2017

CyberSecurity

SYSTEM INFORMATION AND EVENTS MANAGEMENT

LUKE MCCANN - U1364096

HUDDERSFIELD UNIVERSITY | Tutor Simon Parkinson

Contents

Introduction

This report contains a detailed specification and implementation notes for the SIEM application project. Many Information Technology systems record security information and events in log files, in this project the aim is to create an application in which ca be used to analyse and make sense of the data within these log files.

This project uses the software architecture 'C#'. The decision to use C# was made as it contains many libraries for handling system data and makes reading in large files much more efficient with a lot less coding than that of languages like Java or C++. due to the project being heavily based on managing volumes of data this seemed the logical choice.

Development

During the development of this project many challenges have been faced, all of which have been documented in a table in "implementation notes". Many of these challenges revolved around coding items which have not been used before, learning to use new libraries and generally fixing logical errors in the analytical methods.

Previous Experiences

Before this project, I had used very basic C# in Games Development, mostly writing basic scripts to provide character actions in Unity, however chose to use the language after significant research into the libraries it offers for system and data analytics based applications, atop of this I have mostly used Java in the past and knew that C# syntax is very closely related to that of Java.

I have had no previous experience of developing GUI applications in Visual Studio, however after minimal research decided on this as it seemed the more appropriate action for the type of application the project is based on, and I wanted to make data easily readable to the user.

Although I have no previous experience in developing applications of this type (Security and Information Management), I developed a detailed plan on how to tackle the main issues, including a sheet of which to identify the key pieces of data a security analyst would likely have interest in, allowing the user to sort the data in an order that they would like to see.

Specification

As a completed application the following specification details the application in its current standing, this is subject to change in the future should development continue;

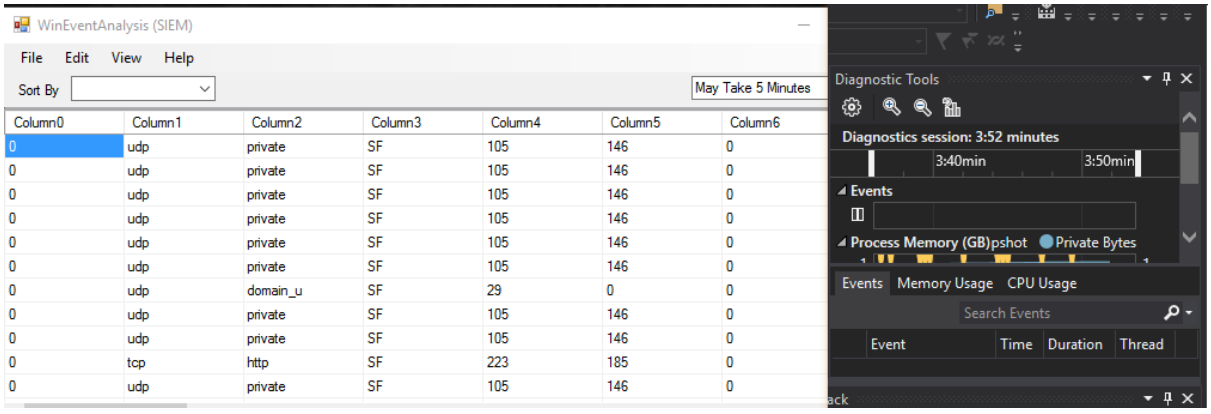
Functionality

1. Loading Csv Data

The key functionality for this application is the ability to load in Csv files, without this the application would be completely useless as it would be unable to make sense of files that it cannot load.
As it currently stands the application can load in Csv files with varying times depending on the size of the file which is being loaded, and the other processes running at the time in memory.

When loading smaller Csv files (below 1000 rows) the application is instantaneous and provides a MessageBox telling the user large files may take up to 5minutes to process. Larger files can take significantly longer, however this was expected and is one of the main challenges of working with SIEM applications (see “Implementation Notes” and “Analysis”).

As of this version, the average time to load large files is 3minutes 53seconds.

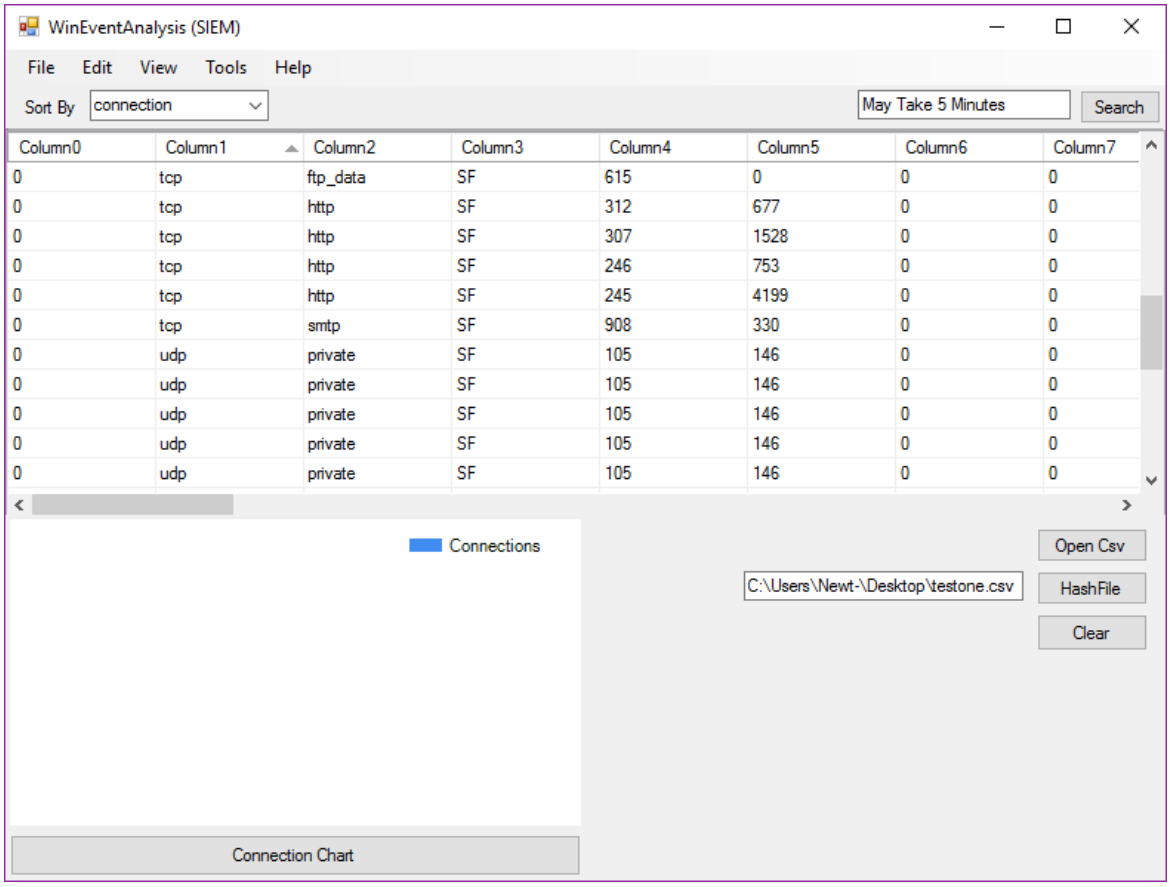


2. “Categorisation” (Sorting)

A useful ability in SIEM is the ability to “Sort” or “Categorise” data. This has been another challenge in the development of the application, however I came up with two methods for users to sort data, as it stands however the secondary method does not work on files over 120MB as it holds the files in memory.

1. ComboBox Sort

This method works on all files, it allows the users to sort the DataGridView by Connection, ConnectionType and more. This is an ascending order sort which groups the files by category, the user may scroll down to each categorisation to inspect the details further.



2. Search Sort

This is an additional feature added more recently. The idea behind this implementation is that the user can search for rows containing specific “string” values. Although this method works well on small to medium sized files, there is an issue with larger files where this function causes an “OutOfMemoryException” due to the amount of data being processed all it once, it was trialled to use threads to combat this, however the same error occurred due to the threads using up the memory rather than the process. Another trialled solution was to use macros to increase the amount of allocated memory at runtime, this sped up the process significantly, however larger files still cause a “OurOfMemoryException”. The third fix trialled for this function was to use Garbage Collection along with limiting the data the process can use (slowing down the process) in an attempt to improve performance, however to date this issue is unsolved for the larger files.

In the future a fix may be issued for this issue, however for now the Csv cannot be processed while it is being held in memory by DataGridView.

