Vulcan

Digital Engineering Software Scope

Jacob Lehmer

2020-04-16

MOD:2020-04-20

MOD:2020-05-05

MOD:2020-09-30

MOD:2021-03-23

V5

# Introduction

This document serves as the initial scope for the Vulcan Digital Engineering software solution that will be developed. This is a living document and will be modified to reflect any changes that need to be made to keep the project operational. Changes required to facilitate proper functionality in the software may not be represented in this document in any form. Diagrams in this document serve to clarify and share Ideas, these diagrams are not formal code descriptions. Functional requirements may be changed, added, or removed without formal review.

# Functional Requirements

This software is meant as a solution to facilitate the data sharing across the research community. To accomplish this task this software will perform data format translation from any form to any other. Secondary features that shall be included will be the ability to graph the data as well as perform bulk manipulation of the data system. The data from the files will not be read into memory unless required, the data will be read when needed, this allows for manipulation of the data without using system memory for storing data that is not being used. Data formats, manipulation techniques, and graphing features not originally included in the software can be added through a .dll plugin system.

The software to be developed for the Vulcan Digital Engineering project shall meet the following requirements.

* Recursive Directory Searching
* Data Translation Formats
  + TDMS
  + Matlab Files
    - All Versions
  + Data Exchange Format
* Network Communication
  + Deep Lynx Uploading/Downloading
  + SQL Uploading/Downloading
* Data Graphing
  + Fully Editable Graphing Output
  + Savable image format
* Bulk Data Manipulation
  + Fourier Transform
    - Heterodyning
    - Zero Padding
  + Scripting system
  + Plugins for other custom algorithms

## System Requirements

* Cross Platform
* Planned cross-platform support
* Backend written in C++
  + Frontend written in Javascript
  + Electron Frontend

