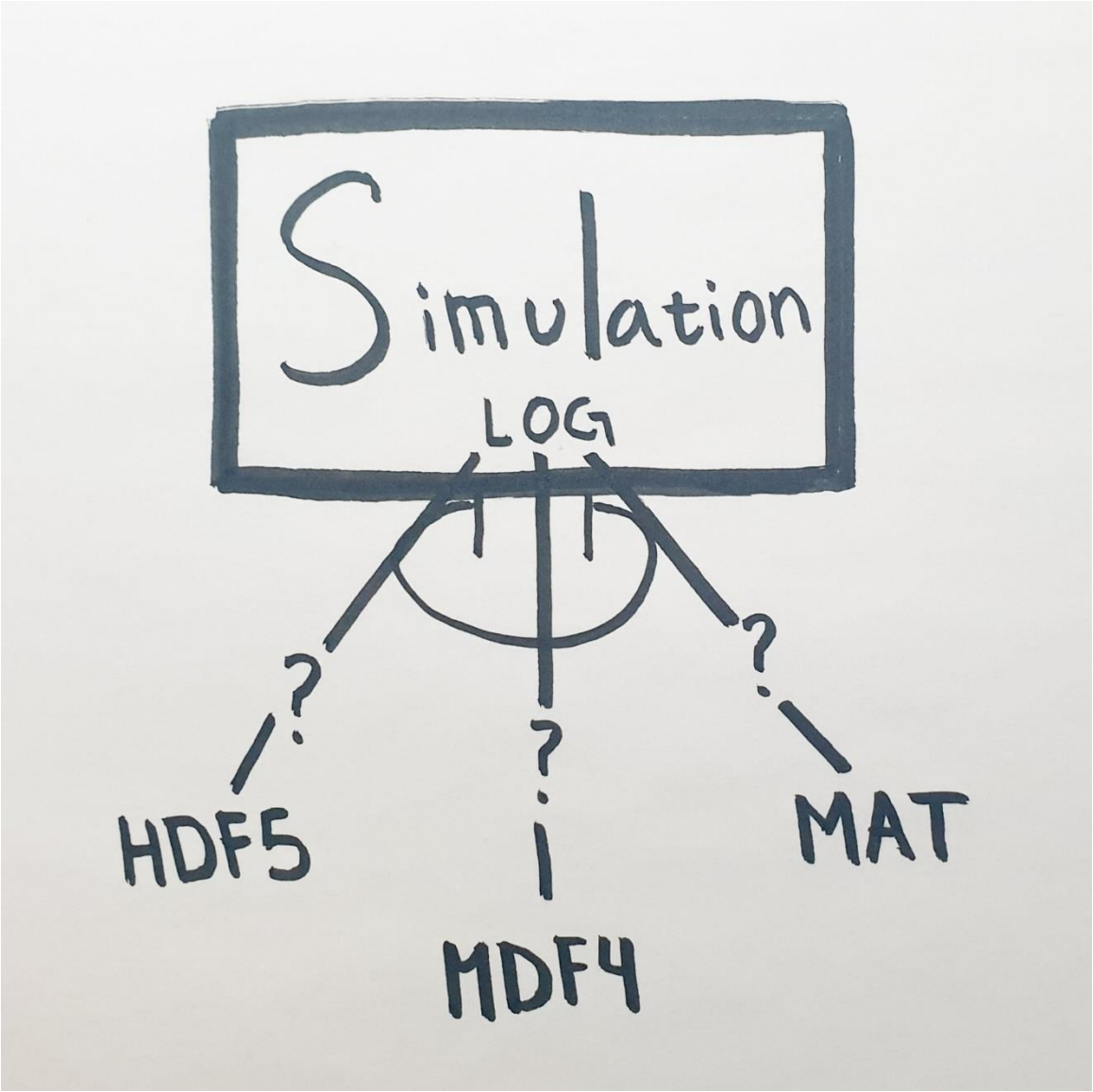
CHALMERS



Simulation signal logging and log file format decision

Bachelor's thesis in Computer Science and Engineering

Rebecka Axelborn

Noora Alsadawi

DEGREE PROJECT REPORT

Simulation signal logging and log file format decision

Rebecka Axelborn
Noora Alsadawi

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Simulation signal logging and log file format decision

Rebecka Axelborn

Noora Alsadawi

Examiner: Jonas Duregård

Department of Computer Science and Engineering

Chalmers University of Technology / University of Gothenburg

SE-412 96 Göteborg

Sweden

Telephone: +46 (0)31-772 1000

Cover:

The simulation logging pointing to the three different file formats used in this report.

Department of Computer Science and Engineering

Gothenburg 2022

ABSTRACT

The SIL team at Volvo group trucks works with creating simulations for testing, when these simulations are run, they produce log files where data is saved continuously. The current log files are of a comma-separated values (CSV) format which is slow, contains a lot of redundant information and has limitations when storing complex datatypes. The aim of this project is to create log files with a new flexible format based on how easy it is to write the file in C++, how easy it is to read the file in MATLAB and how easy it is to read the file in python. One format from the candidate formats (MAT-file, MFD4 and HDF5) is to be selected based on the criteria mentioned before.

Due to the quality of documentation and the available libraries, HDF5 was selected among the candidate formats.

The result of the implementation was the generation of an HDF5 log file inside the simulation that is easy to present signal data with and makes it easy to deal with the data. A visualization program for the log files was also created to visually ensure that the log files were generated correctly.

Keywords: CSV, MDF4, HDF5, MAT-file, visualization, simulation, C++, python.

SAMMANFATTNING

SIL-teamet på Volvo Group trucks jobbar med att skapa simuleringar för testning, när dessa simuleringar körs producerar de loggfiler som sparar data kontinuerligt. De aktuella loggfilerna är av ett kommaseparerade värden (CSV) format som som är långsamt samt innehåller mycket överflödig information och har begränsningar vid lagring av komplexa datatyper. Syftet med detta projekt är att skapa loggfiler med ett nytt flexibelt format baserat på hur lätt det är att skriva filen i C++, hur lätt det är att läsa filen i MATLAB och hur lätt det är att läsa filen i python. Ett format från kandidatformaten (MAT-file, MDF4 och HDF5) ska väljas utifrån de kriterier som nämnts tidigare.

På grund av kvaliteten på dokumentationen och de tillgängliga bibliotek som fanns, valdes HDF5 ut bland kandidatformaten.

Resultatet av implementeringen blev att generera en HDF5 loggfil med det nya formatet som gör det lätt att presentera signaldata med och hantera att hantera datan. Ett visualiserings program skapades också för logg filerena för att visuellt säkerställa att loggfilerna genererats korrekt.

Nyckelord: CSV, MDF4, HDF5,MAT-file, visualisering, simulering, C++, python.

ACKNOWLEDGEMENTS

We would like to thank our supervisor Sakib Sistek at Chalmers, Tobias Falkman, Dr. Javier Jalle and Anand Hariharan at Volvo Group for their great support and direction to complete our thesis work.

Table of Contents

1. Introduction

Almost all companies today work with technology in one way or the other. The technological part of the company may be the main product, it may be a sub product, and it could be an aid in the creation of a product or the management of it. What the usage of technology in the companies has in common is that the purpose is to make everything more effective and better. That is exactly what this project contributes to, making the testing of software easier and faster.

1.1 Background

This project is given by the SIL team in the department BMLT at Group Trucks Technology (GTT) at Volvo Group Trucks. The team provides the information of requirements and the preexisting code for this project and aids in guidance whenever necessary. Volvo group trucks is a company that, as can be deduced by the name, works with developing and building trucks and GTT is the organization within Volvo group trucks for research, engine development and product design linked to the development of the group's trucks [1].

BMLT which stands for Build Measure and Learn Technology, is a section of Volvo group trucks technology that exists with the mission to define, develop and maintain development pipelines as well as providing fast and valuable feedback. This in order to enable an innovation cycle to ensure customer satisfaction [2].

The team that leads this project is the SIL team which stands for Software in the loop. What they do is creating virtual truck rigs which aids in faster error detection in the development of software for the trucks. The developers can use the data from the virtual rig to analyze if the software works as expected.

