BLUESEAL FINANCIAL LTD
Extraction of Bloomberg Data to study the effect of Economic Releases on different Asset Classes

24-03-2014

Prepared by: Eve Anne Allen
Supervisor: Brett Houlding
DECLARATION

I declare that the work described in this dissertation has been carried out in full compliance with the ethical research requirements of the School of Computer Science and Statistics.

Signed: ___________________

Eve Anne Allen
24-03-2014
ABSTRACT

The aim of this project is to develop a program for the extraction of historical and future financial and economic earnings releases by Bloomberg L.P. through the subscription held by BlueSeal Financial Ltd. The program should also be capable of storing the extracted data to the BlueSeal Financial Ltd. proprietary data system. This improves data access and extraction and allows users to bypass the CSV and Excel formatted data extraction that was in place before this.

The final program consists of two main components, the extraction of the data composing a Request and Response interface, and the storage of this data which utilised CSV File formatting and the company data system, both of these components were built using the Java programming language.
BlueSeal Financial Ltd. trades its own capital on the global futures markets using various statistical and analytical models. These models can be built by focusing on the effect Bloomberg L.P. financial and economic earnings releases have on market trends. The main client contact throughout the project was Ross Prenderville, hereafter referred to as ‘the client’.

The client wanted an application that would serve as a user interface to extract data from BlueSeal Financial Ltd.’s subscription to Bloomberg L.P.’s API, returning the results to the user and offering an option to store the results on the company’s proprietary data system. The resulting program meets the client’s terms of reference, and adds additional functionality for the storage and manipulation of the data when on the company’s data system.

The main problems associated with this project were the client’s unfamiliarity with certain aspects of the subscription’s technical storage features and the inability to access the subscription remotely. Further difficulties arose with the complexity of the program. The Java programming language was used to build the application.

I would like to thank Ross Prenderville and Kevan O’Mahony for their invaluable help and feedback throughout the completion of this project. I would also like to thank Slava Kovalchuk for his assistance with the technical side such as the use of BlueSeal Financial Ltd.’s systems. Finally I would like to thank my project supervisor Brett Houlding, for his guidance and advice throughout the project.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>NO.</th>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>INTRODUCTION AND SUMMARY</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>The Client Company</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>The Project Background</td>
<td>1</td>
</tr>
<tr>
<td>1.3</td>
<td>Terms of Reference</td>
<td>2</td>
</tr>
<tr>
<td>1.4</td>
<td>Chapter Summaries</td>
<td>3</td>
</tr>
<tr>
<td>2.</td>
<td>PROGRAM OVERVIEW</td>
<td>4</td>
</tr>
<tr>
<td>2.1</td>
<td>Program Objectives</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>Software Background</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>Technical Aspect</td>
<td>5</td>
</tr>
<tr>
<td>2.4</td>
<td>Program Overview Diagram</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>User Interface Walkthrough</td>
<td>9</td>
</tr>
<tr>
<td>3.</td>
<td>DESCRIPTION OF WORK DONE</td>
<td>10</td>
</tr>
<tr>
<td>3.1</td>
<td>Requirements Phase</td>
<td>10</td>
</tr>
<tr>
<td>3.2</td>
<td>Design Phase</td>
<td>10</td>
</tr>
<tr>
<td>3.3</td>
<td>Development Phase</td>
<td>11</td>
</tr>
<tr>
<td>3.4</td>
<td>Implementation &amp; Testing</td>
<td>16</td>
</tr>
<tr>
<td>3.5</td>
<td>Program Layout</td>
<td>17</td>
</tr>
<tr>
<td>3.4</td>
<td>Problems Faced</td>
<td>18</td>
</tr>
<tr>
<td>4.</td>
<td>CONCLUSIONS AND RECOMMENDATIONS</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td>Conclusions</td>
<td>20</td>
</tr>
<tr>
<td>4.2</td>
<td>Recommendations</td>
<td>21</td>
</tr>
</tbody>
</table>
## APPENDICES

<table>
<thead>
<tr>
<th>NO.</th>
<th>CONTENT</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.</td>
<td>Original Project Outline</td>
<td>A.1</td>
</tr>
<tr>
<td>B.</td>
<td>Interim Project Report</td>
<td>B.1</td>
</tr>
<tr>
<td>C.</td>
<td>User Manual</td>
<td>C.1</td>
</tr>
<tr>
<td>D.</td>
<td>Design Documentation</td>
<td>D.1</td>
</tr>
<tr>
<td>E.</td>
<td>Screenshots</td>
<td>E.1</td>
</tr>
<tr>
<td>F.</td>
<td>Source Code</td>
<td>F.1</td>
</tr>
<tr>
<td>G.</td>
<td>Testing Documentation</td>
<td>G.1</td>
</tr>
</tbody>
</table>

REFERENCES
INTRODUCTION AND SUMMARY

1.1 The Client Company

BlueSeal Financial Ltd.

BlueSeal Financial Ltd. is a proprietary trading company. The company trades its own capital on the global futures markets using its team of risk, systems and support professionals to maximise its global opportunities. Established in 2006, BlueSeal Financial Ltd. researches and executes strategies designed to give superior risk adjusted returns. Their aim is to create trading systems that yield positive returns under all market conditions.

Bloomberg L.P

Bloomberg L.P is a financial software, data, and media company based in New York City. Set up in 1981 by Michael Bloomberg, it provides real time market data, analytics and financial calculations to Wall Street Firms [1]. Bloomberg releases real time financial market data four times per second on average and in 2009 accounted for roughly a third of the global financial data market [2].

1.2 The Project Background

Bloomberg L.P releases huge amounts of financial data information every second. These pieces of information can have either a positive or negative effect on different trading markets depending on their contents. BlueSeal Financial Ltd. can compare the financial data information released by Bloomberg L.P with the relevant trading markets at the time of their release to ascertain any correlation between shifts in the market and the information released. BlueSeal Financial can then use this correlation, if any is found, to adjust their trading strategies and systems for that market at the time of the next Bloomberg L.P financial data information release.

The Software Involved in the Project

This project involves a lot of interlocking and overlapping of different types of software. The Bloomberg API is a financial service to which BlueSeal Financial Ltd. has a monthly subscription. This subscription enables BlueSeal Financial Ltd. to access the data releases across the Bloomberg network as they happen.
The API is programmable by virtually any .NET programming language, although the company itself (Bloomberg L.P) recommends the following four:

- Java
- C/C++
- Perl
- Python

Of these four different software options, Java is the chosen language as the syntax of the Java programming language will fit most seamlessly with the other software involved in the project.

BlueSeal Financial Ltd. has their own database but as of now there is no way to access the Bloomberg data from this database. This software extraction program’s front end or GUI (Graphical User Interface) will be integrated into the proprietary data system in the form of a portal which will be located on the client’s desktop.

Project Aim

The aim of the project is to build an extraction program that will enable BlueSeal Financial Ltd. to access any piece of financial data information released by Bloomberg L.P. that they deem to be relevant to their trading systems and strategies. This data must be accessed in a quick and easily understood manner so as to ensure the BlueSeal Financial traders are neither misinterpreting the output nor being given the wrong information. The extraction program must then be integrated into BlueSeal Financial Ltd.’s database so that the information can be readily available to view, with or without constraints at any time.

Benefits to BlueSeal Financial Ltd.

The benefits of an extraction program like this for BlueSeal Financial Ltd. are fourfold:

- Enabling a way to view what effect data releases have on the overall market.
- To prove or disprove the theory that the effect of various data releases on the market are decreasing as time goes on.
- Enabling a way to view the trends within the data releases.
- Using those trends to help predict the effect of future data releases on the market.

1.3 Terms of Reference

To design and develop a computer program and subsequent database, which

- Extracts past and future economic and earnings releases from Bloomberg L.P.
- Stores the releases from Bloomberg L.P in any easily understood and simple to access manner.
- Can be combined with BlueSeal Financial Ltd.’s proprietary data system.
- Provides relevant user manual.
1.4 Chapter Summaries

Chapter 2 outlines the final program and assorted systems. It includes the reasoning behind the choice of languages, the use of BlueSeal Financial Ltd.’s systems and a walkthrough of the final program.