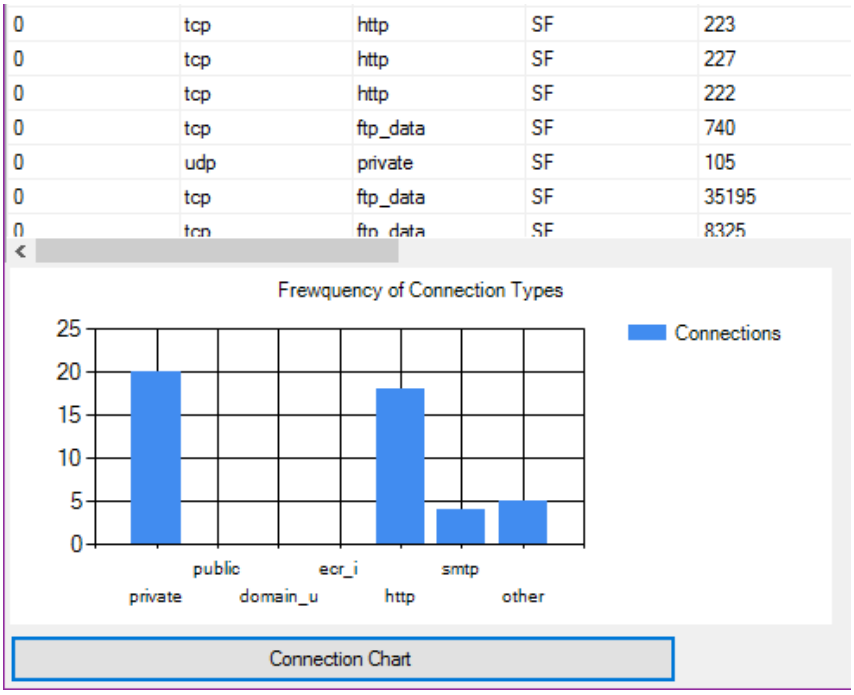
A possible solution to this issue could be to load the file line by line into the DataGridView, this could be done using foreach loops to iterate through the columns and set each cell to the value before the delimiter, however as it currently stands the DataGridView is updated from DataSource, this means that the updates to DataGridView are currently instant and it automatically updates with changes to its source, the performance is affected when changing this and a workaround solution would need to be made, a final trial was done trying to limit memory use using MemoryFailPoint, however this had too greater impact on performance to be viable.

3. Frequency Of information

1. BarChart

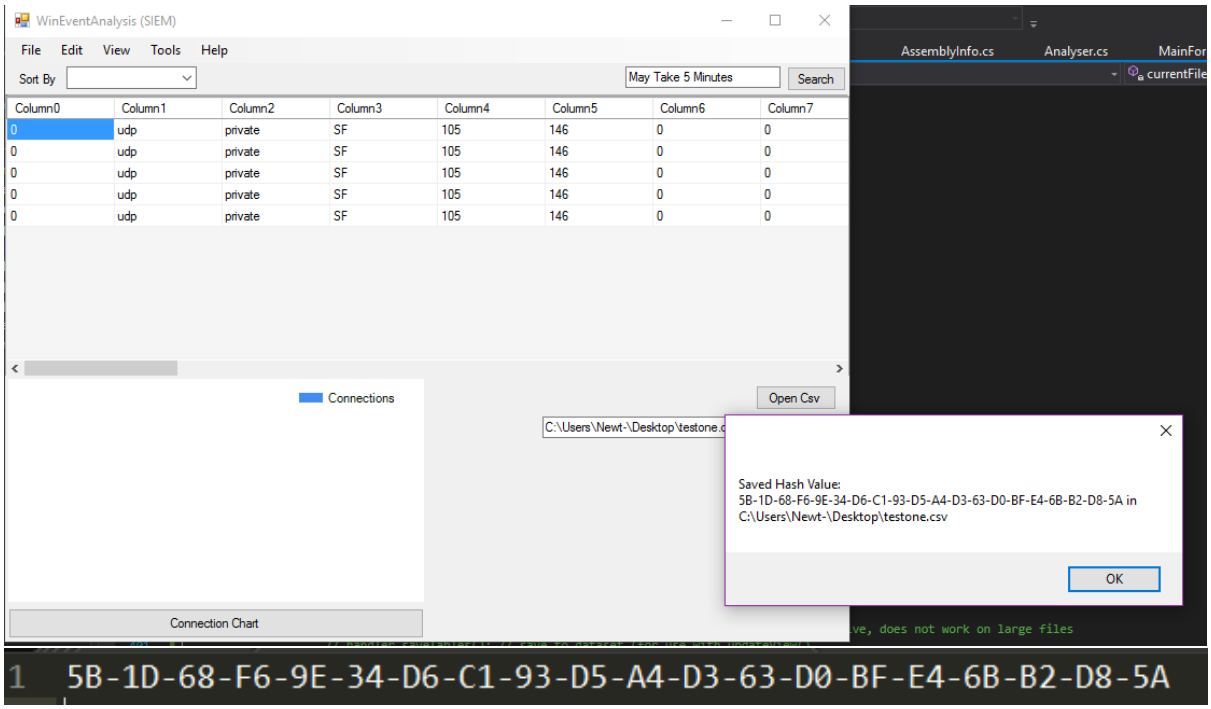
Deducing the frequency at which information appears is a priority in this application. This is important for diagnostics as one odd connection alone can be meaningless, however identifying patterns in connectivity can be extremely useful for not only seeing what is going on with the machine, but also finding issues with the machine, malicious connections, files or items that just don't appear to be "normal". This is also important in businesses as it allows the business to see which users are the most frequent, what the most frequent settings being used and can be used not only for diagnostics but for data collection in order to improve services.

In order to ensure all users can understand frequency data at a glance I have implemented a method which counts the data by iterating through the rows and counting those of equal value to "string". The overall count of the items is then output to the BarChart which enables users to see frequency data very clearly, and even allows those who may not understand computers to understand the meaning behind the data they have.



4. Generating HashSum Of File

Hashsums can be used as a comparator to tell if the information of a file has been tampered with. The HashFile button generates the Hashsum for the file which has been loaded into the table. The Hashsum is not only reported to the user as a MessageBox but saved to a text file for later use in a location of the users choosing. This can be used via the "Tools" menuitem, in which saved hashfiles can be loaded and compared to see if the values are the same, or have been changed, you can even copy and paste in a hashvalue to the textbox to compare with a loaded hashfile value.



Videos	hash.txt	07/04/2017 16:02	TXT File	1 KB
Windows (C:)	hash2.txt	07/04/2017 15:55	TXT File	1 KB

IDE and Language

For this project the language chosen is C#. This decision is due to the libraries available in C# allowing for memory management and simplistic file handling when working with large entries. C# is also a flexible language, it can be used with ASP.NET, LinQ and using SQL commands to read to and from databases.

Before this project, I have previous experience scripting in the C# language and, being a close syntax to Java find it a friendly language to work with.

During this project I have used many different libraries in C# including Microsoft.VisualStudio, Lumenworks.IO, System.Data.OleDb, and, System.Security.Cryptography.

I have chosen to use Visual Studio as an IDE. Visual Studio is a great IDE for many languages, it allows for easy use of Windows Forms in C# projects for fast GUI development and can easily be integrated with external plugins and libraries. Visual Studio also allows larger memory allocation for processes on 64bit systems which can be configured in project properties through the use of runtime macros.