1.2 Purpose

When the SIL team run the virtual rigs, they produce binary data over time that is saved in log files. The current log files are written in a Comma Separated Values (CSV) format. The CSV format is quick to learn and easy to use, which is why it has been used during the development of the virtual rig.

The most important disadvantage to CSV for this project is that they are unable to handle complex data, which is necessary for the log files. The CSV files also has no way of representing binary data, meaning that it must be transformed into plain text before it is stored, which is not time efficient. Because of this, a new format needs to be implemented.

The purpose of this project is to research three formats and implement one of them. The format is to be chosen based on three criteria: how easy it is to write code for generating the file using the programming language C++, how easy it is to read the file in the platform MATLAB and how easy it is to read the file in python. One demand is that the libraries used are open source, a library that is freely available to use, to avoid new dependencies within the company.

1.3 Goal

The goal of this project is to choose a file format and library based on the criteria mentioned in purpose and to then write the code, using the chosen library, for generating the logfiles within the already existing simulation code. The file formats to research are MDF4 (Automotive), HDF5 (Big data) and MAT (MATLAB). After researching file formats, the goal is to create flexible log files containing different data types such as string, integer, double, arrays and structs. The preexisting simulation code in this project is written in the programming language C++. The library chosen therefore must be a C++ library. The first implementation goal is to implement code for saving the binary signals of the 64 bit unsigned integer datatype in the log file. If there is time left, the next step is to implement complex datatypes such as strings and structs. After the logfile generation implementation, the goal is to create a visualization of the data from the generated log files in python using some framework for GUI.

1.4 Limitations

This work will focus on backend development, to generate log files with a new output format instead of the current output format CSV and then visualize the data from the generated log. This work will not deal with any hardware. The simulation software used in this project will not be changed apart from the file containing the logging. Ensuring that the file formats can be read in MATLAB will not be done by testing it in the MATLAB software, only by reading documentation from MATLAB.

2. Theory

This section gives an overview of the formats used in the project to understand the process and the choices that were made. This section also contains some extra information about how some systems work in order to understand some of the problems encountered.

2.1 Comma Separated Values (RFC 4180)

CSV, with a strict form called RFC 4180, is a simple format for saving data. The data is saved in a two-dimensional array with each column separated by the comma character and each row separated by a line break. The file can contain a header with the same format as the other lines, but here the header contains a naming of the columns [3], see example in listing 1. The data in CSV files are saved as numerical and textual values, often using US-ASCII as character set [3].

```
TIME,DEGREES,SPEED
10:40,25,50
10:54,27,78
11:20,30,100
```

Listing 1: Example of a csv file with a header.

The CSV format has many advantages, some of these advantages are its readability, that it is fast to manage as well as quick to learn, easy to implement and that the CSV file is a relatively small file. However, the CSV format has some important disadvantages. The format has no way of representing binary data and therefore must first convert binary data to plain text before storing it, which is time consuming. The CSV file format also does not support complex data [4].

2.2 Measurement Data Format version 4

MDF (Measurement Data Format) is a format originally designed by Vector Informatik GmbH and Robert Bosch and was created with the automotive industry primarily for the area of ECU development, to store signals from the vehicle data bus for documentation and post processing. ECU, which stands for Electrical Control Unit, is an embedded system used in the automotive industry to control electrical systems or subsystems in a vehicle [5]. The first released version of MDF was MDF 2.0 which was released in 1991 and later developed and released as MDF 3.0 in 2002. In 2008 a working group called ASAM started a revision and standardization of MDF which later released MDF 4.0, the version used in this project, in 2009. The new version removes the strict size restrictions of the previous version, which was 4 GB, it provides memory-efficient storage for channels with constant value or variable data length, storage of bus traffic for common data bus systems and storage of additional information about the measurement environment [5].

2.2.1 File Structure

The MDF4 format has a lot of advanced features, this project only requires basic functionality and only basic functionality will therefore be covered. MDF4 format is sorted in a tree like structure where the nodes of the tree consist of various blocks [5]. There are different types of blocks and the block type defines its purpose and content. The blocks that will be in focus for this project is Identification block (ID), Header block (HD), Data group block (DG), Channel group block (CG), Channel (CN) and Data group block (DT) [5].

The identification block is 64 bytes long and its main purpose is to identify the file as an MDF file and to specify the MDF version. The header block contains the general description of the MDF file and can be seen as the root of the tree structure since it will point to the relevant blocks of the file. The ID and HD blocks are the only blocks that will occur only once in the file structure. All other blocks may have multiple instances depending on what data is to be stored. The DG block points to a DT block containing the measurement data stored in records. A record is identified by a record ID and stores signal values that have been sampled at the same time, and therefore has the same timestamp. The DG block also points to a CG block. The CG block and its members are connected to the record ID and describes how to read the data that is stored in the records with the connected record ID. The CG block in turn points to possibly multiple CN blocks. Each CN block represents a signal value such as time, speed, temperature [5]. To get an overview of how the blocks are connected see Figure 1.

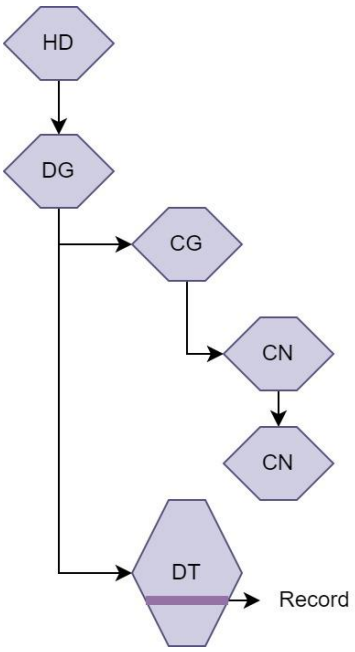


Figure 1: Chart of the file structure of the MDF4 file.

2.3 Hierarchical Data Format version 5

HDF5 is a hierarchical data format created for all industries to manage big data and complex data. The file in hdf5 is an object in itself and can be seen as a container for heterogeneous data objects, here called datasets. The information in this section is mainly gathered from the official web page of the HDF group who created the format to ensure correct information

2.3.1 File Structure and usage

The HDF5 file can be structured in any way the user needs. The data sets can be sorted into so called groups that can be structured into a tree like structure with datasets in groups within groups. A dataset contains metadata such as dataspace and attribute as well as the binary data from the measurement which will all be explained in this section [6]. An overview of the file structure can be seen in figure 2.

2.3.2 Group:

HDF5 contains groups in order to create a hierarchy in the data format. Groups are used to create a structure for the different datasets, this is done by one group linking to another group which links to datasets. The structure used for is where there is a root folder called "\" which links to groups or datasets on lower levels. This can be compared to the file structure of the UNIX operating systems [6].

2.3.3 Dataset:

A dataset is a multi-dimensional array of data elements that can be extensible in any dimension. Each dataset has two important parts which describe how the data is stored, those are dataspace and datatype. Each dataset needs a name, datatype and dataspace to be created [6].

2.3.4 Datatype:

An HDF5 datatype describes the data type of the data element in the multidimensional array. The datatype provides information that can be used for data conversion to or from that datatype. There are many different types such as integer, float, string, array, enum, compound (similar to struct in the C programming language) [6]. All elements of the dataset must have the same type.

The data type can be divided into two parts: predefined and derived datatypes. The predefined datatype is created by HDF5 which is used for simple purposes, these datatypes contain two parts: standard and native. The standard types have symbolic names of the form H5T_arch_base where arch is an architecture name and base is a programming type name, for example, H5T_IEEE_F32BE point to big-endian, IEEE floating point. The native is used to handle the memory operation and ease this operation for example H5T_NATIVE_INT point to an int in C language and H5T_NATIVE_UINT, point to an unsigned int in C. The derived datatypes are created from predefined data, the most common types in the derived data types are string and compound [6].