Chapter 3 includes the creation of the application. It discusses the process of the design and development from the initial requirements stage to the testing stage and any problems encountered therein.

Chapter 4 contains the conclusions and recommendations for the use of the system in relation to trading strategies and systems.
2. PROGRAM OVERVIEW

This chapter outlines how the final program works, the background to the system and the reasoning behind the choice of programming language.

2.1 Program Objectives

The program was designed, as stated in the terms of reference, to quickly and easily access both past and future economic and earnings releases from Bloomberg L.P and to store this data in BlueSeal Financial Ltd.’s database.

Creating a Graphical User Interface (GUI) that facilitates requests from the user improves the coherency and clarity of the data retrieval process. The interface provides a clear and concise process for the data retrieval and storage, which improves the time management of BlueSeal Financial Ltd.’s employees. The back end of the interface improves efficiency of accessing the relevant Bloomberg L.P. data and allows for combination of different pieces of data in the same file which further improves the storage of and access to the data.

The functionality of this system is unique to BlueSeal Financial Ltd. as it runs off their subscription to Bloomberg L.P. The program requires that the machine it runs on has basic java installed, and that it has a subscription to Bloomberg L.P.

The program provides users, who may not be familiar with technology, a way to quickly and easily access the relevant data, while also ensuring that, if they wish, all data is stored in the relevant location. It also has a failsafe that ensures that important data will not be overwritten. Instead the new data will merely be appended to the file in question.

2.2 Software Background

The subscription service that Bloomberg L.P. offers and that BlueSeal Financial Ltd. has availed of is the blpapi v3.x version. Blpapi stands for Bloomberg API. An API is an application programing interface; it dictates how software components should interact with each other and by which programming languages the relevant software and databases can be accessed. Blpapi can be directly accessed by the seven programming languages listed in Figure 2.2.1 [3].
From these languages, it was decided that the program would be written in Java. This language was chosen as it has the most support by both Bloomberg L.P. and online.

In the wake of the 2008 Financial Crisis, Bloomberg L.P. started to release their data on a more open scale in order to combat the issues the crisis caused [4]. On February 1st 2012 Bloomberg L.P. launched their Open Initiative website: www.openbloomberg.com; this website was set up by Bloomberg as part of their Open Data Initiative which was a part of the company’s ongoing effort to foster open solutions for the Financial Services Industry. This website and its assorted services offered support and advice for the programming of the Bloomberg API.

The Open Market Data Initiative BLPAPI: Developer’s Guide Version 1.34, a developer’s guide to the open and subscribed Bloomberg API is a succinct guide to programming the Bloomberg API. This guide has a step by step process in the best way to program the API in order to modify it to any company or individual’s needs. It also included several worked examples which were all programmed through Java [5].

BlueSeal Financial Ltd. uses a source code management (SCM) system known as Git. A SCM is a system that allows programmers and developers to manage the changes they make to their code. The Git SCM is used by BlueSeal Financial Ltd. because of its ability to handle a variety of large scale projects. The system allows programmers to build their code in a tree like structure, adding and deleting branches of code as appropriate. The tree structure allows for unparalleled speed in the editing of the source code for the program being developed. The master branch or “trunk” is the main bulk of the code for the program and as branches are added they can become incorporated into the trunk of the code if appropriate, but they can also be removed completely if unsuitable which does not affect the trunk of the code in any way. This allows for fast and easily editable programming.

2.3 Technical Aspect

This program has three interactive components which pull from two different sources; all of this is compiled in the Java source code of the program, a visual representation of the program’s technical structure is shown in Figure 2.3.1.
Structure of Program

Interactive Components
- Request GUI
  - This is the first component of the program the user interacts with. It uses the user's input to access the Bloomberg API and obtains the relevant data to comply with the user's request.
- Response GUI
  - This is the interface that appears after the user's request has been processed. It prints out all of the data accessed through the Request GUI along with the date each piece of data was originally released. This Interface also offers the user the option to save the data obtained as a CSV File to the user's database.
- CSV File
  - This is an optional interactive component of the program, if the user wishes to save the data obtained in the Response GUI then the CSV storage path is accessed and the data is saved in a CSV File to a pre-set location in the database. This CSV File can then be manipulated and analysed by the user at will.

Source Components
- Bloomberg API
  - This is the API set up by Bloomberg which allows the user to access any data they wish through their subscription.
- CSV Storage
  - This is the aspect of the system that allows the Java code to create and modify a CSV File that is used to store the obtained data in the company proprietary data system.
Java Source Code

- The Java source code is the actual syntax and coding of the program which allows for all of the above aspects of the program to be implemented. The source code is designed in such a way that all other aspects of the program are easily understandable and editable. It takes all user input and uses it to access the appropriate areas of both the Bloomberg API and the CSV Files in the BlueSeal Financial Ltd. data system.

2.4 Program Overview Diagram

Figure 2.4.1 overleaf demonstrates the basic layout behind the program. It can be seen that the primary interface for the program is the Request GUI. This interface is where all of the user information for the request is input. The subsequent option of saving the data is available in the Response GUI.

The divide between the user’s interaction with the Bloomberg API and CSV storage is evident in the diagram.

The diagram displays how the components interact with each other. In the Request GUI -> Bloomberg API section, the user passes information which allows the program to access the appropriate data in the Bloomberg API, this data is then printed in the Response GUI.

Where the Response GUI communicates with CSV storage, it is shown how the data presented in the Response GUI is input into a CSV File and automatically saved. This data can now be accessed at any point by bypassing the Request – Response paradigm completely.
Figure 2.4.1: System Overview Diagram

Key:

- Black lines represent the user path through the program.
- Red lines represent the passing of data through the system.
- Green cylinders indicate server-side software.
2.5 User Interface Walkthrough

From the Response GUI the user has to input the information with regards to what data they wish to extract from Bloomberg API.

Company

The user must enter the name of a company’s equity or stock index to be accessed. The API will return the data of just this company.

Fields

The user must enter which aspects or “fields” of the data they wish to view. The GUI contains a drop down list of the five most popular fields; however other fields are also accepted and can be entered manually.

Dates

The user must enter a start and end date for the data they wish to access.

Time Period

The user must choose a time period in which they wish to receive the data from a dropdown list, the options available are: daily, weekly, monthly, quarterly, semi-annually and yearly.

When the request has been processed, the Response GUI opens. This contains a list of the data requested. It also contains the option of writing the data contained in the GUI to a CSV File. This is done by entering a file name into the textbox provided.

When the “Write to CSV” button is clicked, the data is written out to a CSV File and saved to a pre-set file path on the company’s database. If the user wishes to append data to a file rather than creating an entirely new file for data that should be included in a previous file, then all they must do is enter the original file name in the textbox provided. This ensures that the new data stored in the Response GUI will append itself to the original file and no file will be overwritten. When the CSV Files are stored on the database they are, of course, completely editable.
3. DESCRIPTION OF WORK DONE

This chapter discusses the work carried out during the development of the program. The system was developed using an incremental model, shown in Figure 3.0.1. This means there were five stages in the process, namely requirement, design, development, implementation and testing. These steps were repeated where necessary. A description of the incremental model can be found in Appendix D.1, page D.1.

![Incremental Model Diagram]

Figure 3.0.1: Incremental Model

3.1 Requirements Phase

The first phase was critical to the design process as it was where the needs of the client were clearly defined. This was discussed during client meetings and led to the terms of reference, as seen in Section 1.3 (page 3).

The requirements were that the desired program would include an interface into which the user could input information, which would then be input into the Bloomberg API from which the relevant data could be accessed. The second interface would present this data in an easily understood fashion and give the option to save the data to the company’s database.

3.2 Design Phase

This stage focused primarily on how the system would actually work. Ease of use and simplicity are key to the application, so it was designed as such.
The Bloomberg API is programmable by the programming language Java (see Section 2.2). This language runs on free open source software which meant that BlueSeal Financial Ltd. would not have to redirect funds to purchase expensive software packages.

Developing the interface through Java was ideal as many example sources of Java code for the Bloomberg API are available through the Developer’s Guide.

The interface structure was to be simple and clear, so that users of any level of technical background could use it. The basic layout of the program was determined (using the Request GUI to input all relevant details to access the Bloomberg API) along with generating the code to access the relevant data through the API.