Design

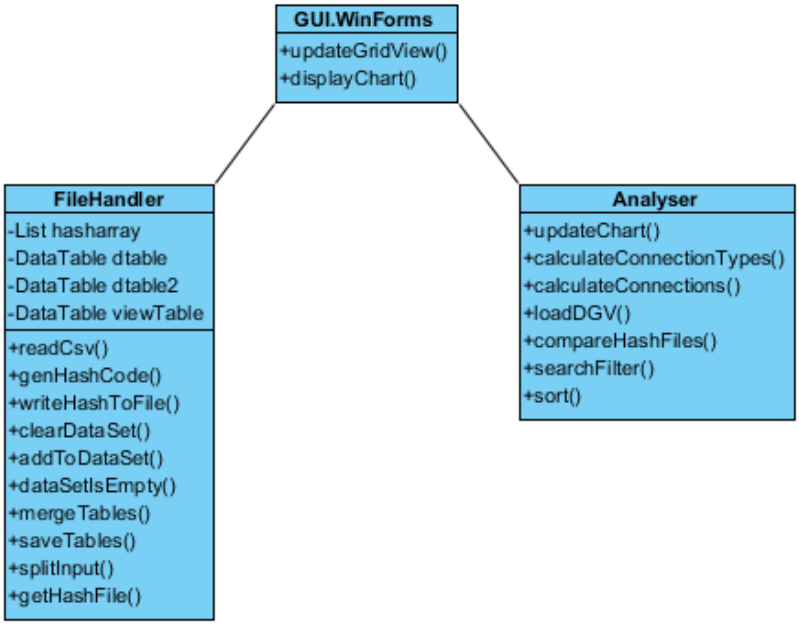
UML Designs

Class Diagrams

Initial Design

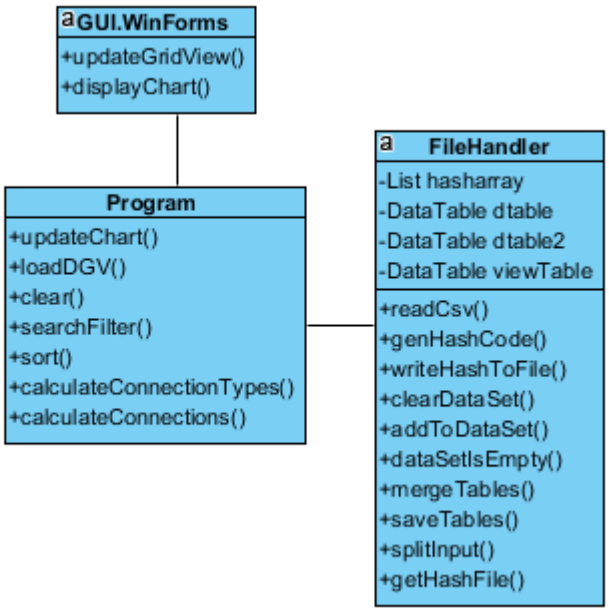
Due to the apparent simplicity of the application, I initially designed the application to consist of only a few classes. The main operations appeared to be Analysis and FileHandling. In this format I decided to have two classes alongside the main “program” class and GUI.

This design however proved to be difficult to stick to as it appears to be too simplistic. Although the application is simplistic some of the behaviours are quite complex and due to a lack of experiences I soon realised that the design would not work as intended.



Upon realising this the application was tweaked to taking less of an Object Oriented approach. Scripts were used to develop much of the functionality due to the short timescale in which to rebuild the application, however this also caused issues. Due to having to load the files into the DataGridView through DataSource.

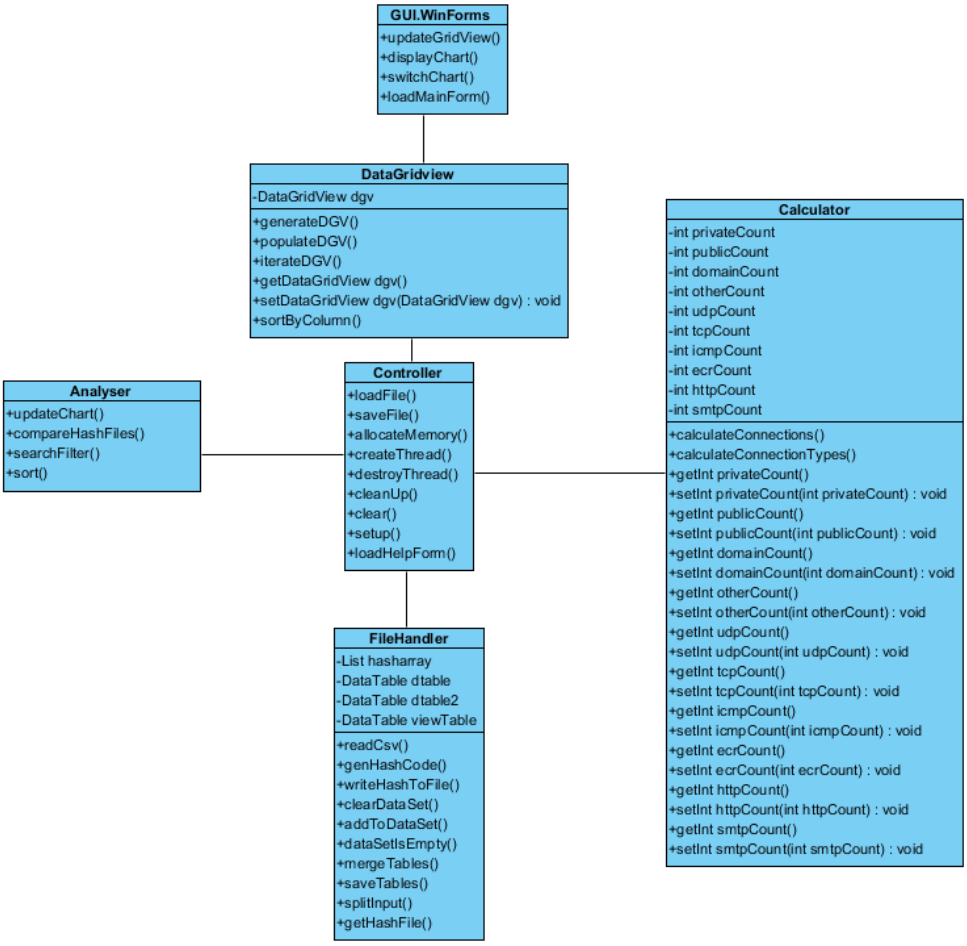
Final Design



Due to the issues with the initial design, the final implementation ended up more like the above diagram. The program does a lot of the work for calculating information and the FileHandler stores information and loads information from external sources.

Future Design Idea (Derived From Experience)

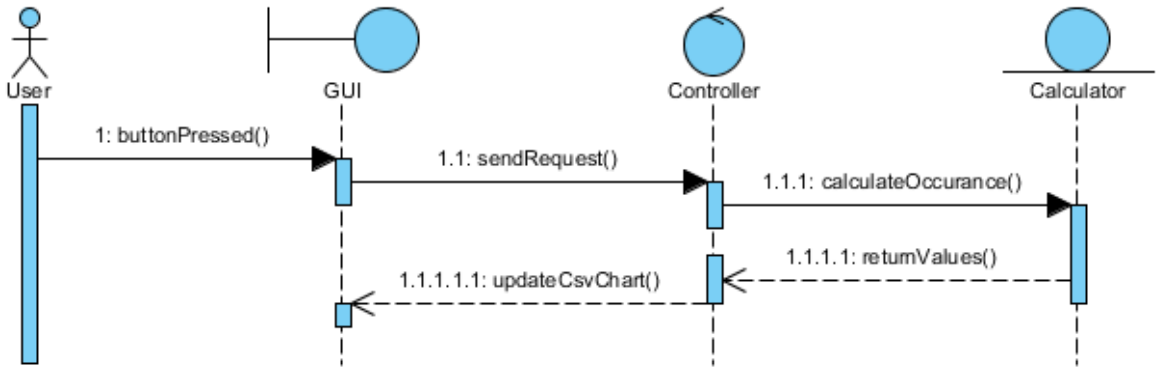
Although as of this time I do not have the time required to change the design completely I have learnt from this experience and redesigned the application with new ideas in mind in the hopes of solving the issues in the current application.



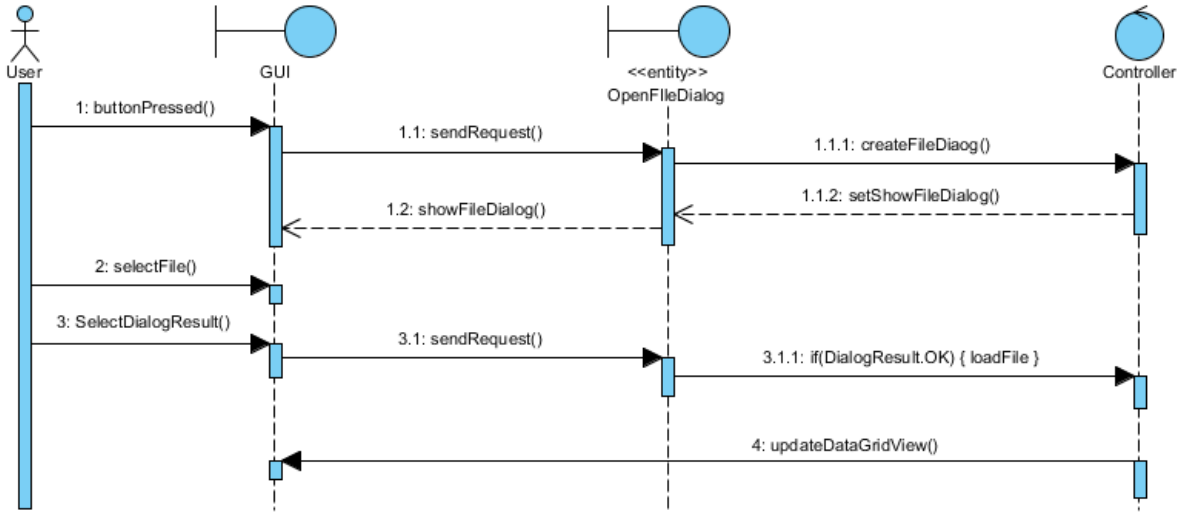
Sequence Diagrams

Sequence Diagrams have not had any significant changes for the functionality. The functionality works the same through OO as it does through the semantic approach, the only difference being the class the methods are being stored within.