2.3.5 Dataspace

The dataspace is the size and shape of the dataset which has different types, the first type is null dataspaces it does not have any element. The second type is simple dataspaces that are the most common, which can be a multidimensional array of elements. The last type is scalar dataspaces which are for a single element, usually a string. To create the dataspace it needs a rank which sets the number of dimensions, it needs the initial dimension size and the maximum dimension size allowed [6].

Dataspace can be unlimited or limited depending on the dataset.

2.3.6 Attribute

Attributes is a small metadata object that has a name and a value. Attributes can be added to any HDF5 object including datasets and group. Attributes look much the same as a miniature dataset so they have their own dataspace and datatype. Moreover, attributes can't uphold partial I/O and cannot be compressed. It is designed to be small [6].

2.3.7 Properties

A property is a characteristic of an HDF5 object. The properties for an HDF5 object are to store data contiguous by default. For more advanced needs the data can be stored in chunks or in compressed chunks [6].

2.3.8 Hyperslab

A hyperslab is a selection of elements from a hyperrectangle. That pattern is defined by four arrays which are start, stride, count, and block. Start is the starting position, stride is the number of elements that separate each block, count is number of blocks and block is the block size [7].

2.3.9 Software

HDF5 source code is written in C and has APIs for C++, Fortran, Java and Python. The website for the library contains many examples describing many use cases in the different languages. There are multiple tools that are provided for HDF5 by the same group that created the library, one tool is a h5dump that can be used to show the structure of a HDF5 file and its content in the command line. Another tool is HDF5 view which is a java browser with a GUI that shows what the structure of the HDF file looks like and allows the user to easily browse for datasets [6].

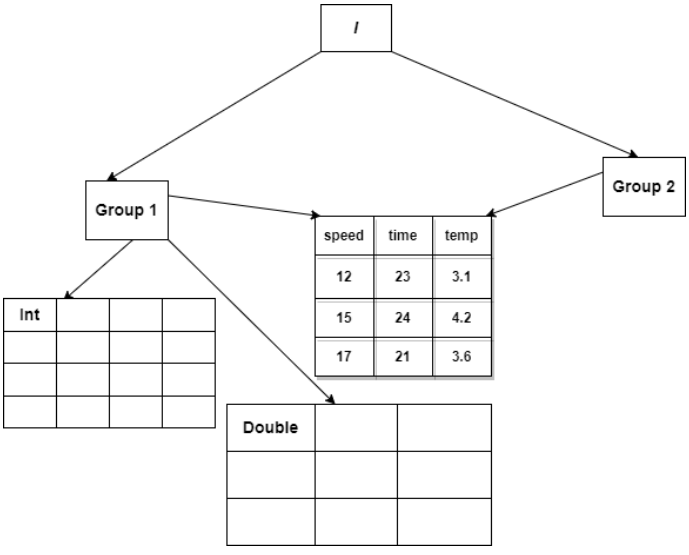


Figure 2: HDF5 tree structure with groups containing datasets of different datatypes.

2.4 MAT-File Format

The MAT-file is a file structure that stores data in binary form. By using tags the MAT-file format provides a way to quickly access data elements within the file [8]. The MAT-file format begins with a header that can contain plain text describing the level of the MAT-file, the platform on which it was created and date and time that the file was created. After the header comes the storing of data. The data is stored in a so-called Data Element Format. Each data element begins with an 8-byte tag followed by the data. The tag contains a description of how to interpret the data and how many bytes of data is in the data field. The data field has a size of 64-bit. If the data saved is less than 64 bit it has to be padded to make sure the tag of the next data starts on a 64-bit boundary. If the data is very small the data can also be saved in an 8 -byte format [8]. Reading the example Data element in [8] it shows that a Data Element can contain multiple values in the data field.To get a visual of how the file is structured, see figure 3.

According to [8] the MAT-file can save the data as the datatype miMATRIX to represent both character arrays, which can be interpreted as a string, and structures. For this miMATRIX datatype, the data field can contain sub elements. Each sub element also has a tag describing the data similarly to the Data element tag. The difference is that there are fewer datatypes. The array data element consists of different sub elements to describe the array. The ones that are consistent for all types of arrays are one sub element for array flags, one for dimensions array, one for array name, one for real numbers and one optional for imaginary numbers [8]. In the flag sub element the class of the array can be set, meaning that it can be set to describe the type of the array for example if it is a character array or a struct.

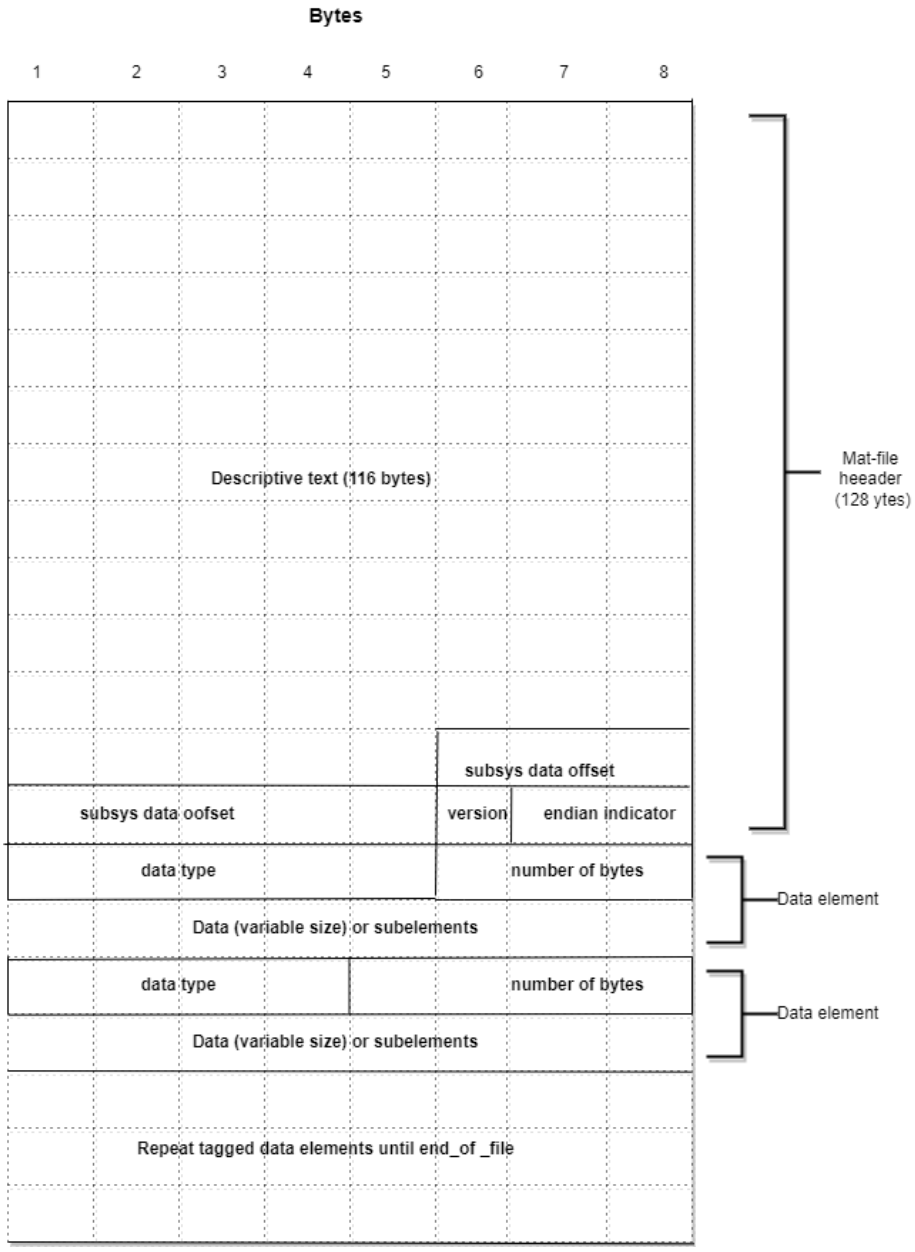


Figure 3: Visual description of the MAT-file structure

2.5 Dynamic and static libraries:

A library is a collection of prewritten code that allows programmers to reuse that code in a program without needing to write it.