It is the functionality of this application that is critical as opposed to its aesthetic features. Due to this, it was decided that the appearance of the different GUIs were not to be the focus of the development phase. This was to save time devising a template that may have had to be rewritten when any changes were made, and to focus on the back end of the system, which was deemed more important.

All source code for the final working program can be found in Appendix F. An electronic version is also located on the CD provided.

### 3.3 Development Phase

The stages of development began after the completion of the planning stages. Each component was built separately, with continuous testing throughout, after the completion of every section, and integration.

**Sample Request Information**

The design of the Bloomberg API is laid out in such a way that if the program works for one set of information, it should work for all sets of information, keeping this in mind there was only one set of sample information used throughout the development and testing process. This was done to ensure that the right data was being conveyed through the system and the figures stayed the same in each test. The sample information was the following:

- **Company:** IBM US Equity
- **Fields:**
  - OPEN (the opening price)
  - PX_LAST (the last price)
- **Start Date:** 01/01/2010
- **End Date:** 31/12/2010
- **Time Period:** Monthly
Stage I: Java Request

Using the eclipse frontend for Java, the Java request was constructed to access the appropriate data needed. The original Java request was then run on BlueSeal Financial Ltd.’s servers using the command line.

The first request design incorporated the sample information into its source code. This meant that this request design would only access the monthly figures for the IBM US Equity in 2010 unless the actual source code supporting the entire program was edited. This was done as the original request prototype was merely a method of ensuring that the appropriate data was being accessed and conveyed through the Bloomberg API.

Stage II: Java Response

When the original Java request was fully functioning, code was added to print out the relevant data on the command line. This code was built within the original Java request to ensure the appropriate data was relayed.

The original printout of the data resembled the following:

IBM US Equity {
    date2010-11-30
    OPEN143.64
    PX_LAST141.46
**********
}

While the data was technically correct, the output method was confusing to read and difficult to follow, the code was adjusted to relay the data in a more user friendly manner. The final printout of the data followed the following format:

IBM US Equity:
    date = 2010-11-30
    OPEN = 143.64
    PX_LAST = 141.46

Stage III: Editable Java Request

This stage involved editing the original Java request code so that the information could be entered by the user, this would allow for the Java request to function appropriately in relation to any accurate information given by the user. This was done by adding a string input reader to the original code in the Request. The string reader allowed the program to ask for and accept the relevant information from the user using the built in console in the eclipse software.

This code would not affect the Java response as the Java response feeds off the code accessed by the Bloomberg API. If the user input Java request accessed the appropriate
sections of the Bloomberg API then the Java response would function as normal regardless of the new input methods.

**Stage IV: Writing the Data to a CSV File**

In order to accurately write the requested data to a CSV File, the com.csvreader package was necessary. This package is designed specifically for Java and allows the user to read in and write out information from and to a CSV File through Java as they wish. As there was no need for the program to read in data from a CSV File, the CsvReader class of the package was not needed, just the CsvWriter.

The CsvWriter class allowed the developer to set a destination for the CSV File to be stored, his location could not be modified by the user. The file name was also pre-set in this stage of the development process. This location was discussed and decided on by the client at BlueSeal Financial Ltd.

When a location was agreed, the source code first checked if that file already existed, if this was the case then the program appended the requested data to the end of the data already located in the file.

If the file did not already exist, the CsvWriter created a new CSV File with the given name in the given location. Then the necessary headers were added. The first being the company name, followed by the dates (the dates that given data was actually released by Bloomberg L.P) followed by the fields that were input by the user.

**Stage V: Building the Request GUI**

It was decided that the Request frontend or Graphical User Interface (GUI) would also be built using the Java programming language. To build this GUI three Java packages were utilised, the javax.swing package, the java.awt (Abstract Window Toolkit) package and the miglayout package. javax.swing is a Java package that is designed to create Java GUIs, it is the primary Java widget toolkit; the miglayout package is a simple package that makes the physical design of the GUI layout much easier and the java.awt package is a toolkit that makes creating the various different aspects of the GUI much simpler.

The Request GUI consists of five different components, namely company, fields, start date, end date and time period.

*Company* is the field into which the stock index or equity title of the company whose financial data the user wishes to access is entered, e.g. IBM US Equity.

*Fields* is the component into which the fields of data the user wishes to access is entered. This Request GUI has eight separate field dropdown boxes. This was the highest number of fields the client stated they would need. Each dropdown box contains the five most popular field choices, but if necessary, the user can enter the field they require manually.
Start and End Date state exactly which part of the Bloomberg API’s historical data framework to access. As Bloomberg L.P is an American owned and operated company, the start and end dates are submitted in the “American style” i.e. YYYYMMDD rather than the “European style” of DDMMYYYY. The option was presented to the client to change the date format from American to European, however this was deemed unnecessary due to the client’s familiarity with the American style date entry system as that is the date entry system used in the current program that accesses Bloomberg L.P. data.

Time Period is the amount of time between data releases that the user wishes to view. The Bloomberg API offers six different choices of time period, namely daily, weekly, monthly, quarterly, semi-annually, and yearly.

Figure 3.3.1 shows an empty Request GUI awaiting user input.

![Figure 3.3.1: Request GUI](image)

Stage VI: Building the Response GUI

The Response GUI used two of the three packages utilised in the Request GUI. The miglayout package was deemed unnecessary as the Response GUI is much less visually complicated and user input reliant than its predecessor.

The GUI was designed to be as simple and straightforward as possible. It had two necessary components; a display of the relevant data and an option to write that data to a CSV File.
The display aspect was designed to printout the data to the GUI in a simple easy to read textbox which contained a scrollbar when necessary. The data is displayed in chronological order and includes the date the figures were originally released by Bloomberg L.P.

The option to write the data to a CSV File was offered by a simple text field and button combination. The text field originally displays the text “Enter File Name” which disappears when this field is clicked by a mouse pointer.

The name, which becomes the CSV File name in the company’s proprietary data system, is entered by the user here and appended to the file location stored in the source code. The file is created, filled with the relevant data and stored in this location.

Figure 3.3.1 shows the Response GUI activated by the standard testing inputs.

![Example GUI](image)

Figure 3.3.2: Response GUI

**Stage VII: Additional Features**

The additional features added to this program came in the form of the design of the CSV writing and storage system within the program. The clients stated that they wished for the data that was extracted to be convertible to CSV File format so that it could be saved to their systems. The program was designed with this in mind and created a way for the conversion process of the files to double as an auto save feature. Once the user decided that the files should be converted to CSV it is implied that they wish to save this file to their servers, therefore the program saves the file automatically for the user.

The user also does not have to specify a file location. The system is designed in such a way that the user does not have to worry about choosing a location or looking for a location to save the CSV File, this is set up automatically by the program. The user does not need to specify anything but a name for the current file they are converting.

The final additional feature that is incorporated into the program is its ability to append to CSV Files rather than writing over them. This means that if a user enters a file name that
already exists on their servers the program will append the new data to the data already saved in that CSV File rather than writing over the file. This is useful for two main reasons.

The first is that it provides a failsafe when writing files to CSV format. If the file name was entered by mistake or the user did not realise that there was an existing file of the same name already on the servers, then the data saved to the original file would not be lost.

The second is that it allows the users to append their CSV Files where necessary, the users can just add the new data to their old data rather than creating multiple files that follow on chronologically from each other. The appending CSV feature allows all of these to be saved in the one file which makes it easier for the client when it comes to accessing and analysing said data. This was a major time and space issue for the client and has been solved by the appending feature in the program.

3.4 Implementation & Testing

As stated in the previous section, the program was continuously tested throughout development. Each component was tested at every stage of completion, and then again once they were integrated with the system. Details of this testing system can be found in Appendix G.1, page G.1.

The final testing was carried out after the final system had been implemented on BlueSeal Financial Ltd.’s system. The test was carried out by the client at BlueSeal Financial Ltd. who accessed various different aspects of financial data from various different companies through the Bloomberg API. These different sets of data were then written out to a location on the company servers.

All aspects of the program tested successfully on the BlueSeal Financial Ltd. servers. Appendix E shows screenshots of the working program in action.