CalculateChartData

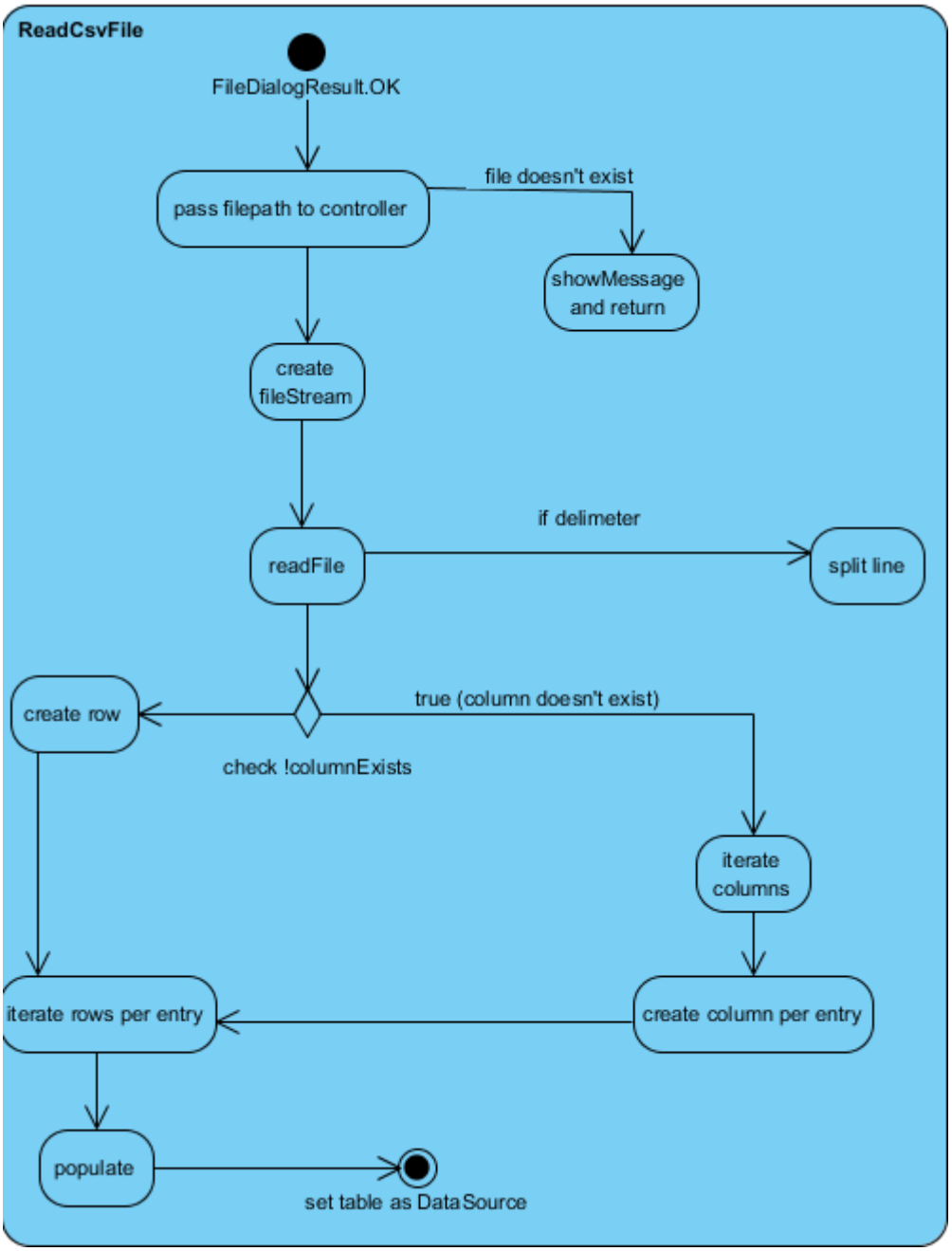


LoadCsvFile

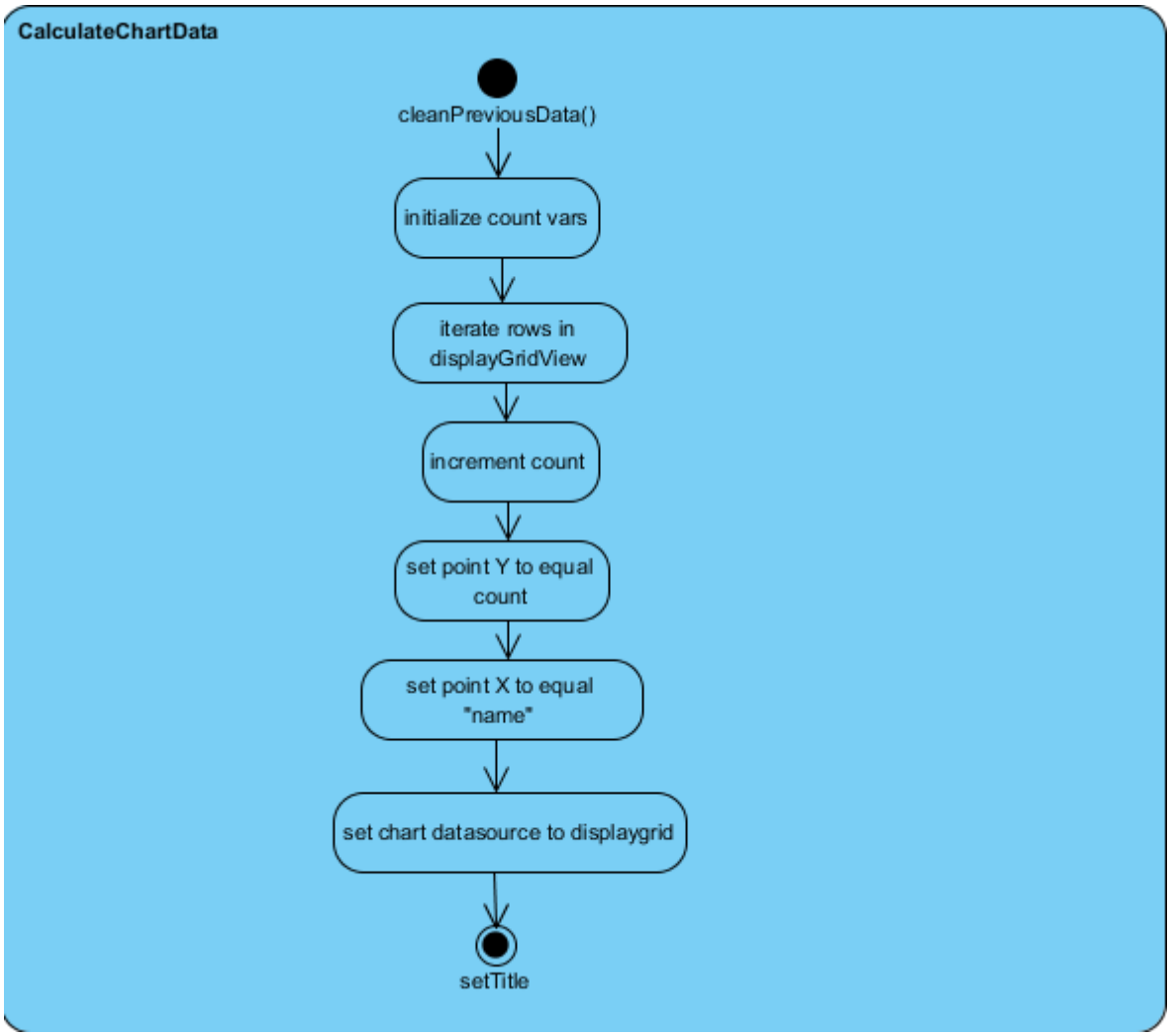


Activity Diagrams

ReadCsvFile

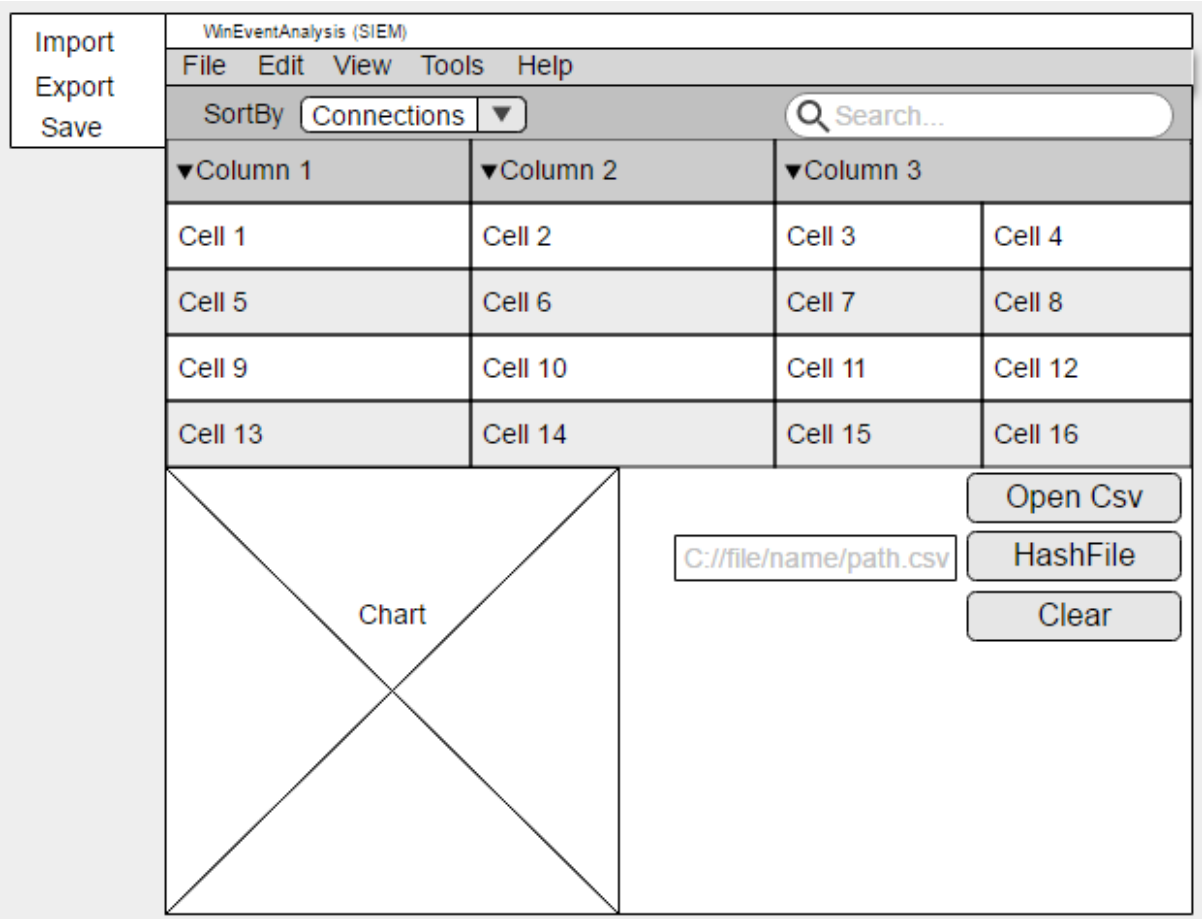


CalculateChartData



WireFrame

Due to the nature of the application I have decided that one View is more than adequate for the application. This allows the user to view all of the data in one place, this also makes writing analysis methods significantly easier. The design is made to be as simplistic as possible, no distracting colours or unnecessary items, many of the functionality is intended to be used through the toolbar or keyboard shortcuts, the intension is to allow users to view as much data as possible all at once In order to get an idea of the statistics. Although I will be designing the application based on a 800 * 600 design I intend to make it scalable so that more rows are shown on larger windows, yet the data is readable at the minimum size.



Implementation Notes

Although I have implemented many more methods in this project below are those which I have found the most challenging.

Implementation	Errors	Bugs	Fixes	Notes
GUI Design: QT	None	Layout does not scale well	More research is needed on Qt in order to complete the GUI.	Research: Qt GUI implementati on
GUI Design: Populating Layouts	None	A few bugs concerning the responsiveness moving buttons out of position	In order to fix this I have inserted the buttons into their own layouts to keep them together	Research: Qt updating the GUI from classes
GUI Design: Reworking	None	A few bugs with layout	I need to learn to use Winforms in more detail to complete the GUI	I have switched to WinForms as I have learnt it is much simpler to update the GUI from and uses a familiar syntax as-well as not having to port to different languages in order to implement with C#, I have also learnt that I can use ASP.NET framework with Winforms
GUI Design : Completion	None	A few bugs with responsiveness	I have fixed this in the way I fixed the Qt designs, I have	The Basic GUI is completed, the next step

		: the buttons and search bars travel too far.	paced all the buttons in layouts inside layouts and anchored their positions, some items have had margins added to place them appropriately.	is to implement the FileHandler to load in a Csv File
Class FileHandler	Issue reading in file	File does not show up after selecting in dialog	To solve this I had to research file input. In order to input the file my first implementation is to use filestream to read the file line by line.	Filestream method has failed. This appears to be due to no checks for delimiters, more research is required on reading specific Csv files.
Method readCsv	OutOfMemoryExceptio n	Lage Csv Files cannot be loaded and stored in memory.	In order to solve this I intend to read the Csv to a DataTable and store each DataTable in DataSets.	More research is required on Csv File Reading

Method CsvToTable	No official error	Loadtime on larger files is extremely slow. Can take up to 8minutes to load.	I have found a library from Microsoft for OleDb Connection, this method of reading a Csv creates a data object called OleDbConnection taking an SQL ConnecitonString query as a Parameter, frm here a definition of FileInfo is created with the path of the file, then using an adapter the file is parsed and can be read into a table using adapter.fill(table)	Although OleDb connection can read in large files up to 200mb, the performance fails as the processing time takes too long to load the file, a new solution is required.