# Diagramming

## Macro Component Diagram

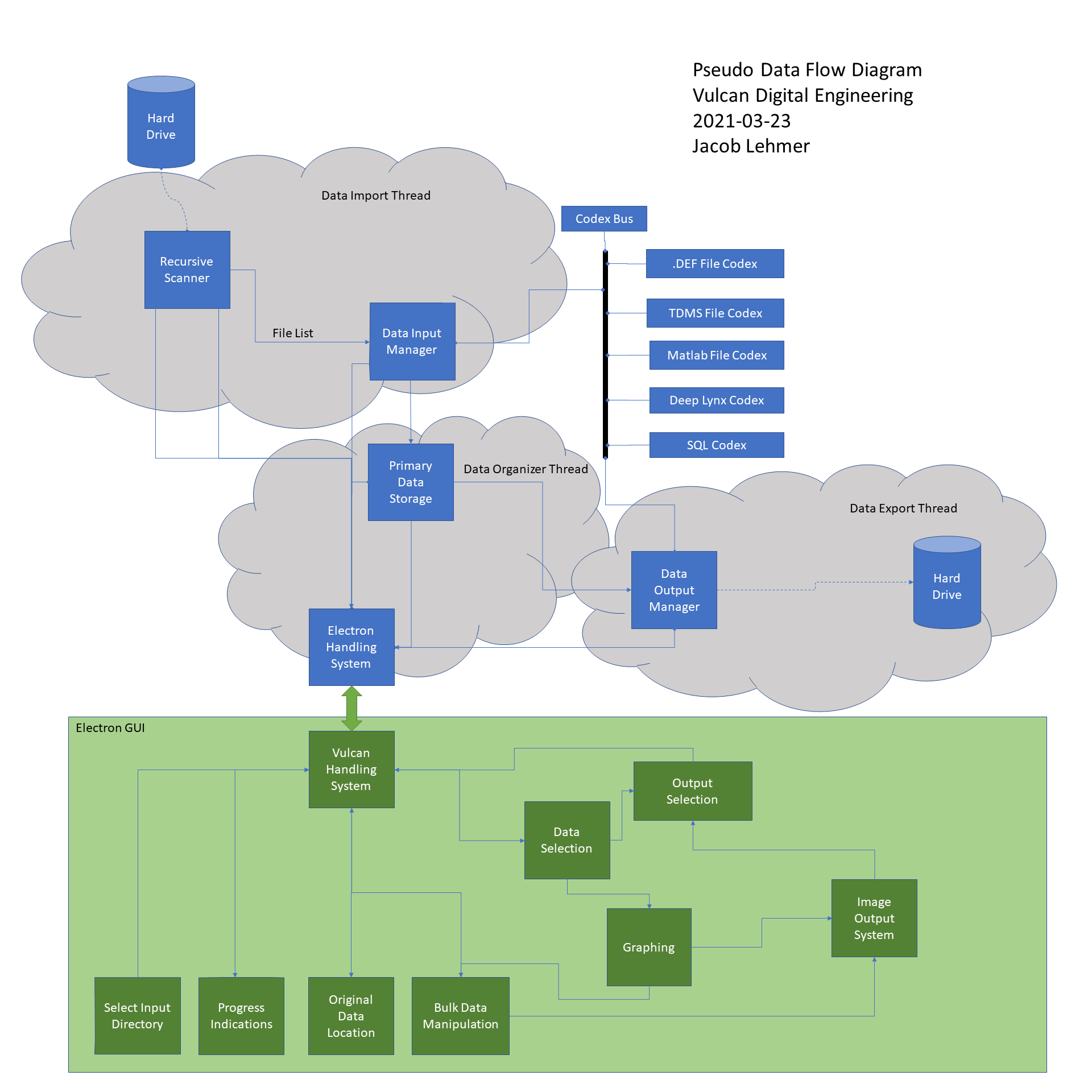


Figure : Pseudo Data Flow Diagram

## GUI Diagram

The following diagrams show a series of mockups for the planned GUI for the system