The system will be on the company server, with a user manual provided so that any employee of BlueSeal Financial Ltd. will be able to use the program regardless of their technical competency. (See Appendix C for manual and Section 4.2 for deployment recommendations, page 21.)
3.5 Program Layout

Figure 3.5.1 displays the final layout of the system and its internal interactions.

A description of the commands and processes carried out in each section can be found in Appendix D.3, page D.2 – D.3.
3.6 Problems Faced

Many problems and issues were faced and overcome throughout the creation of this program for BlueSeal Financial Ltd. One of the main issues was that of the subscription BlueSeal Financial Ltd. has to the Bloomberg API. This subscription is an agreement between the two parties to grant BlueSeal Financial Ltd. access to the economic and financial information released by Bloomberg L.P. through their company servers which are located in their offices in the Dublin Exchange Facility.

This is the only physical location in which Bloomberg L.P. will allow BlueSeal Financial Ltd. to access their data, there is no way to remotely access BlueSeal Financial Ltd.’s subscription to the Bloomberg API. This was a serious problem as it meant that the program, which was being developed remotely, could not be accurately tested remotely.

As the client knew this would be an issue, a provided USB drive was provide containing various portions of historical Bloomberg L.P. financial data with the thought that the program could be tested remotely on this data while in development. The data was stored in CSV format on the USB drive as that was the only format in which the client could access the data at the time, an issue the program would solve.

For the first ten weeks of the development process, the program was built using the sample data as a viable test sample. When the development had reached the onsite testing stage, the program was loaded onto the company servers and run through the system. It was at this point that it was discovered that although the data could be accessed through the Excel system and downloaded as CSV Files, the raw data of the Bloomberg API was not stored in this format.

Unfortunately this made a lot of the previous design and development of the program null and void. The program had to be re-imagined and re-designed so as to incorporate this new information in its source code in order to function to the client’s specifications. The program re-design was an arduous process but was made much easier by the client supplied Developer’s Manual which offered various samples and tips on designing a Request and Response program in a multitude of different programming languages. (See Appendix D.2 page D.1 – D.2, for a detailed view of the original system design and its modifications)

The next problem arose with the way the data was read out to the user’s screen (as previously mentioned in Section 3.3, page 12). While the data was technically present and correct, the output format was neither aesthetically pleasing nor easy to understand. Given that the client’s reason behind the development of this system was to create a faster and easier way to access the relevant information, the output given by the original Java Response was unacceptable. The code had to be rewritten so that the relevant information could be parsed behind the scenes and presented in a fashion that was simple to comprehend and easy to read.

The storage method also posed problems, in the originally designed program (designed in the first ten weeks). The storage aspect of the program was the simplest to build as the data was already presented in CSV format, the re-designed program did not have this advantage.
The re-designed program accessed raw data and then converted it into aesthetically pleasing Java output; this output then had to be converted into aesthetically pleasing and easily understood CSV data to meet with the client’s specifications. Luckily, some of the Java source code that had become irrelevant after the system re-design was now relevant to the new storage method. This code was adapted slightly in order to access Java output rather than CSV data and then proceeded to write out the data quickly and concisely. Given the sheer amount of Java source code that had been written around the CSV format of the data due to the miscommunication of how the client accessed the Bloomberg API data, the additions to the program were added with little to no complications.

The final issue faced was that of creating user friendly GUIs, as the program requires the user to interact with multiple interfaces it was important that they were both simple to use and easy to understand. The Java programming language does not offer as much support to GUIs as other languages and it was rather difficult to write GUIs that appeared when called for, rather than having all interfaces appear when the user activates the program regardless of whether or not they are necessary at that junction. This was done by building the Response GUI within a command in the Request GUI thereby ensuring that unless that command was chosen by the user, the Response GUI would remain dormant.

These issues were all development related and should not be a problem for the client. As the problems have now been fixed, they should not affect the use of the program. Provided the user follows the instructions located in the User Manual provided, they should not face any major problems.
4. CONCLUSIONS AND RECOMMENDATIONS

4.1 Conclusions

The finished program meets the client’s requirements (as outlined in Section 1.3, page 3). The previous system was slow and convoluted as the Bloomberg API data could only be accessed through a program that was set up through various CSV and Excel files. This system provides the client with a program that provides fast and easy access to the Bloomberg L.P. data as well as offering an easier and more useful method of storage whilst still allowing the data to be manipulated for the client’s needs in CSV format.

Outline of how each requirement was met, along with additional capabilities provided:

- The client requested a program to extract past and future economic and earnings releases from Bloomberg L.P. that could then be added to their proprietary data system. This was provided through the creation of a Java program which both accessed the Bloomberg API directly and stored the relevant information in CSV format.
- The program allows for precise data retrieval from the Bloomberg API using field specific Java requests.
- The resulting output is viewed on screen; this can be saved to the company proprietary data system, if desired.
- The program allows for data to append to a file when being stored in the data system rather than overwriting files or creating multiple versions of slightly different files which should be conglomerated.
- The files stored to the company data system are saved in CSV format which allows them to be easily manipulated and analysed by the company systems in that format.

While the program is simple and intuitive, a user manual has been provided to give instruction to ensure that any users at any level of technical competency can utilise the program. (Appendix C)

The development process involved working primarily with the Java programming language, it also utilised access and manipulation of CSV File storage. The final system involves multiple interfaces that can be manipulated by the user in conjunction with direct access to both the Bloomberg API and the company proprietary data system. This meant it was necessary to use and expand a range of current skills, along with developing new ones.

Creating this program was both challenging and satisfying, using a single program to quickly and easily access two different systems and manipulate the company servers to their exact specifications. Converting the data from the Bloomberg API into CSV format was troublesome due to the difference in how the data was formatted and how that formatting could be edited; however learning to work around this was another educational experience.
The client’s acceptance and use of this program proves its success. As new economic earnings are released by Bloomberg L.P. the program will be able to access them so it is not necessary to adapt the program for future use.

4.2 Recommendations

The key recommendation for this system is to adhere to the steps outlined in the User Manual, provided in Appendix C, during deployment. This provides a detailed description and walk through of how to access the Bloomberg API, retrieve the relevant information and store it to the company proprietary data system.

This system has already been deployed on the company server. The program is designed to work on any server with basic Java installed and a Bloomberg API subscription, while all aspects that can be manipulated by the user are designed to the client’s specific requirements, the system can be easily adapted and deployed on any system with the relevant software and subscription.

The program can be accessed through the portal located on the client’s desktop and the CSV File versions of the saved data can be found in BlueSeal Financial Ltd.'s proprietary data storage system.

The data saved in CSV format can be manipulated so it is advised that if a user were to manipulate this data they would state so either within the CSV File itself or in its name. Given the program’s ability to access and store any relevant data from the Bloomberg API in seconds, the need for data backup, while advised, is not fundamental. The only way the data would be fully lost i.e. deleted from the system, would be if the time limit for that data on the Bloomberg API had run out.

As this program was not designed to be extensible, further development of the program would involve directly editing or changing the source code. While not advised, if this is necessary, it should be completed by an individual with a thorough grounding in both the intricacies of the Bloomberg API and the Java programming language.
# APPENDICES

<table>
<thead>
<tr>
<th>NO.</th>
<th>CONTENT</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.</td>
<td>Original Project Outline</td>
<td>A.1</td>
</tr>
<tr>
<td>B.</td>
<td>Interim Project Report</td>
<td>B.1</td>
</tr>
<tr>
<td>C.</td>
<td>User Manual</td>
<td>C.1</td>
</tr>
<tr>
<td>D.</td>
<td>Design Documentation</td>
<td>D.1</td>
</tr>
<tr>
<td>E.</td>
<td>Screenshots</td>
<td>E.1</td>
</tr>
<tr>
<td>F.</td>
<td>Source Code</td>
<td>F.1</td>
</tr>
<tr>
<td>G.</td>
<td>Testing Documentation</td>
<td>G.1</td>
</tr>
</tbody>
</table>

REFERENCES
A ORIGINAL PROJECT OUTLINE

Client: BlueSeal Financial Ltd
Project: Extraction of Bloomberg Data to study the effect of Economic Releases on different Asset Classes
Location: Unit S1
The Dublin Exchange Facility
IFSC, Dublin 1
Client Contact: Ross Prenderville
01 792 0960
ross.prenderville@bluesealfinancial.com
Dept. Contact: Aideen Keaney

Client Background:

BlueSeal Financial is a proprietary trading firm specialising in futures and options. The firm is active on all major futures exchanges across the world. Traders at BlueSeal Financial aim to exploit pricing inefficiencies in Equity Index, Bond and Commodity Futures.

