

Contents

[1 Introduction](#_30j0zll)

[1.1 Project statement](#_1fob9te)

[1.2 Purpose](#_3znysh7)

[1.3 Goals](#_2et92p0)

[2 Deliverables](#_tyjcwt)

[3 Design](#_3dy6vkm)

[3.1 System specifications](#_1t3h5sf)

[*3.1.1 Non-functional*](#_4d34og8)

[*3.1.2 Functional*](#_2s8eyo1)

[3.2 PROPOSED DESIGN/METHOD](#_17dp8vu)

[3.3 DESIGN ANALYSIS](#_3rdcrjn)

[4 Testing/Development](#_26in1rg)

[4.1 INTERFACE specifications](#_lnxbz9)

[4.2 Hardware/software](#_35nkun2)

[4.2 Process](#_1ksv4uv)

[5 Results](#_44sinio)

5.1 Results:Engineering Analysis

[6 Conclusions](#_2jxsxqh)

[7 References](#_z337ya)

[8 Appendices](#_3j2qqm3)

# 1 Introduction

## 1.1 Project statement

The primary objective of this project is to create a Matlab mex function that processes SCADA binary files into Matlab directly. The client has provided the team with an extension ldsf file that is created by a software called Liege Measurement System. The binary file contains measured data that our client is needing for engineering analysis. The process may takes hours depending on the file’s size and each file has to be dealt with separately.

## 1.2 Purpose

 The purpose is to make the process of the conversion of binary files to Matlab files more efficient. The binary files are generated from the Liege Measurement System at Honeywell. The binary files are then transformed into Matlab using a proprietary push process by the client. Once in the .mat file it can be loaded into Matlab to run analysis on the file.

## 1.3 Goals

 The goal is to write a complete Matlab function that would take the ldsf files, and load them directly without having to use the measurement system and push process to convert the input data into a Matlab readable file. A more direct process will shorten the time spent on converting the files so that they are ready to analyze by our client.

# 2 Deliverables

 We feel a fulfilled project consists of a runnable Matlab function with:

1. Associated C++ source codes, so they can be edited for user preferences
2. Sample output from the function based on the example outputs the team was given, as engineers we are able to run an analysis (this should be identical to the sample output Honeywell has provided us)

With the perfected process, the team will demonstrate a presentation of our program. In this presentation, we plan to highlighting all of the methods utilized and overall showing that our program is truly a solution to Honeywell’s time constraint issue. There more than likely will be additional deliverables as the project progresses, including details we feel need some form of concentration or details Honeywell itself wants to know more about. However, as engineers, we will react and respond accordingly.

# 3 Design

##  3.1 System specifications

 As denoted within the initial specifications, the underlying basis for this new system is *speed*. Specifically, Honeywell desires a reduction in process time by up to 20 or 30 percent, depending on the size of the .ldsf file. This is why there is a direct link planned via the .mat function. It would provide a connection from the Matlab software to the C++ scripts of Visual Studio, allowing the existing file conversion to continue like normal. However, because of the direct link, the need for the text file would be eliminated. After all, there is little point in building a unique formatted file just for it to be reinterpreted and parsed by the same system. In avoiding the .txt output, it is believed this would cut enough “excess processing” to qualify for the percentage reduction.

### 3.1.1 Non-functional

 In following with the traits of any coding project, it is important to enhance both the old and the new structure of the system to fully optimize the overall output as a whole. This means that:

* The existing C++ code provided must be able to compile and link to Matlab. This can be done by constructing and utilizing the built-in Mex functionality, which acts as a sort of coding gateway, allowing Matlab to access and use different coding platforms (including C++).
* Along with this idea, the Mex call should be able to apply the necessary conditions to convert the .ldsf figures into the desired Matlab output. This serves as the main “speed up” procedure and therefore the main focus of the team’s efforts when coding.
* Matlab then submits to the user the formatted .mat data along with any figures and charts Honeywell sees essential.

### 3.1.2 Functional

Applying these principles of how the system will work, a primitive procedural layout can be constructed as to what the new project must do in order to :

1. A call in Matlab is made to begin the process of converting the LMS SCADA DAQ binary file. More specifically, through use of Mex functionality, Matlab calls Visual Studio to let it know to begin the process of converting from .ldsf to .mat
2. The .mat file must return to Matlab
3. Once there, the .mat file is enhanced to express the data found. This will be expressed in the form of a graph indicates the positioning of the accelerometer figures.