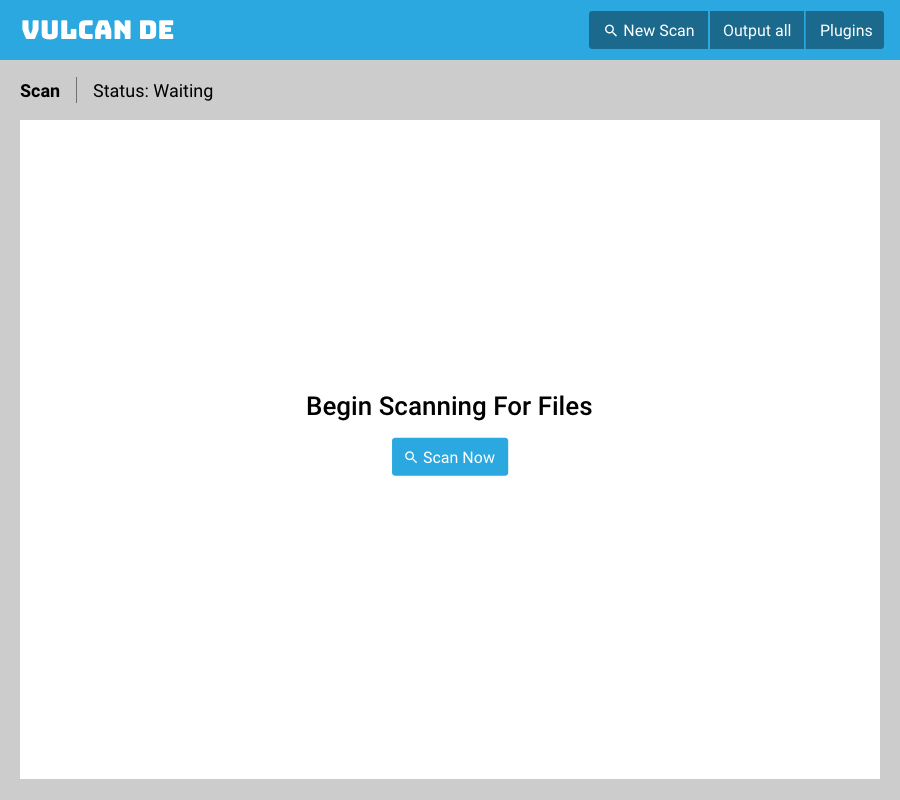


Figure : Initial Start Screen

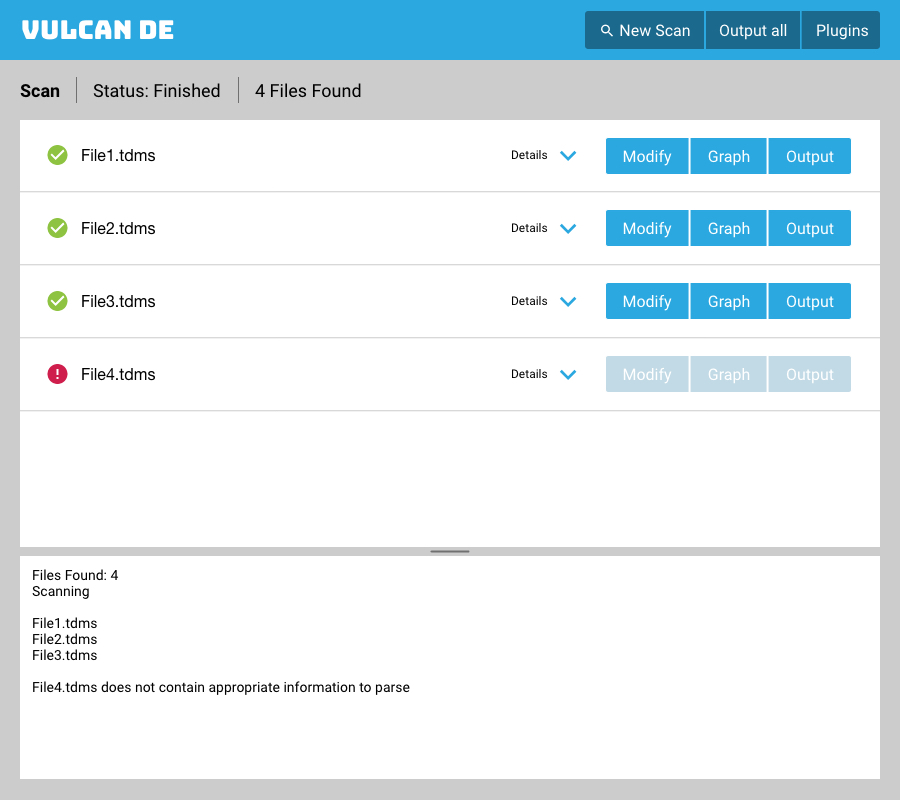


Figure : Screen After Running Scan

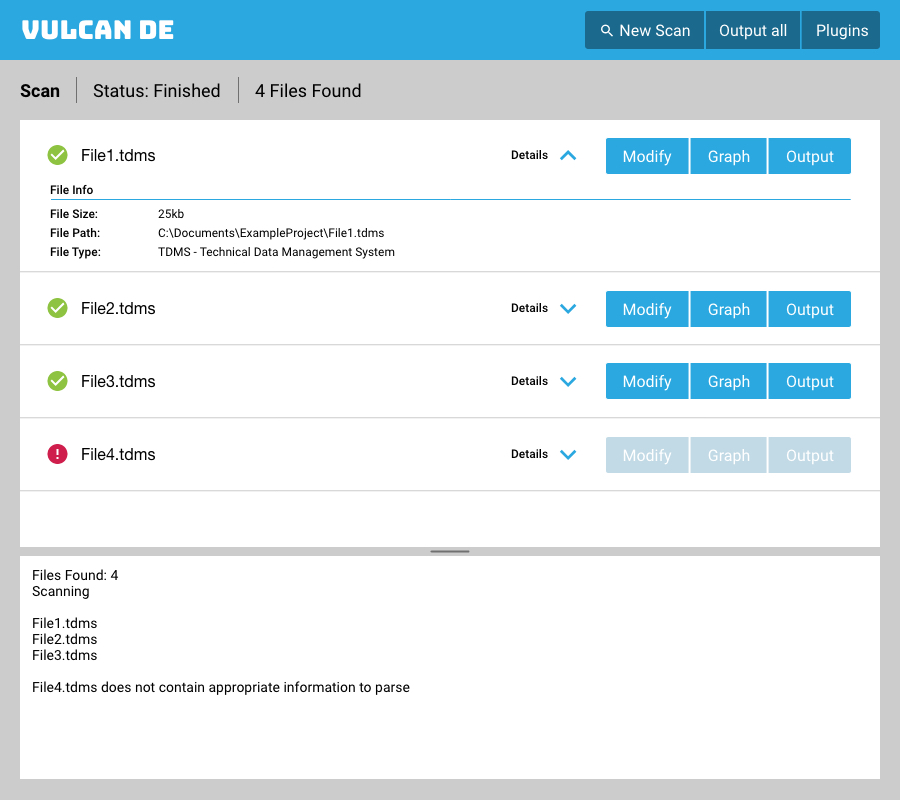


Figure : Screen After Running Scan Details

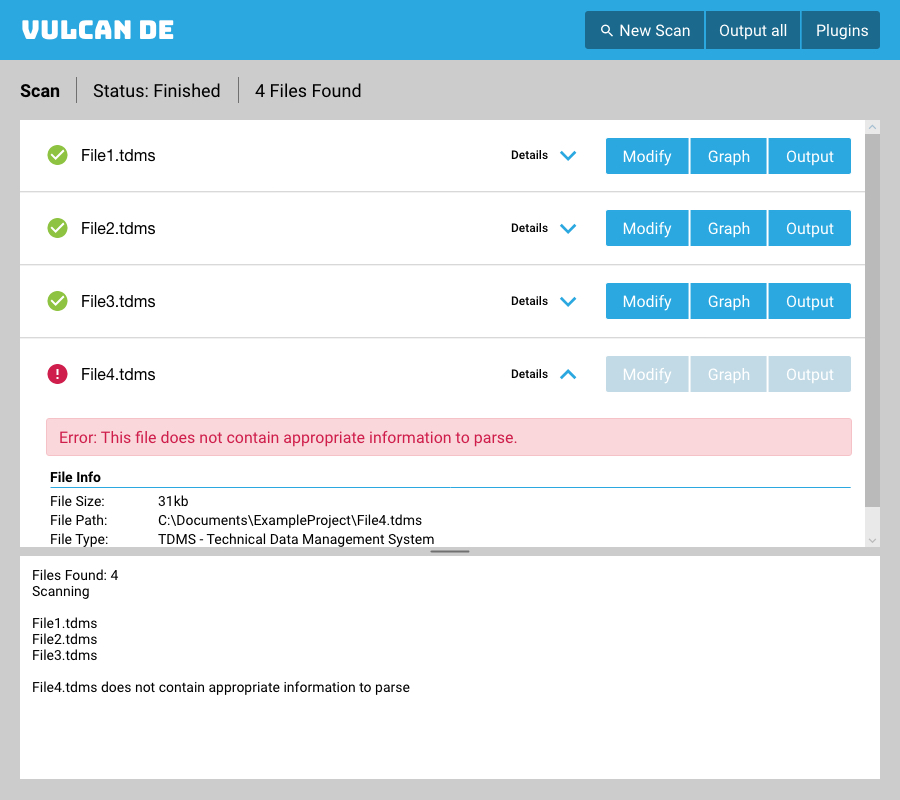


Figure : Screen After Running Scan Error

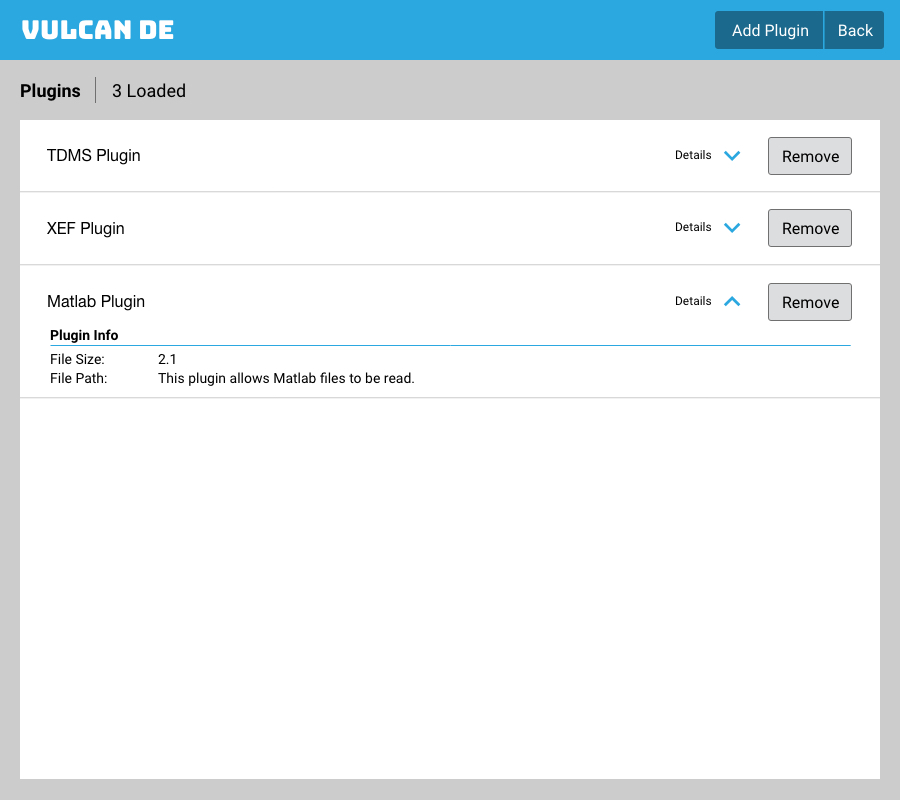


Figure : Data Handling Plugin System

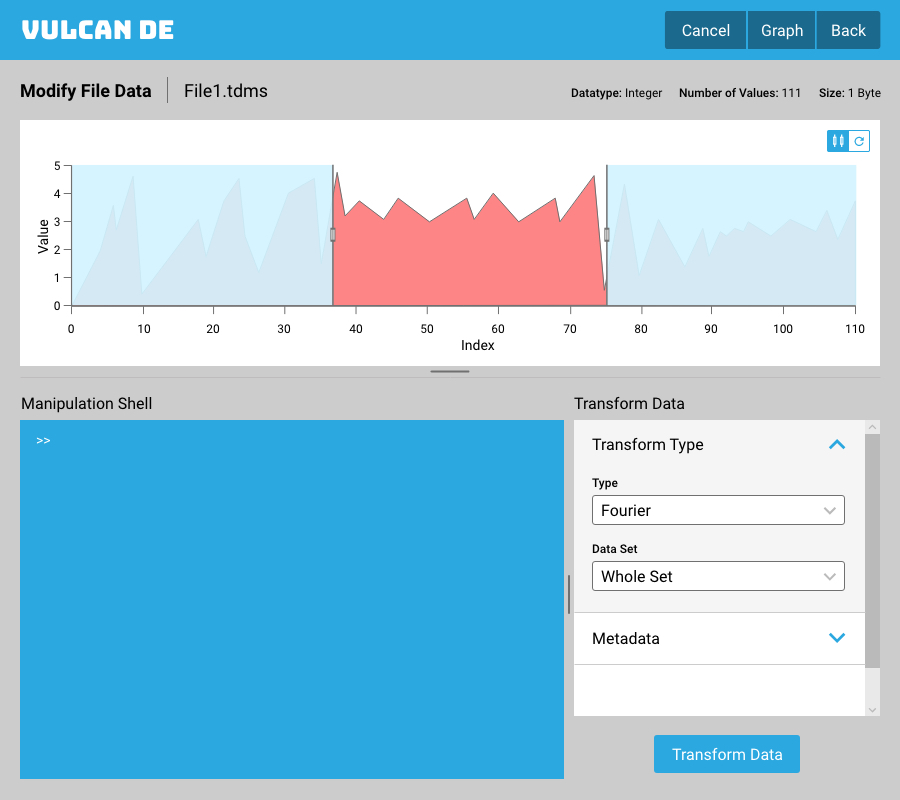


Figure : Bulk Data Manipulation Screen