Client Requirement:

The project involves the analysis of economic and earnings releases on different asset classes. The goal is to measure the movements in asset classes prior to a release and when a release deviates from or meets expectations. The effects will be measured for the immediate aftermath of the release and for several periods after the release. The project will aim to determine the significance of these releases, whether the importance of certain releases trend, that is become more significant depending on the economic reality, and whether the size of any deviation from expectations is significant. The project will offer the student experience in the proprietary trading field and an introduction into how ideas are generated and put into practice as trading strategies.

What is involved for the student?

- Writing a program to extract past and future economic and earnings releases from Bloomberg.
- Creation of a database of releases from Bloomberg.
- Writing a program to combine this database of releases to BlueSeal’s proprietary data system.
- Analysis of the impact of the releases on the price of different assets over time.
- Report on the process used to arrive at the results of the study and a report on possible applications of the study.
B. INTERIM REPORT

Management Science and Information Systems Studies
Project: Data Extraction and Storage Program
Client: BlueSeal Financial Ltd.
Student: Eve Anne Allen
Supervisor: Brett Houlding

Review of Background and Work to Date

BlueSeal Financial Ltd. is a proprietary trading company that trades its own capital on the global futures markets using its team of risk, systems and support professionals to maximise its global opportunities. Bloomberg L.P is a financial software, data and media company based in New York City. It provides real time market data, analytics and financial calculations to Wall Street Firms.

The aim of the project is to create a computer program that will automatically extract the relevant data from Bloomberg L.P on its release and store it in a self-contained database on BlueSeal Financial Ltd.’s servers.

Although there are various programming languages that are compatible with the Bloomberg L.P API, the aspects of this project that directly interact with the API will be carried out using Java. SQL will also be utilised to create and merge the new database containing the Bloomberg financial information with BlueSeal Financial Ltd.’s own database.

To date, a prototype program has been made which reads in financial data from the test set. This data is then written out to a new file created to be saved in a specified location. The pilot program allows the data to be manipulated and abbreviated so that only the relevant aspects of the data are written into the database and stored on BlueSeal Financial Ltd.’s servers.

Terms of Reference

To provide a structured review of available systems and to design and develop a computer program and subsequent database, which

- Extracts past and future economic and earnings releases from Bloomberg L.P.
- Stores the releases from Bloomberg L.P in any easily understood and simple to access manner.
- Can be combined with BlueSeal Financial Ltd.’s proprietary data system.
- Provides relevant user manual.

Further Work

Much of the work still needs to be developed. The Java program needs to run and update the relevant database automatically. The creation and integration of the Bloomberg L.P API financial database needs to be designed and formatted in the same way as BlueSeal Financial Ltd.’s current database so as to create a professional and easily understood portal by which to access the data. This aspect of the project will require a lot of input from the client.
The Java program should be fully functional, to client specifications, over Christmas. The SQL database should be fully designed and integrated with the client's systems in January. The user manual will be prepared in February. The full report will also be written in February and March.

**Conclusions**

It is too early in the project to state any concrete conclusions at this stage. Meetings with the client will confirm that the Java program successfully extracts and reformats the Bloomberg L.P API financial data in a fashion that will be useful to the work and financial analysis that will be prepared on the data. The integration of the new database and the client's current database will allow for fast and simple access to the relevant data stored on their servers. So far it has been proven that the data can be extracted and reformatted from one database to another using the sample data the client provided. Further testing on real data needs to be carried out along with an investigation of the back end of their database so as to discover the best way to integrate it with the new database.
C. USER MANUAL

Please see the User Manual accompanying this report. The User Manual details instructions to navigate the various user interfaces.

An electronic copy can be found in the CD provided.
D. **DESIGN DOCUMENTATION**

This appendix describes the process through which the program was developed. The initial design of the program and its re-imagined version are included, along with descriptions of the nature of the final Java interfaces and commands.

D.1 **Software Methodology – Incremental Method**

Software methodology provides structure during the development process, the model used whilst building this program was the incremental method. This model involves the following steps:

- Requirement Analysis
- Design
- Development
- Implementation
- Testing

As the program progressed, the design and development phases were repeated where modifications made it necessary.

![Figure D.1.1: Incremental Design Methodology](image)

D.2. **Program Design**

During the design phase of the system the basic structure was devised. It originally followed the thought that the Bloomberg API data was stored in CSV format and that because of this there would be no need for file conversion in the program.

However after discovering that the raw data was not stored in CSV format, the basic structure was redesigned. The program would be centred on the main interface in the program, known as the *Request* page from which the requests for Bloomberg L.P. data can
be sent. The secondary interface, \textit{Response}, would then allow access to the company proprietary data system.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{basic_system_redesign.png}
\caption{Basic System Redesign}
\end{figure}

\section*{D.3 Command Descriptions}

The interactions between the different interfaces and servers the program uses are all dependent on the commands input by the user. The following is a simple description of the nature of each command.

\textit{REQUEST GUI:}

\textbf{Company}

\begin{itemize}
  \item States the stock index or equity of the company whose data the user wishes to access.
\end{itemize}

\textbf{Field}

\begin{itemize}
  \item Allows the user to detail which aspect of the data they wish to view.
  \item Can contain up to eight separate values.
\end{itemize}

\textbf{Start Date}

\begin{itemize}
  \item Specifies from which data release date the user wishes to view the data.
  \item Does not have to be the exact data release date as the program will access the first data release on or after that date.
\end{itemize}
End Date

- Specifies up to which data release date the user wishes to view the data.
- Does not have to be exact data release date, the program will only access data up to that date, whether there has been a data release on that date or not is irrelevant.

Time Period

- The number of days between data releases that the user wishes to view. The user is offered a choice of five options for this section:
  - Daily
  - Weekly
  - Monthly
  - Quarterly
  - Semi-Annually
  - Yearly

Results

- Accesses the Bloomberg API using the input information and displays the Response GUI

**BLOOMBERG API SERVER**

- The data input by the user in the Request GUI is used to access and return the appropriate data in the Bloomberg API server.

**RESPONSE GUI**

"Enter File Name"

- States the name by which the user wishes the data to be stored as in CSV format in the company proprietary data system.

**Write To CSV**

- Writes the data displayed in the Response GUI to a CSV File named by the user and automatically saved, stored, and in some cases appended to the company proprietary data system.

**PROPRIETARY DATA SYSTEM**

- The CSV File created in the Response GUI is stored in this system in a pre-set location.

**CSV FILE**

- The CSV readout of the data displayed in the Response GUI.
E. SCREENSHOTS

This appendix contains screenshots from the various user interfaces and the storage aspects of the program.

Figure E.1: Desktop Access Portal

Figure E.2: Empty Request GUI
Figure E.3: Company Input

Figure E.4 Start & End Dates Input
Figure E.5: Fields Dropdown Input

Figure E.6: Fields Manual Input
Figure E.7: Time Period Input

Figure E.8: Response GUI
Figure E.9: Named CSV File Response GUI

Figure E.10: Named CSV File Location
<table>
<thead>
<tr>
<th>Date</th>
<th>OPEN</th>
<th>PX_LAST</th>
</tr>
</thead>
<tbody>
<tr>
<td>29/01/2010</td>
<td>131.18</td>
<td>122.39</td>
</tr>
<tr>
<td>25/02/2010</td>
<td>123.23</td>
<td>127.16</td>
</tr>
<tr>
<td>31/03/2010</td>
<td>127.5</td>
<td>128.25</td>
</tr>
<tr>
<td>30/04/2010</td>
<td>128.35</td>
<td>129</td>
</tr>
<tr>
<td>31/05/2010</td>
<td>129.39</td>
<td>125.26</td>
</tr>
<tr>
<td>30/06/2010</td>
<td>124.69</td>
<td>123.48</td>
</tr>
<tr>
<td>30/07/2010</td>
<td>123.55</td>
<td>128.4</td>
</tr>
<tr>
<td>31/08/2010</td>
<td>129.25</td>
<td>123.13</td>
</tr>
<tr>
<td>30/09/2010</td>
<td>125.31</td>
<td>134.14</td>
</tr>
<tr>
<td>29/10/2010</td>
<td>135.51</td>
<td>143.6</td>
</tr>
<tr>
<td>30/11/2010</td>
<td>143.64</td>
<td>141.46</td>
</tr>
<tr>
<td>31/12/2010</td>
<td>143.51</td>
<td>146.76</td>
</tr>
</tbody>
</table>

Figure E.11: Named CSV File
F. SOURCE CODE

This appendix contains the source code of the application. An electronic copy can be found in the CD provided.