Method splitInput	There is no row at position “x”	There is definitely data in the file at this position, however loop seems to miss this ?	I have tried to implement this method in order to fix the loading issue using OleDb. splitInput takes the table that OleDb fills, iterates through and assigns each half of the file to two separate tables, the idea behind this is to then load these tables at separate times giving the DataGridView separate “pages” in order to decrease load time, as the data	I have tried to fix this issue for many hours, I have checked over the code many times and cannot seem to find the culprit, I will continue to check the code later to see if I have missed anything. I have tried changing the values of totalrows to dtable.rows.count +1 and -

			is still complete in the original table, the original table can be used for calculations which need access to all of the data at once, or the table can be destroyed if needed and new tables created as needed using table.merge(table)	1 however this has not succeeded.
Method SplitInput	There is no row at position “x”	Data is still not being read at position x?	I have solved this issues after another few hours of playing with the method. The issue resided in having the iterative for loops nested within a foreach loop, this meant that row I thought was being read was not the correct row and the method was infact trying to read data from a row consisting of “empty”	This issue has been solved, more research may be needed into Csv however as the loading issue is still not resolved, the amount of time it takes to split the input has decreased the time taken to load, although not by as much as I had hoped.
Method readCsv	Timeout when reading large files	None header items being read in as column headers	I had found another solution to the issue of reading Csv files. In this solution I have used the Lumenworks.Framework.IO.Csv library to create a CachedCsvReader, this reader allows back and forth interaction with Csv files due to having its own cache, it is reportedly faster than reading in	This has been the most successful solution so far. Although the reader cuts it quite close to the 5minute maximum load time it is able to load large files into memory within 4minutes 37seconds

			using System.IO from the benchmarking on the documentation site.	There is one more solution worth trying, using FileStream, StreamReader and StringReader Update: The issue appeared to be an accidental setting of headers to true, after a few days I have realised this and switched to false, this method now works and is able to read large Csv files however I now have a much faster method
Method tertiaryCsvReader	Column does not exist Column already exists		I have decided to include the creation of columns within this method, creating them at runtime instead of pre-estimating the amount needed for larger files, this occurred due to having a few columns less than needed.	Implement: Check column exists, if the column does not exist -> createcolumn -> else -> iterate rows and create rows (reminder: check UML design)
Method tertiaryCsvReader		Empty row/column values	In order to combat this, the method needs a way of checking that the line is !null	Implement: if line == null || length == 0 continue; changeExist = true