## 3.2 PROPOSED DESIGN/METHOD

 Completing this project would be adding onto the client's current Visual Studio system, a code whose computational structure is laid out as such:



In applying the strategies discussed, the proposed changes made would allow a user to call a function in Matlab and directly convert a binary .ldsf file into an complex .mat array. Further analysis of the data can then begin.



##

## 3.3 DESIGN ANALYSIS

 Though the procedure sounds rather restated at this point, the fact of the matter is that the infrastructure of this design is rather simplistic in nature. Notably, the team requires a Mex call to the initial C++ code. This allows for the proper transference of .ldsf figures into comparable .mat data. Once obtained, the associated numerics of the array can be assorted into a visual representation.

 Though easy to comprehend, the task of developing this project is rather complex. The Honeywell code in its initial state, for instance, does not build properly, disallowing any changes to be made to the software until its rectification is satisfied. Being that the team is relatively unexposed to this type of environment, considerable time will undoubtedly be spent here. The actual linkage of Matlab to Visual Studio via Mex connection, likewise, is also of unfamiliar territory, thus requiring even more research into this realm as well.

# 4 Testing/Development

## 4.1 INTERFACE specifications

 This project is solely software based. We integrated binary .ldsf files into Matlab. As a team, we closely followed the procedure that Honeywell follows. We transferred and converted the files provided to us.

##  4.2 Hardware/software

 We used a variety of software throughout the last two semesters. Matlab and Visual Studio were used to test our newly created function. Since Matlab is already used by our client we were assured that the expected data output and ours will be the same. In Matlab, we will be using the Matlab Mex functions to do the data conversion portion of this project; thus, this project does require us to use the C++ background softwares. Currently, we are using the Microsoft Visual Studio to study and compile the sample C++ script given by Honeywell.

## 4.3 Process

 We would just use the C++ script compiler such as C++ Shell or Dev-C++ to test out our codes and get the desired output of the data conversion from the SCADA binary file to .txt file. After we have the C++ project working, we will create the Matlab mex function and make any adjustment and changes needed. The Mex functions would execute the C++ script associated for the data conversion so it can convert the ldsf file into Matlab workspace directly. So, the testing process is very similar to our proposed process.



 While constantly keeping in touch with Honeywell on obtaining the output from the project provided to us, we tried to create a prototype which would compile the C++ codes. We created and generated the Matlab Mex function. Once we successfully got this prototype to work, we can easily implement the Matlab MEX function directly when we have the Honeywell C++ codes to build properly. The prototype is simple, we have to generate the Matlab Mex function from the C++ project that is successfully compiled in Microsoft Visual Studio. Below is the C++ code that we used as our prototype:



The C++ code above is not the C++ provided by the client. The code consists of a required built-in mex function to successfully generate the mex function. Every C++ code has a “main” function that is called everytime the code is ran. To generate a Matlab Mex function, we use “mexFunction” instead of main function. In Line 17 in the code above, it takes input from the user and sets ‘num’ which is our input mxArray to the user input. There are four arguments that come with the use of the mexFunction:

* int nlhs = Number of expected output mxArrays
* mxArray \*plhs[] = Array of pointers to the expected output mxArrays
* int nrhs = Number of input mxArrays
* const mxArray \*prhs[] = Array of pointers to the input mxArrays

mxArray is special data that is used to pass data to and from the built-in function in the mex header file or library to perform operation in Matlab.

mxCreateString is initialized the mxArray to the string (str) argument.

mexCallMATLAB is used to call Matlab function, mex file or user-defined function to do something with the input and output of the mxArray. It’s called by

mexCallMATLAB(int nlhs, mxArray \*plhs[], int nrhs,
 mxArray \*prhs[], const char \*functionName);

mxGetScalar is used to get the scalar number from mxArray to other fundamental types in C++.

mexPrintf is used to print something which is similar to the ‘printf’ that is frequently used in C++ code.

A side note, you are able to free the mxArray and reuse it by calling the mxDestroyArray.The team noticed that some C++ lines are unrecognizable by the mex compiler. We tried “using namespace std”, “cin” and “cout” lines, but when we tried to generate the mex function in the Matlab background the lines mentioned were not working even though the generated Mex function was in the output directory.