GuiClas.java

/**
 * All relevant Java packages and commands are imported
 */
import java.awt.EventQueue;
import javax.swing.JFrame;
import net.miginfocom.swing.MigLayout;
import javax.swing.JComboBox;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JTextField;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;

/**
 * Runs the GUI
 * @author user
 */
public class GuiClas {

/**
 * Builds the physical shape of the GUI
 */
private JFrame frame;

@SuppressWarnings("rawtypes")

/**
 * Allows access to the information input by the user
 */
private JComboBox field1CB,
field2CB,timePeriodCB,field3CB,field4CB,field5CB,field6CB,field7CB,field8CB;

private JTextField sDateTF,eDateTF,companyTF;

/**
 * Launch the application.
 */
public static void main(String[] args) {

}
EventQueue.invokeLater(new Runnable() {
    public void run() {
        try {
            GuiClas window = new GuiClas();
            window.frame.setVisible(true);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});

/**
 * Create the application.
 */
public GuiClas() {
    initialize();
}

/**
 * Initialise the contents of the frame.
 */
@SuppressWarnings({ "unchecked", "rawtypes" })
private void initialize() {
    frame = new JFrame();
    frame.setBounds(100, 100, 450, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setLayout(new MigLayout("", 
        
            
                    
                                
                               
           

/*
 * Set the names and positions for each user editable aspect of the application
 */
JLabel lblNewLabel = new JLabel("Company:       ");
frame.getContentPane().add(lblNewLabel, "cell 0 0,alignx left,aligny center");

companyTF = new JTextField();
frame.getContentPane().add(companyTF, "cell 1 0,growx");

JLabel lblStartDateyyyymmdd = new JLabel("Start Date (YYYYMMDD)");
frame.getContentPane().add(lblStartDateyyyymmdd, "cell 3 0,alignx trailing");

sDateTF = new JTextField();
frame.getContentPane().add(sDateTF, "cell 4 0,growx");
sDateTF.setColumns(10);

JLabel lblFields = new JLabel("Fields: ");
frame.getContentPane().add(lblFields, "cell 0 1,alignx left,aligny center");

field1CB = new JComboBox();
/**
 * Allows the combo boxes in the "Fields" to be editable
 * the user can enter an item not from the pre-set list
 */
field1CB.setEditable(true);
field1CB.setModel(new DefaultComboBoxModel(new String[] {"","OPEN", "CLOSE", "HIGH", "LOW", "PX_LAST"}));
frame.getContentPane().add(field1CB, "cell 1 1,alignx left,aligny center");

JLabel lblEndDateyyyymmdd = new JLabel("End Date (YYYYMMDD) ");
frame.getContentPane().add(lblEndDateyyyymmdd, "cell 3 1,alignx trailing");

eDateTF = new JTextField();
frame.getContentPane().add(eDateTF, "cell 4 1,growx");
eDateTF.setColumns(10);

field2CB = new JComboBox();
field2CB.setEditable(true);
field2CB.setModel(new DefaultComboBoxModel(new String[] {"","OPEN", "CLOSE", "HIGH", "LOW", "PX_LAST"}));
frame.getContentPane().add(field2CB, "cell 1 2,alignx left,aligny center");

field5CB = new JComboBox();
field5CB.setEditable(true);
field5CB.setModel(new DefaultComboBoxModel(new String[] {"","OPEN", "CLOSE", "HIGH", "LOW", "PX_LAST"}));
frame.getContentPane().add(field5CB, "cell 1 3,alignx left,aligny top");

JLabel lblTimePeriod = new JLabel("Time Period ");
frame.getContentPane().add(lblTimePeriod, "cell 3 3,alignx trailing");
timePeriodCB = new JComboBox();
timePeriodCB.setModel(new DefaultComboBoxModel(new String[] {"DAILY", "WEEKLY", "MONTHLY","QUARTERLY","SEMI_ANNUALLY", "YEARLY"}));
frame.getContentPane().add(timePeriodCB, "cell 4 3,growx");

field3CB = new JComboBox();
field3CB.setEditable(true);
field3CB.setModel(new DefaultComboBoxModel(new String[] {"", "OPEN", "CLOSE", "HIGH", "LOW", "PX_LAST"}));
frame.getContentPane().add(field3CB, "cell 1 4,alignx left,aligny top");

field4CB = new JComboBox();
field4CB.setEditable(true);
field4CB.setModel(new DefaultComboBoxModel(new String[] {"", "OPEN", "CLOSE", "HIGH", "LOW", "PX_LAST"}));
frame.getContentPane().add(field4CB, "cell 1 5,alignx left,aligny top");

/**
 * Sets the action performed by the "Results" button
 */
JButton button = new JButton("Results");
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            /**
             * Accesses the DataRunClas to access the
             Bloomberg API
             */
            DataRunClas.Test(companyTF.getText(),

            field1CB.getSelectedItem().toString(),
            field2CB.getSelectedItem().toString(),
            field3CB.getSelectedItem().toString(),
            field4CB.getSelectedItem().toString(),
            field5CB.getSelectedItem().toString(),
            field6CB.getSelectedItem().toString(),
            field7CB.getSelectedItem().toString(),
        }
    }
});
field8CB.getSelectedItem().toString(),
   sDateTF.getText(),
   eDateTF.getText(),

   timePeriodCB.getSelectedItem().toString() 
   
});

   catch (Exception e1) {
   // TODO Auto-generated catch block
   e1.printStackTrace();

   }

});

field6CB= new JComboBox();
field6CB.setEditable(true);
field6CB.setModel(new DefaultComboBoxModel(new String[]{"","OPEN","CLOSE","HIGH","LOW","PX_LAST"})
frame.getContentPane().add(field6CB,"cell 1 6,alignx left");

field7CB= new JComboBox();
field7CB.setEditable(true);
field7CB.setModel(new DefaultComboBoxBoxModel(new String[]{"","OPEN","CLOSE","HIGH","LOW","PX_LAST"})
frame.getContentPane().add(field7CB,"cell 1 7,alignx left");

/**
 * Adds the "Results" button to the application
 */
frame.getContentPane().add(button, "cell 3 7");

field8CB= new JComboBox();
field8CB.setEditable(true);
field8CB.setModel(new DefaultComboBoxBoxModel(new String[]{"","OPEN","CLOSE","HIGH","LOW","PX_LAST"})
frame.getContentPane().add(field8CB,"cell 1 8,alignx left");
}
DataRunClas.java

/**
 * Import all necessary aspects for building the Response GUI
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;

/**
 * Import all necessary aspects for writing the Response to a CSV File
 */
import java.io.File;
import java.io.FileWriter;
import com.csvreader.CsvWriter;

/**
 * Import all necessary aspects to access the Bloomberg API
 */
import com.bloomberglp.blpapi.*;
import com.bloomberglp.blpapi.Event;

/**
 * Main Class that accesses the Bloomberg API retrieves and displays all the relevant information
 * @author user
 * *
 */
public class DataRunClas {

    /**
     * Creates a JTextField where the user input will be
     * the name of the data when saved as a CSV File
     */
    private static JTextField testeree;

    /**
     * Accesses the GuiClas and retrieves the relevant information to perform the request
     */
    public static void Test (final String companyName, final String field1,
            final String field2, final String field3, final String field4, final String field5,
final String field6, final String field7, final String field8, final String sDate, final String eDate, final String timePeriod) throws Exception{