2.5.1 Static library

A static library is a type of library which is a collection of routines that are compiled directly into the program, the file extension of a static library on Windows is usually “.lib”, while on Linux the extension is “.a”.[9]

A static library allows the user to easily run and distribute the program since the library becomes a part of the user's executable. This makes it so that when distributing the program only the executable is needed and not an additional distribution of the library. Another helpful part of a static library is that it can guarantee the right version of the library because the library is a part of the program's executable [9].
Despite all these advantages, the static library has a drawback which is waste of space because the copy of the library is a part of the executable that uses it. It also makes it difficult to update the library for the program when needed [9].

2.5.2 Dynamic library

A dynamic library is a set of routines that are loaded in the application at run time. This library will not be a part of the executable and will, at runtime, instead produce a dynamic library where the library remains as a separate unit. On Windows the dynamic library has the file extension ".dll", while on Linux it is the extension ".so".[9].

There are many advantages to using a dynamic library, it will save space since one copy can be shared between the programs, it is also easier to update the version without having to replace all the executables that uses the library [9].

According to [9] the disadvantage of using the dynamic library is that the library has to be explicitly linked to the program which can be confusing and difficult to do for the user.

3. Method

The method used in this project is the agile methodology instead of the common waterfall methodology. The waterfall methodology is a work process where the small parts of the project are finished before moving on to the next. This makes the process rigid and makes it more difficult to back track later in the project if some parts turn out not to work the way it was intended. The agile work method is instead a work process where the small parts are started on somewhat simultaneously to create a prototype of the entire project that can do the basics. The project can then be back tracked to further develop the system to handle more complicated stuff.

The project is split into sprints of four week. The first sprint consists of the research part of the project. The two formats received a time limit of two weeks each. The definition of done for this step has two alternatives. One alternative is when the two-week span has run out, proving the format or its libraries to be too difficult. The second definition of done is when a file has been created with data in form of different datatypes. The datatypes required in the first sprint are integers, strings, doubles and arrays. The more complicated datatypes will be backtracked later in the project according to the agile methodology.

The second sprint consists of implementing the chosen format integrated with the simulation code written by the SIL team. The definition of done for the implementation has multiple steps until being completely done. The first step is to implement the creation of a file within the simulation environment. The first step only requires the 64-bit unsigned integer data to be saved. In the second step the program must also extract the datatypes from the simulation to be saved. A potential third step is to also save the complex datatypes.

In the third sprint the implementation will continue, and the visualization part will be started. In this part, a way of visualizing will be decided. Based on the time left in the project the visualization may be implemented in a simple way in python, if there is more time, it will be implemented using an analytics visualization engine written in python created by the DAVA team at the company. This requires help from the DAVA team. The definition of done will be when some of the simple signals can be visualized on an XY-graph.

3.1 Tools

This section provides information about the different tools used to work on this project.

3.1.1 Microsoft visual studio professional

Microsoft visual studio professional is an integrated development environment that was used in this project [10]. The 2017 version was used to match with the SIL team’s development environment.

3.1.2 Microsoft teams

This project is almost entirely being implemented remotely so it is one of the important tools that were used for communication, meetings as well as for discussion. In this program it is also easy to create and share the files [11].

3.1.3 Bitbucket

Bitbucket is a git repository management solution, this tool is used for code sharing and version control. Bitbucket has many different features including restriction of access to the code, it can control the flow of the work (progress in work), as well as it provides pull requests with in-line commenting for code reviewing and finally is able to track the development by using Jira integration [12].

3.1.4 Stack overflow

Stack Overflow is a one of the most popular public platforms that was founded in 2008. It is used by individuals who code to gain information for learning and also for knowledge sharing. The platform is used by 100 million people every month [13]. In the Stack Overflow platform one can find the answer to many questions or technical issues. For this project stack overflow is used to get a quick overlook on how likely it is to get help from with the library, by seeing how many results appear when searching for each library.

4. File format decision

Before starting the implementation of the system a choice of format has to be made. The demands for the format are that it must be an open-source library.

4.1 MDF4 format

The MDF4 standardization page [5] links to various libraries for MDF4 but only one of them are open source. The library linked in [5] is a library from turbolab [14] provided by Michael Bührer and Bernd Sparrer. Starting out using the library there were some minor issues. A new Software Development Kit (SDK) had to be downloaded, there was an imported header file called VLD.h that gave us an error, the solution was to simply remove this because it did not cause any errors when removed and no other solution was found. This library seemingly does not have any documentation where the user can look for available functions. What it does have however, are some examples of reading and writing and a viewer program for the files. When searching for MDF4 in stack overflow only 15 result were found, meaning very little help can be expected from there. In this project the only interest was the writing part in C++ which is what was focused on when learning the library. Using the MDF4 library as it is in a new project was not achieved. However, running the writing example was successful. The library did also provide a program to view the mdf4 files. This program did successfully run and was able to view the created mdf4 files.

Though running and successfully creating a MDF4 file where both a data group and channel group had been generated it was difficult to understand and do anything beyond what was already in the example. Writing values to the data block was very difficult, since the existing available examples used functions that did not exist in the library. The only part managed was to write integer values. Writing the other datatypes, such as strings, turned out to be very difficult to understand how to do since there was no examples and no function documentation to try to find out how to use it. In the end, what was managed was to write only integer values. The writing of the rest of the datatypes was unsuccessful.

MathWorks, who created MATLAB, has a web page [15] that shows an example of how to read an MDF4 file in MATLAB when using the Vehicle Network Toolbox which for this project sufficiently proves that MDF4 files can be read in MATLAB.

Reading the values of data types in python was attempted. The library was installed and imported in python but there were error messages after attempted installation. At this point a lot of time had been spent on this library and the decision was made to move on to the next file format.

4.2 HDF5 format

Finding an HDF5 library did not take a lot of time. The same group that has created the standard has also created an open-source library with multiple releases [16]. The library has a support page with a lot of examples for both the basic functionality [17] and the more complicated functionality [18]. There are examples for writing both strings and compound types which can be used for structs. The examples for the more advanced functionality only have examples in C but seem to work for C++ as well with minor alterations. The library also has a full documentation of the C++ API of all classes and functions [19]. When searching for

“HDF5 C++” in StackOverflow there are 500 results meaning there is a possibility that some help with issues can be found through this page. The same group also provided a viewer to view the HDF5 files which is very user friendly.

4.2.1 Writing different data in HDF5

The first step to try out this library was to create a "hello world" application to try out the basics such as creating a file. The file needs a filename and as well as a file access mode which is used to regulate how the file can be opened and closed. After creating the file, creating a dataset was attempted. To create the dataset it needs three parameters to be created which are dataspace, dataset name and datatype.

The first parameter was a dataspace, for writing integers and doubles the dataspace needed to have a rank of size 2 and a dimension of the desired set size. The rank describes the number of dimensions and the dimension decides the size. For a string it was different since the dataspace needed to be a scalar type, which did not need any dimension size and the rank of the scalar type was always zero. And finally for an array, it was a bit similar to the integer and the double cases, the difference was the rank should be 1 instead of 2 since it should be one dimensional array of elements and not a two-dimensional array of elements such as integer and double cases.

The second parameter was the name of the dataset which should be different for each dataset, so each dataset should have its own name. Lastly, the third parameter was the type for the dataset that can vary between datasets. The type should be of a standard predefined type that describes the contents of the dataset. The datatype was therefore different for writing integers, doubles, strings and arrays.

To write the data to the dataset a C++ array was used with test data that was the same size as the previously set dataspace. This data together with a predefined native type was then used in the write function. This native type also differentiates depending on what type of data is to be written. The write was successful and could be opened in the viewer program that can be seen in figure 4, where the test data could be seen successfully written into the dataset. Creating the file with the dataset was simple and straight forward when getting help from the examples and the C++ documentation.