				Once this has been implemented test the result. If the test passes remember to add the table as dgv datasource
Method tertiaryCsvReader	Due to the volatility of this method and the amount it does there may be unknown errors	“column” already exists	I have added Column + I to column name to ensure all of the columns generated have a unique name.	As I am unsure of how well-written this method is and not sure of what errors could possibly occur I have surrounded the main file I.O with a general try/catch statement to ensure a safe return if anything goes wrong with the file.
Method tertiaryCsvReader	No official errors	No known bugs	The tertiaryCsvReader has improved performance significantly	This method is the most efficient method and is the one I will use as my final implementati on. This method has cut down the time to read in the larger files down to a maximum of 3minutes 53seconds

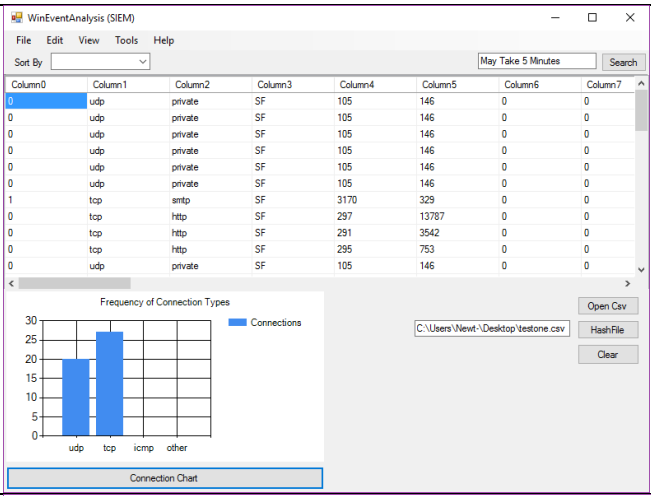
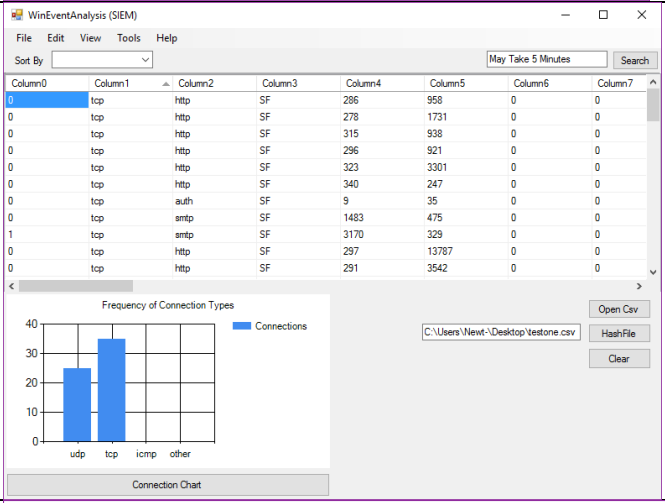
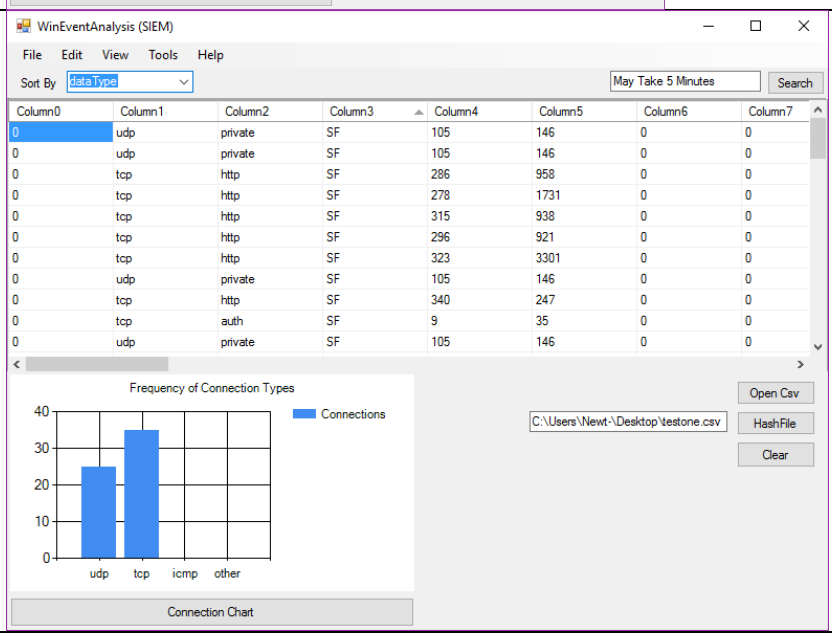
				This is the fastest I have been able to read in larger files and minimum to mid sized files load instantaneously.
Methods: ClearButton	None	None	A simple implementation of clearing out data	Research generating bar charts as next implementation.
Methods: Sort	None	None	The DataGridView made sorting by Category or "column" a very easy implementation, unknown to me the gridView can already sort in this way, however I have already implemented the ComboBox to which allows the user to select which "string" they sort by.	I have been trying to think of a way to create a string search for the application, however I am unsure whether or not it will work as planned after doing some research on the DataGridView
Methods: Search	OutOfMemoryException	Long wait time on larger files	Although the search appears to work on smaller files, the larger files either throw an outOfMemoryException or they take as much time to load as loading in a new file	The reason for the long search time is that the only way I can think to filter the results in this way involves either creating a new table and loading it in with only the results containing "string" or by

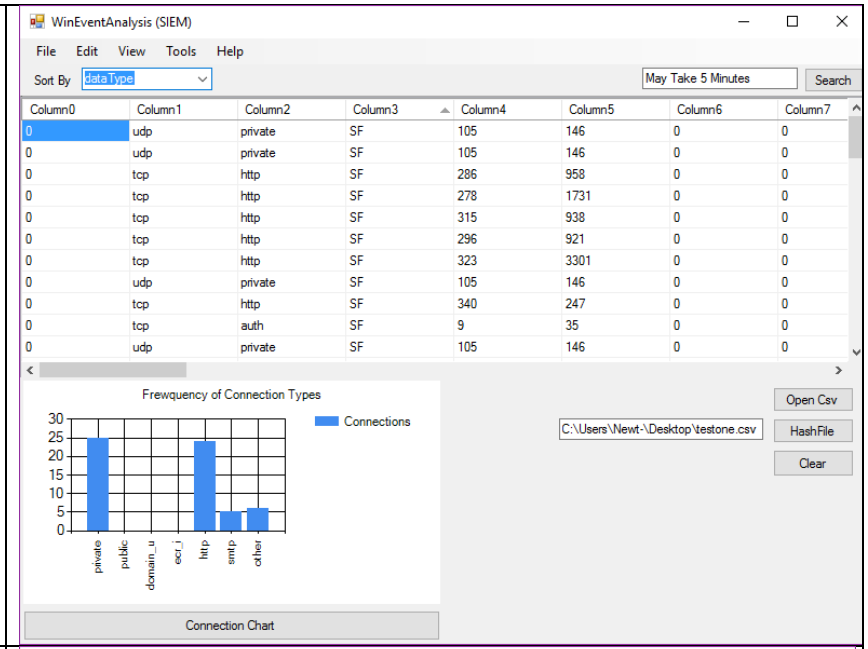
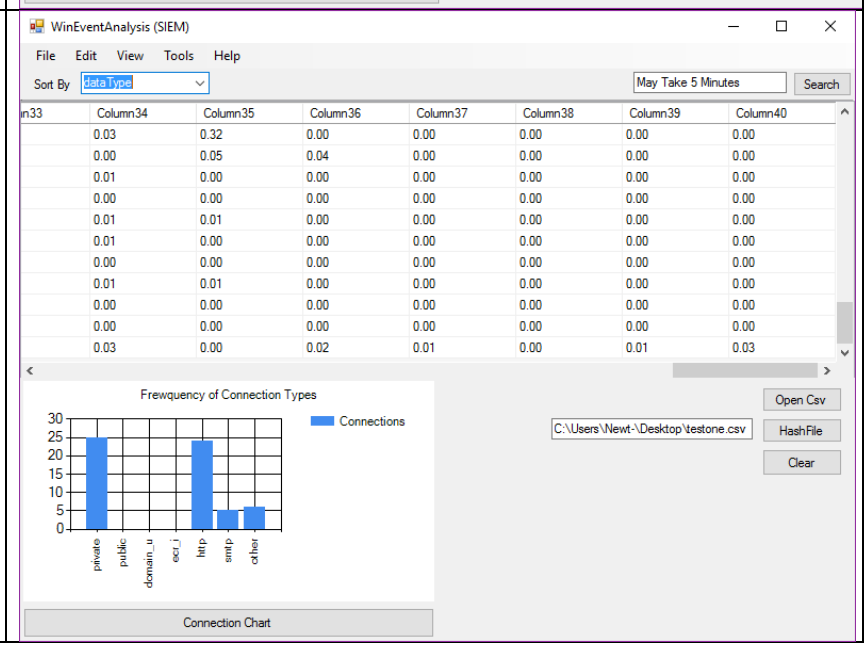
				removing anything that's value is not equal to "string", both of these methods consist of iterating the entire table, meaning that it will take as long as reading the file in again, for now this implementation has been abandoned for large files
Methods fillConnectionChart	OutOfMemoryException	Bar chart not loading values?	I have fixed the issue on smaller files where the bar chart does not load the vlaues	This issue occurred due to the fact I was counting the cells in the table containing "string" and then saving the count as the variable to the chart, however the chart AddXY method takes a string value as an argument, after realising this I used the toString method to solve this.
Methods: fillConncectionChart	OutofMemoryException		Although bar chart generation works on small -> mid sized files, larger files have been an issue due to the amount of memory the files	This issue appears to be somewhat resolved, the charts have loaded on the largest file on a few