/**
 * Establishes a connection to the Bloomberg API
 */
SessionOptions sessionOptions = new SessionOptions();
sessionOptions.setServerHost("localhost");
sessionOptions.setServerPort(8194);
final Session session = new Session(sessionOptions);

if (!session.start()) {
    System.out.println("Could not start session.");
    System.exit(1);
}
if (!session.openService("//blp/refdata")) {
    System.out.println("Could not open service "+
                     "/blp/refdata");
    System.exit(1);
}
/**
 * Physically builds the response GUI
 */
JFrame guiFrame = new JFrame();
guiFrame.setTitle("Response GUI");
guiFrame.setSize(300,250);
guiFrame.setLocationRelativeTo(null);

final JPanel text = new JPanel();

/**
 * Sets a textfield in which the Bloomberg API data will be displayed
 */
JTextArea textTest = new JTextArea(10,20);

/**
 * Adds a scrollbar to the textfield that displays the Bloomberg API data
 */
JScrollPane scrollPane = new JScrollPane(textTest);
scrollPane.set VerticalScrollBarPolicy(ScrollPane.VERTICAL_SCROLLBAR_ALWAYS);

text.add(guiFrame.getContentPane().add(scrollPane));

/**
 * Makes the text area ineditable so that human error does not occur when reading or saving the response data
 */
textTest.setEditable(false);

testeree = new JTextField();
testeree.setText("Enter File Name");

/**
 * Makes the text "Enter File Name:" disappear when clicked with the mouse pointer
 */
testeree.addFocusListener(new FocusListener(){
    public void focusGained(FocusEvent e){
        testeree.setText("");
    }

    @Override
    public void focusLost(FocusEvent arg0) {
        // TODO Auto-generated method stub
    }
});

final JButton button = new JButton("Write to CSV");
/**
 * Sets the action performed by the "Write to CSV" button
 */
button.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent event){

        /**
         * Sets the file path for the storage of the CSV File
         */
        String outputFile = "C:/Users/bloomberg1/Documents/Data CSV/"+testeree.getText()+".csv";

        boolean alreadyExists = new File(outputFile).exists();
try {

    CsvWriter csvOutput = new CsvWriter(new FileWriter(outputFile, true), ',');

    /**
     * If the file does not already exist, Headers are written first
     * The dates and fields
     */
    if (!alreadyExists){
        csvOutput.write(companyName);
        csvOutput.endRecord();
        csvOutput.write("Date");
        csvOutput.write(field1);
        csvOutput.write(field2);
        csvOutput.write(field3);
        csvOutput.write(field4);
        csvOutput.write(field5);
        csvOutput.write(field6);
        csvOutput.write(field7);
        csvOutput.write(field8);
        csvOutput.endRecord();
    }

    /**
     * The request sent to the Bloomberg API
     */
    CorrelationID requestID = new CorrelationID(1);
    Service refDataService = session.getService("//blp/refdata");
    Request request = refDataService.createRequest("HistoricalDataRequest");
    request.append("securities", companyName);
    request.append("fields", field1);
    request.append("fields", field2);
    request.append("fields", field3);
    request.append("fields", field4);
    request.append("fields", field5);
    request.append("fields", field6);
    request.append("fields", field7);
    request.append("fields", field8);
    request.set("startDate", sDate);
    request.set("endDate", eDate);
    request.set("periodicitySelection", timePeriod);
}
session.sendRequest(request, requestID);

boolean continueToLoop = true;

/**
 * Loops through each piece of data for the time period
 */
while (continueToLoop) {
    Event e =
    (e.eventType().intValue()) {
        switch

            Event.EventType.Constants.RESPONSE: // final event
            continueToLoop = false; //
            case

            Event.EventType.Constants.PARTIAL_RESPONSE:

            handleResponseEvent2(e, csvOutput);

            break;

            default:
            System.out.println(e);
            break;

        }

        csvOutput.close();
    }

    catch (Exception e) {
        e.printStackTrace();
    }
}

/**
 * The request sent to the Bloomberg API
 */
CorrelationID requestID = new CorrelationID(1);
Service refDataService = session.getService("//blp/refdata");
Request request = refDataService.createRequest("HistoricalDataRequest");
    request.append("securities", companyName);
    request.append("fields", field1);
request.append("fields",field2);
request.append("fields",field3);
request.append("fields",field4);
request.append("fields",field5);
request.append("fields",field6);
request.append("fields",field7);
request.append("fields",field8);
request.set("startDate",sDate);
request.set("endDate",eDate);
request.set("periodicitySelection",timePeriod);

session.sendRequest(request, requestID);

boolean continueToLoop=true;

/**
 * Loops through each piece of data for the time period
 */
while (continueToLoop) {
    Event event = session.nextEvent();
    switch (event.eventType().intValue()) {
    case Event.EventType.Constants.RESPONSE: // final
        continueToLoop = false; // fall through
    case Event.EventType.Constants.PARTIAL_RESPONSE:
        handleResponseEvent(event,textTest);
        break;
    default:
        System.out.println(event);
        break;
    }
}

guiFrame.add(text, BorderLayout.CENTER);
guiFrame.getContentPane().add(testeree, BorderLayout.SOUTH);
guiFrame.getContentPane().add(button, BorderLayout.NORTH);
guiFrame.setVisible(true);

/**
 * Accesses the relevant data from Bloomberg API
 * and prints this data to the Textfield in the Response GUI
 * along with the dates each piece of data was released
 */
private static void handleResponseEvent(Event event, JTextArea textTest)
    throws Exception{
MessageIterator iter = event.messageIterator();

while(iter.hasNext()){
    Message message=iter.next();
    Element ReferenceDataResponse=message.asElement();

    if(ReferenceDataResponse.hasElement("responseError")){
        System.out.println("UhOh");
    }

    Element typeData =
    ReferenceDataResponse.getElement("securityData");
    Element fieldDataArray = typeData.getElement("fieldData");

    for(int i =0; i <fieldDataArray.numValues();i++){
        Element fieldData =
        fieldDataArray.getValueAsElement(i);

        for (int j =0; j<fieldData.numElements();j++){
            Element field = fieldData.getElement(j) ;

            textTest.append(field.name()+" = ");

            textTest.append("\n");
        }
        textTest.append("\n");
    }
}

/**
* Accesses the relevant data from Bloomberg API
* and prints this data to the CSV File in the File location
* along with the dates each piece of data was released
* and saves it automatically
*/
private static void handleResponseEvent2(Event e, CsvWriter csvOutput)
throws Exception{

    MessageIterator iter = e.messageIterator();
    while(iter.hasNext()){
        Message message=iter.next();
        Element ReferenceDataResponse=message.asElement();

        if(ReferenceDataResponse.hasElement("responseError")){
        }
System.out.println("UhOh");
}

Element typeData = ReferenceDataResponse.getElement("securityData");
Element fieldDataArray = typeData.getElement("fieldData");

for(int i =0; i <fieldDataArray.numValues(); i++){  
  Element fieldData = fieldDataArray.getValueAsElement(i);

  for(int j =0; j<fieldData.numElements(); j++){  
    Element field = fieldData.getElement(j);
    csvOutput.write(field.getValueAsString());
  }
  csvOutput.endRecord();
}
}
CsvWriter.java

/*
 * Java CSV is a stream based library for reading and writing
 * CSV and other delimited data.
 *
 * Copyright (C) Bruce Dunwiddie bruce@csvreader.com
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License as published by the Free Software Foundation; either
 * version 2.1 of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public
 * License along with this library; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA
 */

package com.csvreader;

import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.BufferedWriter;
import java.io.Writer;
import java.nio.charset.Charset;

/**
 * A stream based writer for writing delimited text data to a file or a stream.
 */
public class CsvWriter {
    private Writer outputStream = null;

    private String fileName = null;

    private boolean firstColumn = true;

    private boolean useCustomRecordDelimiter = false;

    private Charset charset = null;
}
/**
 * Double up the text qualifier to represent an occurrence of the text
 * qualifier.
 */
public static final int ESCAPE_MODE_DOUBLED = 1;