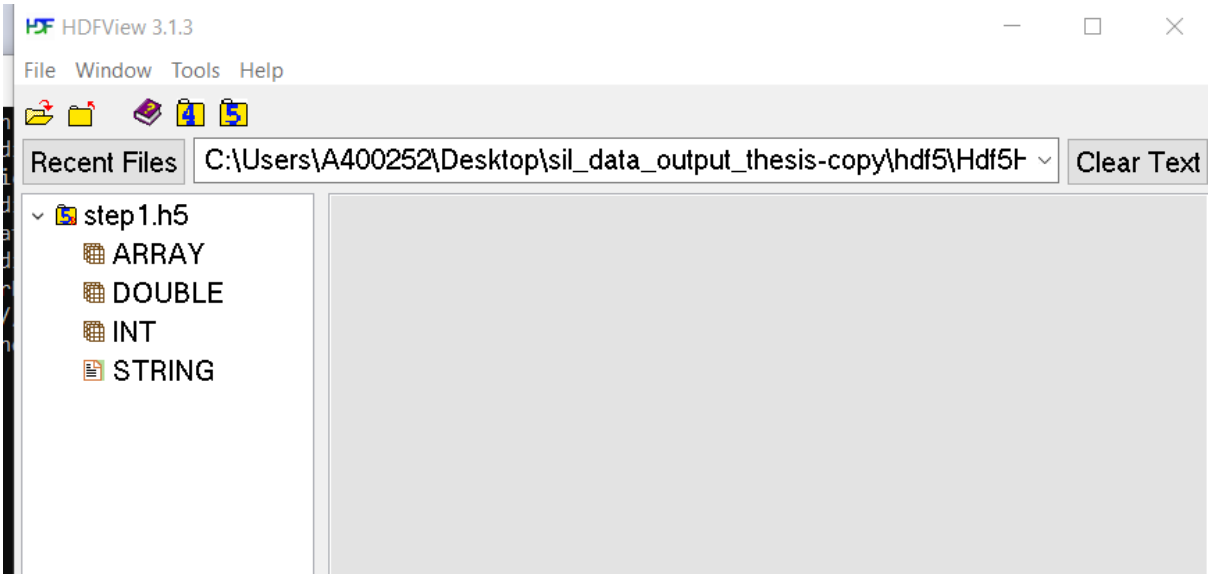


Figure 4: A screenshot from the result of the test program showing the file step1.h5 with datasets of the different datatypes.

4.2.2 Reading in Python:

Implementing a function for reading HDF5 files in python was simple. The first step was to open the existing file, which was previously created in C++. To open the file the function needs the path of the file. For reading the data that had been written in C++ for all types, it needed to get the HDF5 datasets for different types. A loop was used to go over all datasets and access each line of the file and print it to the console to prove that the read works. This was successful and created in only a few hours.

4.2.3 Reading in Matlab:

The MathWorks web pages [20] [21] describes and provides examples on how to read HDF5 files in MATLAB. This is sufficient evidence for this project that the files can be read in MATLAB.

4.3 MAT-File

Because of set time restrictions to be able to finish this project in time, this file format was spent less time on that the previous once. There was no time left to create a test project to try out the library.

The format is created for MATLAB by the same company that created MATLAB and it therefore seems obvious that the files can be read in MATLAB. But to prove this further it says in [8] that MAT-files are compatible with MATLAB, with some version conditions

When looking for an open-source library in C++ there was no obvious library to use. MathWorks appears to have the possibility to write mat-files in c++ but only through MATLAB editor [22] which requires a MATLAB license, thus not being open source. An open-source library called matio [23] was found by searching in google. This library claims to

be able to read and write MAT-files. This library is a C library, but while continuing the search, a C++ wrapper for this library was found [24]. The library provides some examples of how to use it, but they are very basic and would not provide much help when working on the more complicated tasks such as saving structs. When searching for the library in stack overflow there are a total of 71 results. The matio library has a dependency to the HDF5 library described in 4.2.

4.4 Decision

The decision was made based on above research, that has been presented clearer in figure 5, to use the HDF5 format, because this format had been tested to successfully write all needed datatypes in C++ and could read all these datatypes in python as well. For the implementation part in the next section, HDF5 will be used.

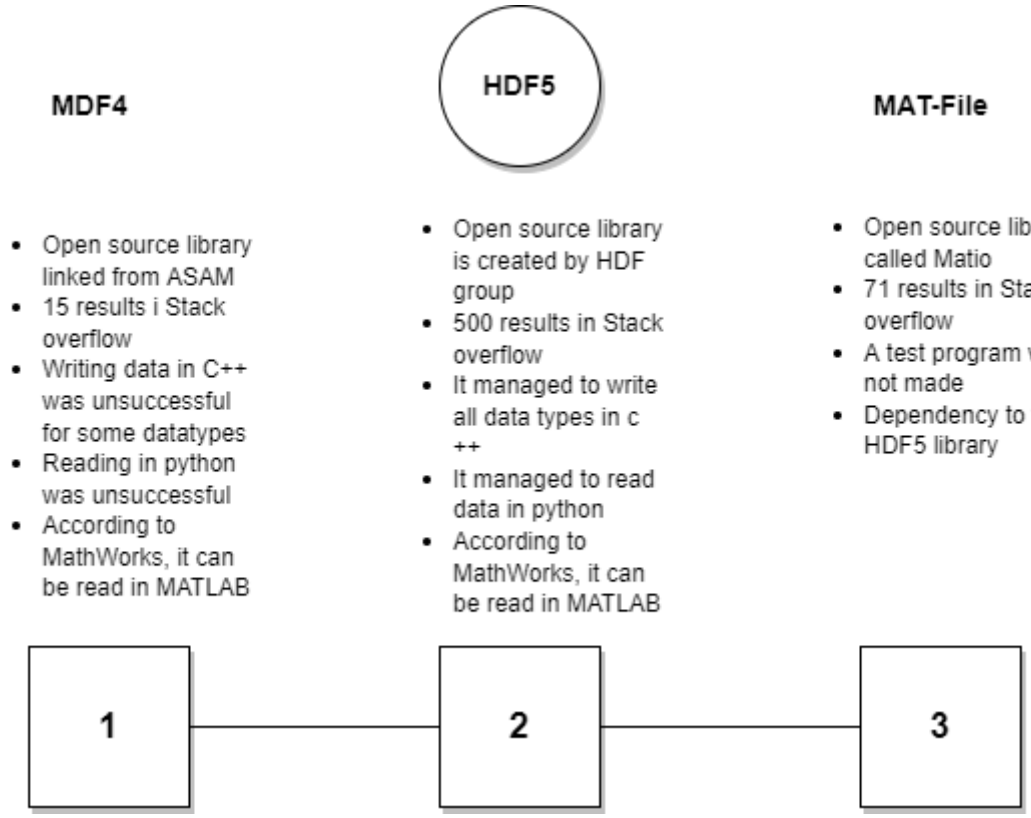


Figure 5: The different research results for the file formats in this project, with the chosen format circled.

5. System Implementation

The product of this project is to generate logfiles of the chosen HDF5 file format for the simulation output, using an open-source library. This is to be implemented in the simulation code instead of the CSV format. To do this, changes had to be made in a program which is already connected to the simulation. For test purposes no live simulation was done, but a signal re-player that plays recorded signals which would behave like a real simulation.

The simulation program that the SIL team has built has a command line interface (CLI) where the user can enter commands to start the simulations. In this CLI window the user can initialize, start the replaying of certain signals, step through the signal changes, set which signals to log and set the name of the file. For each command a connected part of the code is run. The parts of the code that are relevant for this project are the file initialization part where the file is to be initialized, the signal initialization where the properties of the signal are set in relation to the file and the part that runs each time the signal has a change of value.

The simulation has three files that were relevant for this project. It has the file with the functions and code that was already connected to the simulation as well as a header file to declare variables and functions. Another important file is the signal header file where signal specific variables are declared which can be used to access information about the different signals.

5.1 First version

In the beginning of the project, it was attempted to implement everything needed in one section of the code at one time, which resulted in getting stuck for a long time on details. To further avoid this, the project was split into smaller parts and the decision was made to create a first simpler version. To make the process easier a separate HelloWorld program was used where the functionality could first be tested to see that the code works as intended. This code then had to be separated and slightly adjusted to fit into the simulation project.