Personalized settings are required to be set in Visual Studio for this project application. Be aware that these settings may vary depending on the Microsoft Visual Studio version. Below are the settings we used to generate the Matlab Mex function:


After the settings have been specified, click “Build” to generate the Matlab mex function in Visual Studio!



The Matlab mex function will be generated in the output directory that is specified in the project properties in Visual Studio. To call the function, type the project name before .mexw64 in Matlab.



# 5 Results

The entire time working on this project, we ran into a number of issues with the original C++ code that Honeywell provided to us. While they were originally a major stumbling block for us we eventually worked through some of the errors with the assistance of Dr. Zambreno. This mostly consisted of relocating the library files into a new folder where the C++ code was able to find them. Even though we could never build the C++ code we still were able to complete the majority of work required for this project.

In completion of this project, we have constructed a prototype and successfully generated the Matlab mex function from C++ code. This proves that we understand the fundamental concept of this project. In the future we would be able to convert and made changes to the client’s C++ code to generate the mex function they were wanting.

Using our constructed prototype, we have found a way to measure efficiency of running a function in Matlab versus running the Matlab mex function. We performed the following calculations and observations to see how much more efficient the mex function would be for the client’s process:

Team Dec1604 Sample Code Comparison:

|  |  |  |
| --- | --- | --- |
|  | **Size of File (KB)** | **Time (Seconds)** |
| **Matlab** | .291  | 9.123\*10-6 |
| **Mex Function** | 13.5 | 1.5396\*10-5 |

Project Code Comparison:

|  |  |  |
| --- | --- | --- |
|  | Size of File (KB) | Time (Seconds) |
| Matlab | 311,115 | 9.75 |
| Mex Function | 311,115 | 0.035 |

**Matlab Calculation:**

31.35$μ$s/1KB

311,115 \* 31.35$μ$s = 9.75 seconds

**Mex Calculation:**

114ns/1KB

311,115 \* 114ns = 0.035 seconds

 In conclusion, the sample code we used to convert the C++ code into mex gave us a rough idea of the computing time of Matlab versus the mex function. Assuming a linear system, we calculated how efficient the mex function process would be on the client’s system. The file size provided to the team was 311,115KB and therefore would make the mex function process more efficient than the current process.

# 5.1 Results: Engineering Analysis

 As part of the files we were originally provided we also received a set of data in MATLAB that should match the output of the ldsf binary file the program is meant to use as an input. Once we had completed the project we intended to do a series of analysis on the data in order to gain a better understanding of what we were working with. Through talking with the client we found that the data we provided was a set of random data generated by an accelerometer, but as a demonstration of the data we chose to run some analysis on it anyway. The data provided was allocated in sets of acceleration data in the X, Y, and Z directions as shown below in the Workspace figure. Each of these was further broken down into the respective component represented as a Y variable and the time it was recorded over. The following code was designed to reconfigure the time data into an array for comparison against its respective Y variable. Here the data used was from the n\_accel3 struct.



When plotted against the acceleration data within the n\_accel3 struct, the following graph is produced:

One can see from the graph that the sensor recorded being moved back and forth from its maximum to its minimum reading for the span of 25 seconds. The graphs of the other testing outputs were similar, but held some differences in the magnitude of the acceleration, indicating that different accelerometers were used for each test instead of simply one accelerometer being used across multiple instances.

# 6 Conclusions

In outcome of this project, as a team we progressed over the last two semesters regardless of the obstacles we faced along the way. We were not only able to learn a new programming language, but learn different applications of software we had previously been taught in other classes. The project goal was to create a more efficient process of directly reading a large binary file into Matlab. As a team, we were able to understand and utilize Matlab mex functions to complete the goal. Although we could never successfully build the project given to us by the client, we still created a process that could be implemented by the client. We created a process that is more efficient and that would minimize file computing and transforming times. We were able to each bring a diverse range of skillsets to complete the Binary LMS File Interpretation project.

# 7 References

1. Online Source

1. [www.cplusplus.com](http://www.cplusplus.com). C++ programming learning
2. [www.mathwork.com](http://www.mathwork.com). Understanding MATLAB Mex, Mex file creation API and etc.
3. <https://source.ece.iastate.edu/projects/dec1604/>. The link to the svn source website where all of our C++ codes/scripts will be uploaded and shared.

2. All other photos or information has been captured by the team as we have completed the project.

# 8 Appendices

All we have is C++ code and they are not compiled successfully right now.