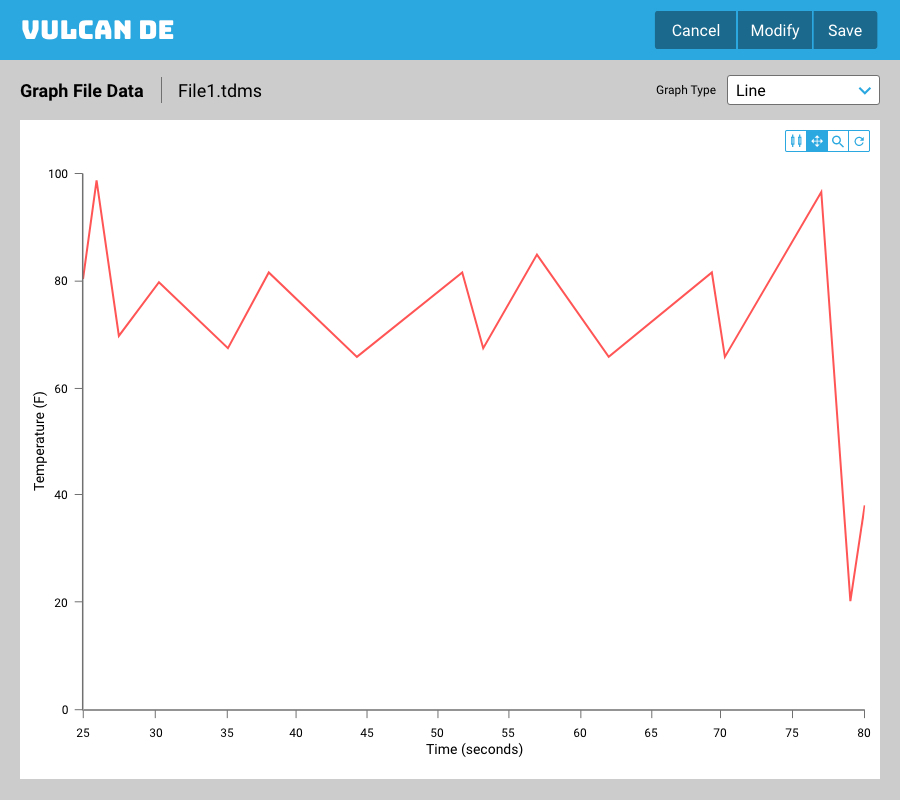


Figure : Data Graphing Screen

# Data Exchange Format

A new data format has been developed for the purpose of data interchange. This format will serve as the tools default data format. The name for this format is the data exchange format.

## Tenants

The format has been developed to meet the following design tenants

* Easy to read
* Data Dense
* Extensible

## Components

In order to accommodate the requirement for an easy to read while still being data dense file the decision was made to separate the meta data information and the data information. The metadata is stored in one of currently two formats, .xml or .json. The extended identifier for these files is .xdef and .jdef in order to differentiate between the multitude of other .xml and .json files that are in current usage. These file extensions are not necessary but do serve as an immediate way to determine what the form of the data is. The binary file contains data. The structure section of this document goes into further detail as to how the two components are related.

## Structure

An agnostic description of tags is below. These tags are the same, with the same capitalization for both .json and .xml. The differences in reading and writing .xml and .json do not have any bearing on the functionality of the file and will not be covered here. The tags can occur in any order within the document but nesting needs to be maintained.