The first step was to do the code for the initialization. In this the filename must be set, which for the first version was hardcoded. For the first version it was decided that the dataset would have a set size, which would have to be further developed later. Because of this a class member variable dataspace was created that could be reached from all functions. The size was set to 2 columns and 10000 rows. The 2 columns are for time and value and the rows are to save every signal change. The information received from Volvo was that the replayed simulation that was used for testing would not have more than 10000 signal changes which is why this works for the first version. This is not applicable to all possible scenarios though and will later be further developed.

In the HDF5 library, the code for creating and opening an already existing file is the same. Therefore, for the first version, instead of creating a class member for the file which is somewhat complicated with pointers, every time the file object was needed, the file function with the hardcoded name was called to retrieve the file object. This works but it takes up unnecessary memory and CPU usage.

The next issue was writing the data one row at a time. The simplest way of writing data in HDF5 is by writing all data at once to the dataset in a big array. This is not ideal for this project where the data needs to be written row by row every time a signal changes. To be able

to write row by row an index variable is needed to keep track of which row in the dataset to write to. This index had to be kept in the signal header file since the index will vary between signals based on how many times the signal changes. To write to a small section of a dataset a so called hyperslab selection is needed, see section 2.3.8. The hyperslab needs the arrays called start, stride, count, block to know where in the dataset to mark, this can be seen in listing 2. Once a hyperslab is marked in the place of the index a write can be done together with a dataspace with the size of one row. And with this the data can be written row by row.

```
const int DIM0_SUB = 1;//Dimentions for write space
const int DIM1_SUB = 2;
hsize_t start[2] = { index,0 };
hsize_t stride[2] = { 1,1 };
hsize_t count[2] = { DIM0_SUB, DIM1_SUB };
hsize_t block[2] = { 1,1 };
hsize_t subDim[2] = { DIM0_SUB,DIM1_SUB };
// Set sub space for writing to hyperslab in dataset
H5::DataSpace rowDataSpace(2, subDim);

//Gets resized dataspace
DataSpace originalDataSpace = signalDataSet.getSpace();

//Select hyperslab to write data to
originalDataSpace.selectHyperslab(H5S_SELECT_SET,
 count, start, stride, block);
signalDataSet.write(data,
PredType::NATIVE_UINT64, rowDataSpace, originalDataSpace);
signal->incrementIndex();
```

Listing 2: Piece of the code for writing to one row in the dataset. The arrays start,stride,count and block are used to select a hyperslab. The array subDim is used to create a dataspace the size of one row to use when writing to the dataset.

This finalizes the first version of the project which was then presented to the technical advisor at Volvo for approval. This version can now log all the replayed data of the two signals continuously with the name of the signal as the dataset name. This version does not yet support all possible logging situations where the number of signal changes surpasses 10000 times.

5.2 Further development

The first issue to address with the further development was the usage of calling the open file every time data was to be written. Opening a file is a relatively large process, especially in this case where it will be run many thousand times in a row. This operation will slow down the simulation and should be avoided. The solution was to make the file into a class member variable in the CsvWriter header. This way the file was kept open and could be called from all

the functions within the class CsvWriter. To keep a file open means that data can be written to it and if it is closed data cannot be written to it.

The next important step was to make the datasets extendable. Meaning that the size of the dataset can start off with a smaller size and then extend as the number of signal changes increases. The decision was made to start with a dataset of 100 rows and to increase by 100 rows each time when reaching the end of the dataset. The number of columns will not change, they will remain to be 2. The datasets for each signal will now be of different sizes and the dimensions of each dataset must be tracked. The need for keeping track of the dimensions comes from how the library works. To use the extend function from the library, the method needs an array of the new dimensions. There is no function to simply add 100 rows. The function needs the exact new dimensions, meaning the previous dimensions need to be kept track of to add 100 rows to it and then provide it to the function. Keeping track of the dimensions is also needed to know when to resize the dataset. To understand further why the dimensions are needed in this way see listing 3 which shows the code for extending the dataset. To keep track of the dimensions, they are saved to a class member variable in the signal header file called signal.h. The implemented code uses the index variable from signal.h, which was described in the first version, to check when to resize. When the index reaches the same number as the size of the dimension it is time to resize. With this step done the implementation is fully functional for every possible simulation scenario, but there are still optimizations and helpful improvements that can be done.

```
hsize_t* sigDim = signal->getDim();

//Extend when index reaches size of dataset
if (signal->getIndex() == sigDim[0]) {
    hsize_t dimsext[2] = { 100, 2 };
    hsize_t exsize[2];
    exsize[0] = sigDim[0] + dimsext[0];
    exsize[1] = sigDim[1];
    signalDataSet.extend(exsize);
    signal->setDim(exsize);
}
//Gets resized dataspace
DataSpace extendedDataSpace = signalDataSet.getSpace();

//Select hyperslab to write data to
extendedDataSpace.selectHyperslab(H5S_SELECT_SET,
count, start, stride, block);
signalDataSet.write(data,
PredType::NATIVE_UINT64, space, extendedDataSpace);
signal->incrementIndex();
```

Listing 3: The code for extending a dataset. The code first needs to retrieve the previous dimensions. Then check if it is time to extend. If the index has reached the end of the dataset, the code for extending will be run. The writing to the dataset can then continue with the extended dataspace.

Another functionality that was requested was to record the start time of the simulation in the HDF5 file. This was relatively easy to do with attributes and the time function from the C

time library. As described in section 2.3, an attribute contains metadata that can be used to explain the file or the dataset. The implementation sets a time stamp in the metadata for each dataset which describes the time that the simulation of the signal started. This means that the dataset can have different timestamps depending on when the user starts the simulation of the signal.

When doing the improvement of extending the dataset, the dataset was extended by 100. After resizing it is possible that less than 100 writes will be done before the simulation is finished. There will then be empty rows in the dataset that will be wasting space. There were no examples online and no documentation for making a dataset smaller in C++, only for making a dataset larger. Since the extend method accepted an array of the dimensions it seemed like there was a possibility that it might also work for decreasing the size of the dataset. This idea was then tested in the separate "Hello World" project and proved to be working. With this knowledge the implementation of removing the empty rows could start. This action had to be done when ending the simulation of a signal and when the entire simulation was shut down. This was done in a code piece that is run when manually ending the logging of a signal and in a destructor method that is called when the lifetime of an object ends [25]. To resize the datasets for all signals the code loops through a list of all signals and sets the new dimensions to be the index variable of every signal, see listing 4 for the code for removing the empty rows.

```
hsize_t index = signal->getIndex();
DataSet dataset = signal->getDataset();
hsize_t *dims = signal->getDim();

hsize_t exsize[2];
exsize[0] = index;
exsize[1] = dims[1];
dataset.extend(exsize);
dataset.close();
```

Listing 4: Code for resizing the dataset for a signal to remove empty rows.

5.3 Problems and solutions

One problem encountered during the implementation was an unexpected issue with the library. The latest release of the library was released as a 64-bit library and was tested in a separate project to understand how to use it. However, as it turned out, the simulation where the code had to be implemented was written and must be in 32-bit. The latest releases do not exist as 32-bit libraries, but older releases do. An attempt to fix this issue was made by downloading an older release in 32-bit. The library fortunately worked and had all the functionality necessary for it to work in this project.

Another problem with the library was that it could only be used as a dynamic library and not as a static library. That means the library is loaded to memory at runtime. This way of using the library turned out to not be compatible with the simulation program and would cause the program to crash in debug mode. The quick but not ideal solution to this issue was to copy the

DLL files from the library folder into the working folder of the simulation. This makes it easy for the simulation to find the DLL files.

6. Visualization

To easily be able to control that the output files from the simulation are correct, a visualization program in python was created using the HDF5 python library to read the data. The matplotlib for python was also installed to plot the data.