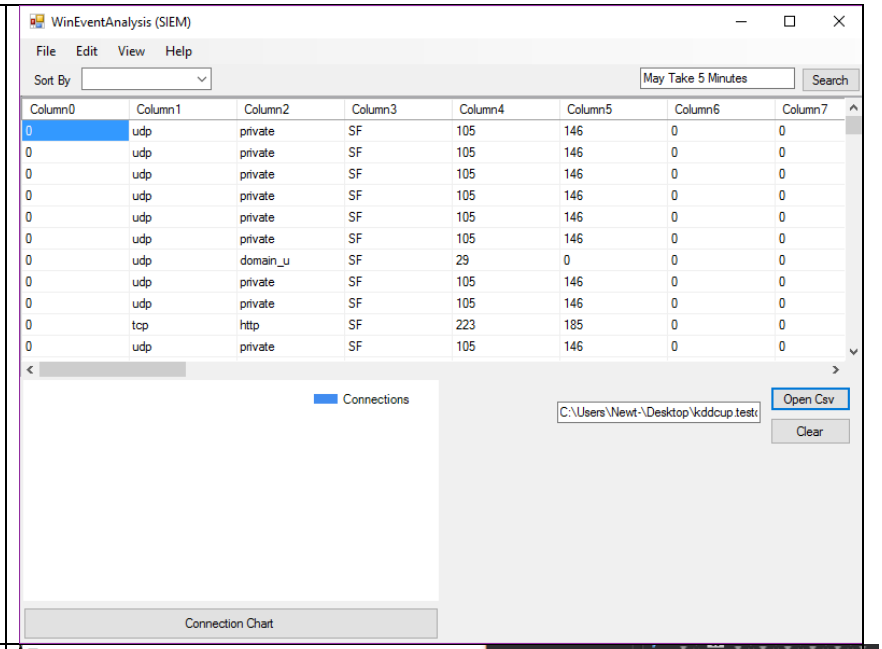
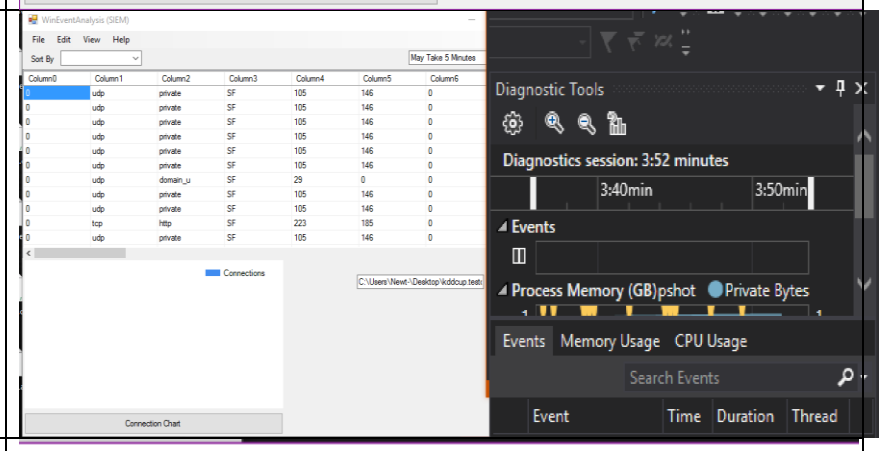
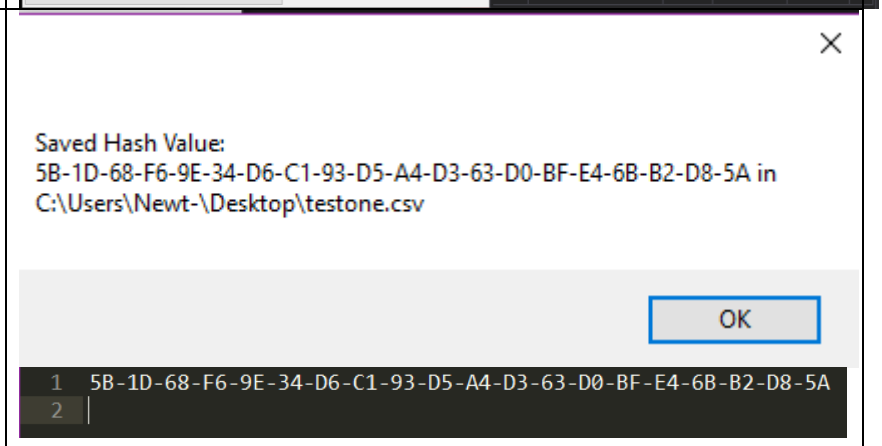
			themselves are taking up. In an attempt to solve this I have used MemoryFailPoint and changd the properties of my project to include macros which enable a larger memory access at runtime on 64 bit machines.	occasions, I have surrounded in a try/catch statement for the times it fails, although these techniques have improved performance for other files, the memory taken by the file appears to be too large for such larger-scale processing.
Method: Cleanup	None	None	Simple cleanup method implemented at the start of large processes and in the clear method, this forces garbage collection freeing up more space for processing.	Unfortunately this has not particularly improved the execution on larger files, I have also tried implementing threads, however the threads seem to take up more space.
Method: generateFileHash saveHashArray	None	None	A small hash generation, this works by using filestream, taking the path of the loaded file and using a sha256 algorithm provided by the Microsoft.security library to create a hash. I have converted the bitConverter to string so it can be printed to the user and saved to	After researching the Microsoft.Security.Cryptography library I have chosen to use the SHA1CryptoServiceProvider this is a very fast hashing provider even compared to the MD5 algorithm it appears to

			a text file for later use.	work much better for larger files. I have also created List<string> in order to store hash values with the intention of including the ability to compare hashsums somehow.
Method: CompareHashValues	No official error	Always returns either true or false	I understand why this is happening, it is because I am comparing two strings for equality and a string is always equal to string and the same for the attempt to compare readOutputs, but cannot think of a way to implement this method within the time that is left	I am in need of more research to implement this method as I do not understand how to compare the contents of two textfile values

Testing

Reading Csv	File Read Correctly, All Data Present	
Chart generation	Chart Generated Correct Data	
Sort	Sort Works Correctly, items are grouped by the selected item in ascending order	

Switching Chart Type	Chart Destroyed and Regenerated with correct data	
Scrolling	Scrolling works correctly	

Large Csv	Large Csv File Loaded correctly	
Large Csv TimeTest		
HashingTest	FileHashed and Saved to text file correctly	

Analysis

During this project, I have had to learn a lot through self-study, with this has come many challenges. As I have very little experience in programming, especially with larger and GUI based applications I have found this project very challenging.