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | Parent Tag | Acceptable values | Description |
| MetaData |  | List of DataDescription Elements | Defines main structure of file. Needs to be the first tag in file for file to be recognized as def file |
| DataDescription | MetaData | Required DataDescription sub elements | Defines how to read and where to find actual binary data file |
| DataType | DataDescription | “Int”,”Real”,”Char” “Unsigned Int” | Defines the type of data to be read |
| BytesPerElem | DataDescription | Integer value representing number of bytes | Defines the number of 8-bit bytes that represents an individual data element in the binary data |
| Endianness | DataDescription | “Big”,”Little” | Represents the endianness of the data elements |
| NumberOfElements | DataDescription | Integer value representing the number of elements in the data channel. | Number of values in the data channel.  If the file is streamed this represents the channel chunk size, if the file is not streamed the value represents the total number of values |
| DataFilePath | DataDescription | Relative or absolute file path to the actual binary data file | Defines where the binary file for the data processing is. The relative file path is in relation to the meta data file. |
| Interleaved | DataDescription | “Yes”,”No” | [NOT REQUIRED]  [DEFAULT: “No”]  Indicates if the data defined by this meta data channel is interleaved in the binary data |
| ChunkOffset | DataDescription | Integer representing the number of bytes into a chunk the data can be found | [REQUIRED IFF Interleaved is “Yes”]  Indicates the number of bytes into a new chunk that this data channel can be found |
| ChunkSize | DataDescription | Integer representing the number of bytes in a chunk | [REQUIRED IFF Interleaved is “Yes”]  Indicates the number of bytes in a chunk in total. |
| ID | MetaData | String | [NOT REQUIRED]  [DEFAULT: metadata filename + #of entry]  This will give the ID of the current data channel, displayed. |
| Anything else | MetaData | Any | Any additional tag can be added, these tags are viewable, and will be written out in the system but do not have any intrinsic value. |

Table : Meta data tags for .def files

The binary files are references from the “DataFilePath” tag in the meta data. One meta data file can contain many entries, each pointing to a different file. Or the meta data can have multiple entries pointing to a single file, in which case the file would be an interleaved file. Or any combination therein. It is required that the meta data accurately define the structure of the data. The binary file only contains a stream of binary data. The file extension of the binary file is completely irrelevant and can be omitted. The following diagram shows the relation of the different meta data elements in the binary file.

A screenshot of a cell phone

Description automatically generated

Figure : Binary Data Format

# Feature Set Addition Order

Development and prototypes will follow this order.

1. Recursive directory scanning prototype
   1. Command Line interface
   2. Interim File Codex
   3. Matlab Codex
   4. TDMS Codex
2. Codex plugin system
   1. Deep Lynx Codex
   2. SQL Codex
3. Make all the documentation
   1. Make the plugin system documentation
4. Cross platform support
5. Test on real world data
6. Electron Integration refactoring
7. GUI addition
   1. Progress indicators
   2. Data selection
   3. Data Outputting
8. Graphing
   1. Graph modification
   2. Image output
9. Bulk Manipulation
   1. Fourier Transform
      1. Heterodyning
      2. Zero Padding
   2. Scripting
10. Bulk manipulation algorithm plugin system

# Command Line Tags

The format for a command line tag is as follows:

-TAG Argument

The dash indicates the start of a new tag.

Arguments do allow spaces, however if spaces and dashes are used the argument should be surrounded by quotations to prevent confusion

These command line tags allow

|  |  |
| --- | --- |
| Tag | Expected Values |
| -InputDirectory | File path that points to the stored location of the data to be input |
| -OutputDirectory | File path that points to the location to transfer the data to |
| -InputType | A string identifier indicating the data type of the input data |
| -OutputType | A string identifier indicating the data type of the output data |
| -PluginDirectory | A string identifier indicating the location that the plugins for the system are located |
| -PrintCodexes | A command line tag which will cause the system to exit immediately after reading the plugin directory and printing all of the codexes that are available to use |