The first step for the visualization was to start with visualizing the output for just one signal. The next step was to develop it so it can plot all possible signals in the file. To plot the signal first it is necessary to open the file and retrieve the dataset inside it. Using the HDF5 library for reading the dataset it returns an array of the data. This data could then be plotted by using the matplotlib plot function. The result can be seen in figure 6.

The second step was to plot all signals in the file which at this point were only two signals. In order to do this, code had to be written that retrieves all datasets. This was done by using the HDF5 library for reading dataset and then using a loop to access each dataset one by one. To make it easier to compare the shape of the graphs it was decided to not have overlapping plots in the graph. To do this a subplot was used for each signal to which makes it easier to overview the plots when there are many of them that are very similar.

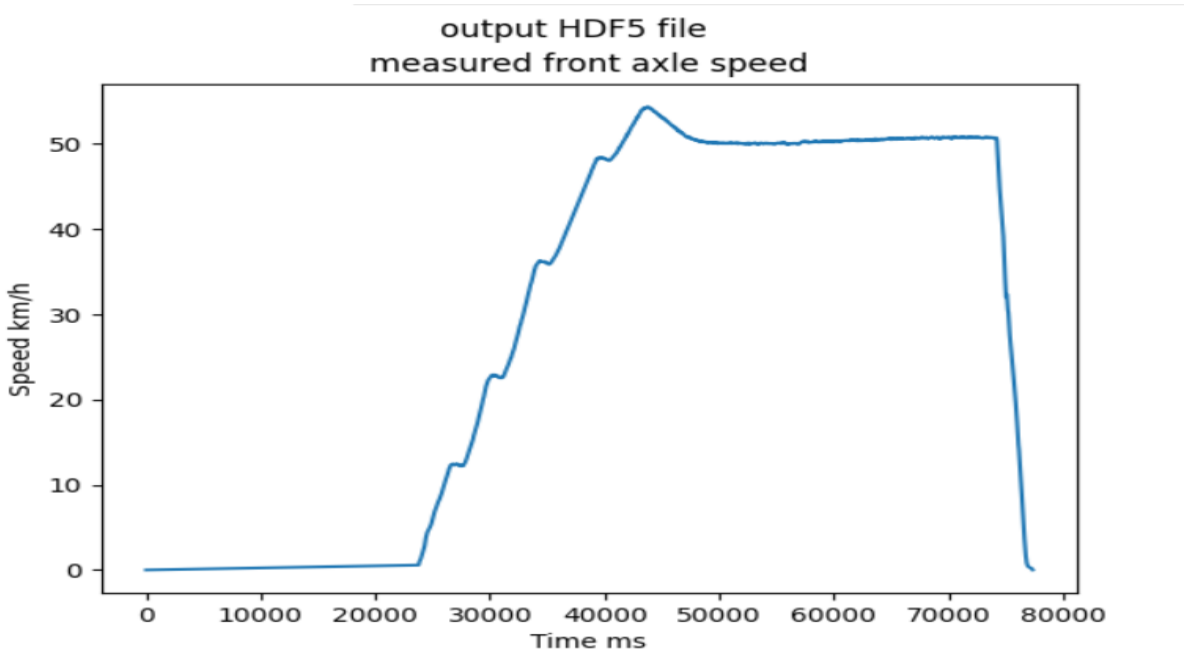


Figure 6 Plot from visualization program of one signal from the HDF5 file

7. Result

The result of the research of file formats for this project was to use the HDF5 file format and library. The result of this projects work is the functionality of saving the simulation log data of type 64 bit unsigned integer in an HDF5 file. The simulation log data of each signal can be saved for any number of signal changes without any empty spaces at the end of the signal's dataset. The data from each signal is saved in a dataset and the dataset name is the name of the signal which makes it easy to find in the file. The result can handle stopping the simulation, then starting to write again and ending the simulation both via the CLI window and by ending the application in the IDE while still saving correctly and saving without empty rows in the dataset.

When starting the simulation, it is possible to decide the name of the file, with the requirement of adding the file ending ".h5" at the end for the file. In order to verify the correct behavior of the simulation and the logging, the SIL team had recorded two signals, the measured front axle speed and the estimated longitudinal speed by the driver assistance module. These recorded signals could be used with the HDF5 code and visualizer to verify expected behavior. The expected behavior was to see that the signals are very similar but not an exact match. These signals are real input data from a truck in an emergency brake test scenario and should show a speed increase reaching a target speed followed by a sudden decrease to zero. As we can see in figure 7 produced by the visualization code, the graphs show the expected behavior and therefore proving that the logging works.

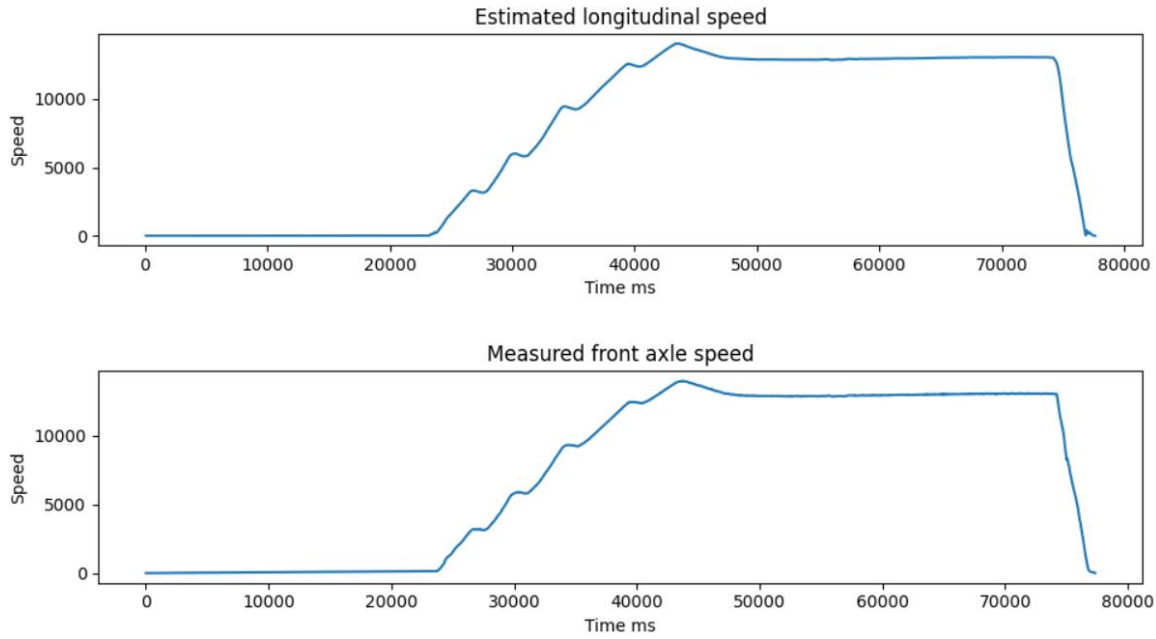


Figure 7 Plot of all signals from the HDF5 file

8. Discussion

This section discusses how the decision was made of choosing the HDF5 library for the project, what could have been done better in the planning of this project and what can be further developed in the future.

8.1 File format decision reasoning

The reason why the MDF4 format was first investigated was because it was a standard specifically created for the automotive industry. This seemed like a perfect choice for this project, but when looking into open-source library's it was no longer the obvious choice. The library was difficult to understand, it lacked in documentation and there were only 15 posts on Stack Overflow about the standard so no help could be expected from there. Since the testing of writing the data types that was needed failed it was obvious that this library would not work for this project.

The MAT-file matio library was more documented with clearer examples. However, the examples lacked examples for how to write the different datatypes. Since there was no time to test the library, there was no way of knowing how user friendly the library is. The library is also dependent on the HDF5 library that was the other investigated library in this project. This library does have more results in Stack Overflow than MDF4, but still only 71, meaning there is some chance of getting help through that platform. Because of the lack of examples, not having tested the usability and because it has a dependency on another library, this library was also deemed not a good match for this project.

The HDF5 library had a lot of examples. While only the basic examples exist in C++, the more advanced examples exist in C and could also be used. The examples showed both how to write the datatypes needed and examples for a lot more of the functionality that was assumed to be needed in this project such as extending a dataset. The function documentation was also very well described, and Stack Overflow had a lot of results that could aid in the project. The test project was successful, and the library was user friendly.