The first complication which occurred was learning the File I/O required to read in a CSV file.
My earliest implementation of this consisted of trying to read the CSV like a normal file and then split up the rows depending on the delimiter, I soon came to realise this was not the most efficient way of doing this as I was reading a entire file, placing it to a table splitting it and then loading it, this took a long time to complete on large files. I have learnt a lot since then including on how to use new libraries efficiently, which libraries are the most efficient for which tasks and the inclusion of cryptography algorithms.

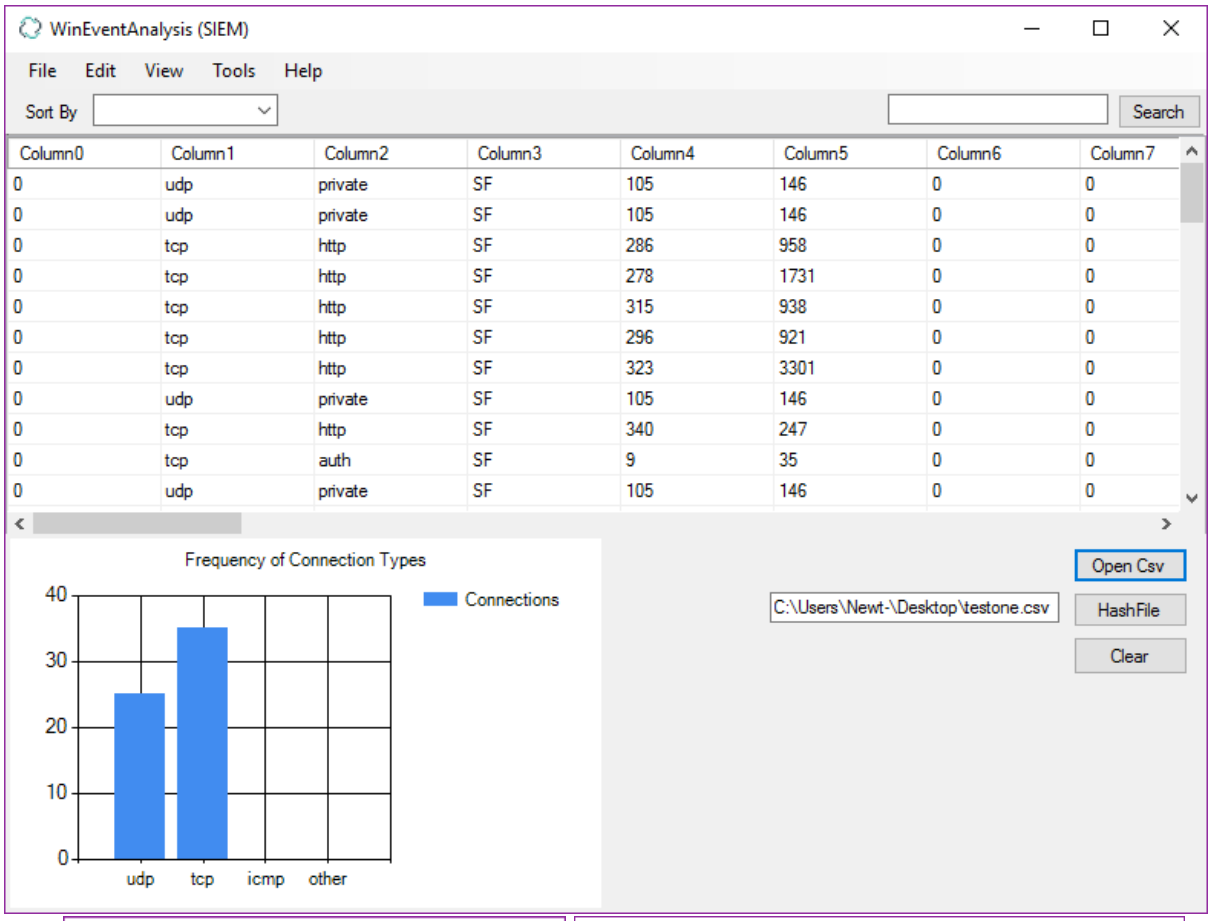
Although it took a lot of time and many different trials of implementations I have successfully overcome this challenge and learned a lot about the difficulties of SIEM management, but also how we can write code with better performance.

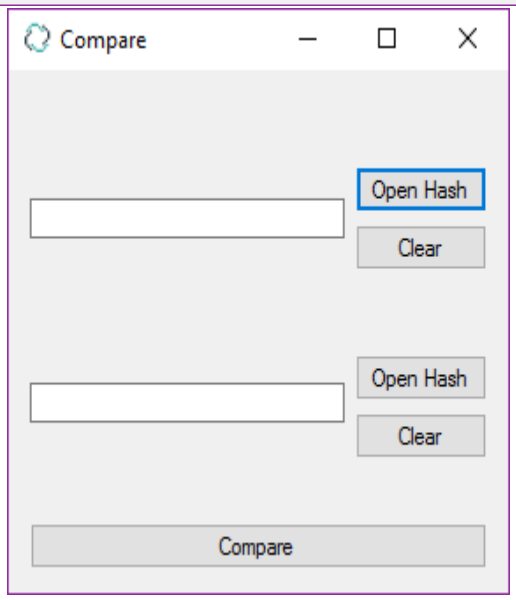
In the future I will spend more time before designing researching the project and coming up with a clearer picture of how the application might work. Saying this I find it difficult to envision before the project has been started and this may not help, however I intend to do much more research into writing Object Oriented programs as I have realised thinking of items in an abstract object oriented way is where I struggle the most, I would also write the functionality before the GUI in future, I feel this would have gone much smoother had I done this to begin with as a lot of the time I felt like I was simply “patching” methods onto the GUI frame instead of writing a class, this would also give me more time working on the most important items in the program and I may not come across the issues where I am unable to make sufficient progress for a few days at a time due to being unable to complete a method.

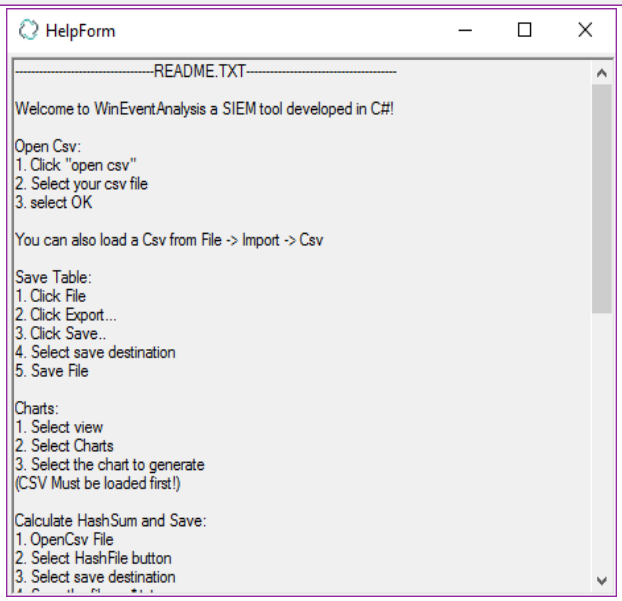
Overall I would try harder to ignore issues when I am stuck. To clarify, this does not mean ignore major issues exponentially, but I have realised I lost a lot of time working on issues that turned out to be quite simplistic and had I left and come back to it with a less tired mind I would have probably fixed the issues much sooner.

In conclusion, I believe this project to have been a success. Although it has been tricky and I am not 100% happy with the resulting application (I would have liked to had time to add more functionality, and I am not quite happy with my coding) I believe I have learnt a lot from the project and will use it to improve my skills for future use.

Finalised Artefact







Future Plans/Fixes

There are currently known bugs in the program. Due to complications nearing the end of the project there is less validation than I would have liked to include. Future plans include the plan to add a confirmation dialog when opening new files to ensure the user doesn't forget to save changes, although this is a simple implementation the time spent fixing larger bugs (such as issues reading Csv correctly) has taken precedence. The most major bug which impacted this was the fact that the CsvReader created the table columns dynamically, this meant however that every time a file was loaded the method tried to create the columns again when they already existed, this has been fixed by removing column creation from the method and adding checks to ensure the program only creates columns should they not exist.

A bug which is still lasting is the loading of files with headers. Although this is not a "breaking" bug, it can be confusing. Due to mostly working with Csv files which did not contain headers, when a Csv with a header is loaded into the project the first row is the "header" names below the columns, if I had more time I would fix this by checking if the file has headers first, and if it does remove the header row once the file has been loaded.