Because of the lacking parts of the other libraries and the unsuccessful tests compared to the extensive documentation and the successful test in HDF5, the HDF5 library was chosen.

8.2 Critical Discussion

When investigating the different file formats, as previously stated, the MAT-file format was not spent as much time on as the other formats. Also, the MDF4 was spent a lot of time on without being successful. The reason for spending less time on the MAT-file was partly poor time management, but also personal bias. When the technical leader introduced this project, it appeared as if the MAT-file format was less interesting to the team than the others. This created a personal bias which led to the format not being given enough time to be investigated in this project. This personal bias should have been ignored and an equal amount of time should have been split between the libraries. This was especially clear when the MAT-format turned out to be better documented than the MDF4 library.

When planning the research part of this project it was decided to spend two weeks per library with MAT-file not being included. This led to an unnecessary large amount of time being spent on a library where it was moving forward very slowly and, in the end, not being successful. A better strategy would have been to lightly try them out simultaneously to

discover quickly which one works better, and then work on trying that one further. This would have resulted in discovering that the MAT-file library was better than MDF4 quicker and would probably would have resulted in there being time left to test that library.

8.3 Future development

For this project, the basic requirements were fulfilled but there were some optional tasks that were left for future improvement. The saving of complex datatypes from the simulation was not completed because of reaching the time limit for this project. Through the research done for HDF5 it is known that this is possible to do

It was decided in this project to do the most basic visualization step instead of doing the visualization that required the help from the DAVA team. The reason this decision was made was also because there was too little time to do this.

In the problems and solution sections it was mentioned that the solutions to the problems were not the best solution. This is also something that could be improved. Instead of downloading an old version of the library to get it in 32-bit, a possible solution could be to download the source code and use that to build it to a 32-bit library with the latest functionality. Another problem was that the library was dynamic, which was solved by copying the dll files into the project. This could also be improved to connect the library in a correct way.

8.4 Ethics and sustainability

Sometime has been spent considering the ethics and sustainable aspects of this project. One aspect for sustainability could be that, because the new log files avoids conversion to plain text, the system will require less power and therefore reduce emissions.

An aspect of ethics could be that we use open-source software which has been provided by individuals for free, but we are using it in a commercial company that keeps all software internal and has no intention of contributing to the community by releasing open-source software. There may be people who consider this to be unethical. However, it is very common that commercial companies use open-source software and there are no regulations against it if the license of the software does not state otherwise.

Both aspects are quite farfetched, so the conclusion is that, applying the aspect of ethics and sustainability to this project is not very relevant because it is such a small part of a program and therefore does not have a lot of impact on either of these aspects.

9. Conclusion

For this project the conclusion is that HDF5 is the best file format and open-source library to use for signal logging. This project managed to create new log files and extend the old simulation logging using HDF5 format and their available libraries in C++, and then visualize the data in python from the generated log files. It was also proven that HDF5 can be read in MATLAB. In this way the purpose of this project was fulfilled because the HDF5 format that was chosen, and its open-source library, has achieved the demanded criteria.

Further developments for this project are to save the complex data types so it will be able save structures. The second development is to improve visualization so instead of building a small visualization in python, it can be done by using an analytics visualization engine written in python.

Reference

[1] AB Volvo, “Organization” . [Online]. Available
https://www.volvogroup.com/en/aboutus/organization.html
[Retrieved: January 05, 2022]

[2] T.Falkman, private communication, November 04, 2021.

[3] Sustainability of Digital Formats, “CSV, Comma Separated Values (RFC 4180)”, November 02, 2021. [Online]. Available
https://www.loc.gov/preservation/digital/formats/fdd/fdd000323.shtml
[Retrieved: January 20, 2022]

[4] Tanya Dalik, “CSV - What, Why and How”, November 07, 2012. [Online]. Available
https://www.shopping-cart-migration.com/must-know-tips/5985-csv-what-why-and-how
[Retrieved: November 01, 2021]

[5] ASAM, “ASAM MDF”. [Online]. Available:
https://www.asam.net/standards/detail/mdf/wiki/
[Retrieved: November 10, 2021]

[6] HDF5 Group, ” Introduction to HDF5”, June 03, 2019. [Online]. Available
https://portal.hdfgroup.org/display/HDF5/Introduction+to+HDF5
[Retrieved: November 02, 2021]

[7] HDF5 Group, ” HDF5 Dataspaces and Partial I/O”. [Online]. Available
http://davis.lbl.gov/Manuals/HDF5-1.8.7/UG/12_Dataspaces.html
[Retrieved: December 02, 2021]

[8] MathWorks, “MAT-File Format”. [Online]. Available
https://www.mathworks.com/help/pdf_doc/matlab/matfile_format.pdf
[Retrieved: January 05, 2022]

[9] ALEX, ” Static and dynamic libraries”, January,23,2020. [Online]. Available
https://www.learncpp.com/cpp-tutorial/a1-static-and-dynamic-libraries/
[Retrieved: November 03, 2021]

[10] Microsoft, “Microsoft visual studio professional”. [Online]. Available
https://visualstudio.microsoft.com/vs/professional/
[Retrieved: November 01, 2021]

[11] Microsoft, “Microsoft Teams”. [Online]. Available
https://www.microsoft.com/sv-se/microsoft-teams/group-chat-software
[Retrieved: November 01, 2021]

[12] Atlassian, “Bitbucket: What is Bitbucket?”, September 23, 2021. [Online]. Available
https://confluence.atlassian.com/confeval/development-tools-evaluator-resources/bitbucket/bitbucket-what-is-bitbucket
[Retrieved: November 01, 2021]

[13]Stack overflow, “Empowering the world to develop technology through collective knowledge”. [Online]. Available
https://stackoverflow.com/company
[Retrieved: January 05, 2022]

[14]Michael Bührer and Bernd Sparrer, “MDF4-LIB”, February 19, 2021. [Online]. Available
https://www.turbolab.de/mdf_libf.htm
[Retrieved: January 05, 2022]

[15]MathWorks, “read”. [Online]. Available
https://se.mathworks.com/help/vnt/ug/read.html
[Retrieved: January 05, 2022]

[16]HDF5 Group, “Download HDF5”. [Online]. Available
https://www.hdfgroup.org/downloads/hdf5
[Retrieved: January 05, 2022]

[17] HDF5 Group, “Examples from learning the basics”, October 22, 2020. [Online]. Available
https://portal.hdfgroup.org/display/HDF5/Examples+from+Learning+the+Basics
[Retrieved: January 05, 2022]

[18] HDF5 Group, “Examples by API”, May 22, 2019. [Online]. Available
https://portal.hdfgroup.org/display/HDF5/Examples+by+API
[Retrieved: January 05, 2022]

[19] HDF5 Group, “HDF5 C++ API”. [Online]. Available
https://support.hdfgroup.org/HDF5/doc/cpplus_RM/annotated.html
[Retrieved: January 05, 2022]

[20] MathWorks, “HDF5 Files”. [Online]. Available
https://se.mathworks.com/help/matlab/hdf5-files.html
[Retrieved: December 10, 2021]

[21] HDF Group, “HDF5 MATLAB EXAMPLES”, November, 04, 2016. [Online]. Available
https://support.hdfgroup.org/HDF5/examples/api18-m.html
[Retrieved: December 10, 2021]

[22] MathWorks, “Create MAT-File in c or c++”. [Online]. Available
https://se.mathworks.com/help/matlab/matlab_external/creating-a-mat-file-in-c.html
[Retrieved: January 05, 2022]

[23] Matio, Version 1.5.21, [Software], tbeu,2021.

[24] Matio-cpp , Version 0.1.1, [Software], ami-iit,2021.

[25] Microsoft, ”Destructors(C++)”, March 08,2021. [Online]. Available
https://docs.microsoft.com/en-us/cpp/cpp/destructors-cpp?view=msvc-170
[Retrieved: January 05, 2022]