/**
 * Use a backslash character before the text qualifier to represent an
 * occurrence of the text qualifier.
 */
public static final int ESCAPE_MODE_BACKSLASH = 2;

/**
 * Creates a{@link com.csvreader.CsvWriter CsvWriter} object using a file
 * as the data destination.
 *
 * @param fileName
 *      The path to the file to output the data.
 * @param delimiter
 *      The character to use as the column delimiter.
 * @param charset
 *      The{@link java.nio.charset.Charset Charset} to use while
 *      writing the data.
 */
public CsvWriter(String fileName, char delimiter, Charset charset) {
    if (fileName == null) {
        throw new IllegalArgumentException("Parameter fileName can not
be null.");
    }

    if (charset == null) {
        throw new IllegalArgumentException("Parameter charset can not
be null.");
    }

    this.fileName = fileName;
userSettings.Delimiter = delimiter;
this.charset = charset;
}

/**
* Creates a {@link com.csvreader.CsvWriter CsvWriter} object using a file
* as the data destination. &nbsp;Uses a comma as the column delimiter and
* ISO-8859-1 as the {@link java.nio.charset.Charset Charset}.
* @param fileName
* The path to the file to output the data.
*/
public CsvWriter(String fileName) {
    this(fileName, Letters.COMMA, Charset.forName("ISO-8859-1"));
}

/**
* Creates a {@link com.csvreader.CsvWriter CsvWriter} object using a
* Writer to write data to.
* @param outputStream
* The stream to write the column delimited data to.
* @param delimiter
* The character to use as the column delimiter.
*/
public CsvWriter(Writer outputStream, char delimiter) {
    if (outputStream == null) {
        throw new IllegalArgumentException("Parameter outputStream can
not be null.");
    }

    this.outputStream = outputStream;
    userSettings.Delimiter = delimiter;
    initialized = true;
}
public CsvWriter(OutputStream outputStream, char delimiter, Charset charset) {
    this(new OutputStreamWriter(outputStream, charset), delimiter);
}

/**
 * Gets the character being used as the column delimiter.
 * @return The character being used as the column delimiter.
 */
public char getDelimiter() {
    return userSettings.Delimiter;
}

/**
 * Sets the character to use as the column delimiter.
 * @param delimiter The character to use as the column delimiter.
 */
public void setDelimiter(char delimiter) {
    userSettings.Delimiter = delimiter;
}

public char getRecordDelimiter() {
    return userSettings.RecordDelimiter;
}

/**
 * Sets the character to use as the record delimiter.
 * @param recordDelimiter The character to use as the record delimiter. Default is combination of standard end of line characters for Windows, Unix, or Mac.
 */
public void setRecordDelimiter(char recordDelimiter) {
    useCustomRecordDelimiter = true;
    userSettings.RecordDelimiter = recordDelimiter;
}

/**
 * Gets the character to use as a text qualifier in the data.
 *
* @return The character to use as a text qualifier in the data. */

public char getTextQualifier() {
    return userSettings.TextQualifier;
}

/**
 * Sets the character to use as a text qualifier in the data.
 * @param textQualifier The character to use as a text qualifier in the data.
 */
public void setTextQualifier(char textQualifier) {
    userSettings.TextQualifier = textQualifier;
}

/**
 * Whether text qualifiers will be used while writing data or not.
 * @return Whether text qualifiers will be used while writing data or not.
 */
public boolean getTextQualifier() {
    return userSettings.UseTextQualifier;
}

/**
 * Whether text qualifiers will be used while writing data or not.
 * @param useTextQualifier Whether to use a text qualifier while writing data or not.
 */
public void setTextQualifier(boolean useTextQualifier) {
    userSettings.UseTextQualifier = useTextQualifier;
}

public int getEscapeMode() {
    return userSettings.EscapeMode;
}

public void setEscapeMode(int escapeMode) {
    userSettings.EscapeMode = escapeMode;
}

public void setComment(char comment) {
    userSettings.Comment = comment;
public char getComment() {
    return userSettings.Comment;
}

/**
 * Whether fields will be surrounded by the text qualifier even if the
 * qualifier is not necessarily needed to escape this field.
 * @return Whether fields will be forced to be qualified or not.
 */
public boolean getForceQualifier() {
    return userSettings.ForceQualifier;
}

/**
 * Use this to force all fields to be surrounded by the text qualifier even
 * if the qualifier is not necessarily needed to escape this field. Default
 * is false.
 * @param forceQualifier Whether to force the fields to be qualified or not.
 */
public void setForceQualifier(boolean forceQualifier) {
    userSettings.ForceQualifier = forceQualifier;
}

/**
 * Writes another column of data to this record.
 * @param content The data for the new column.
 * @param preserveSpaces Whether to preserve leading and trailing whitespace in this
 * column of data.
 * @exception IOException Thrown if an error occurs while writing data to the
 * destination stream.
 */
public void write(String content, boolean preserveSpaces)
    throws IOException {
    checkClosed();
    checkInit();
if (content == null) {
    content = "";
}

if (!firstColumn) {
    outputStream.write(userSettings.Delimiter);
}

boolean textQualify = userSettings.ForceQualifier;

if (!preserveSpaces && content.length() > 0) {
    content = content.trim();
}

if (!textQualify && userSettings.UseTextQualifier
    && (content.indexOf(userSettings.TextQualifier) > -1 ||
        content.indexOf(userSettings.Delimiter) > -1 || (!useCustomRecordDelimiter && (content
            .indexOf(Letters.LF) > -1 ||
        content
            .indexOf(Letters.CR) > -1)) || (useCustomRecordDelimiter && content
            .indexOf(userSettings.RecordDelimiter) > -1) || (firstColumn && content.length() > 0 &&
        content
            .charAt(0) == userSettings.Comment) ||
        // check for empty first column, which if on its own line must
        // be qualified or the line will be skipped
        (firstColumn && content.length() == 0))) {
    textQualify = true;
}

if (userSettings.UseTextQualifier && !textQualify
    && content.length() > 0 && preserveSpaces) {
    char firstLetter = content.charAt(0);
    
    if (firstLetter == Letters.SPACE || firstLetter ==
Letters.TAB) {
        textQualify = true;
    }
}
if (!textQualify && content.length() > 1) {
    char lastLetter = content.charAt(content.length() - 1);

    if (lastLetter == Letters.SPACE || lastLetter == Letters.TAB) {
        textQualify = true;
    }
}

if (textQualify) {
    outputStream.write(userSettings.TextQualifier);

    if (userSettings.EscapeMode == ESCAPE_MODE_BACKSLASH) {
        content = replace(content, "" + Letters.BACKSLASH, ""
                         + Letters.BACKSLASH + Letters.BACKSLASH);
        content = replace(content, "" + userSettings.TextQualifier, ""
                         + Letters.BACKSLASH + userSettings.TextQualifier);
    } else if (userSettings.EscapeMode == ESCAPE_MODE_BACKSLASH) {
        content = replace(content, "" + Letters.BACKSLASH, ""
                         + Letters.BACKSLASH + Letters.BACKSLASH);
        content = replace(content, "" + userSettings.Delimiter, ""
                         + Letters.BACKSLASH + userSettings.Delimiter);

        if (useCustomRecordDelimiter) {
            content = replace(content, "" + userSettings.RecordDelimiter,
                               "" + Letters.BACKSLASH +
                               userSettings.RecordDelimiter);
        } else {
            content = replace(content, "" + Letters.CR, ""
                               + Letters.BACKSLASH + Letters.CR);
            content = replace(content, "" + Letters.LF, ""
                               + Letters.BACKSLASH + Letters.LF);
        }
    }

    if (firstColumn && content.length() > 0 && content.charAt(0) == userSettings.Comment) {
    }
}
if (content.length() > 1) {
    content = "" + Letters.BACKSLASH +
    userSettings.Comment
        + content.substring(1);
} else {
    content = "" + Letters.BACKSLASH +
    userSettings.Comment;
}

outputStream.write(content);

if (textQualify) {
    outputStream.write(userSettings.TextQualifier);
}

firstColumn = false;

/**
 * Writes another column of data to this record. Does not preserve
 * leading and trailing whitespace in this column of data.
 *
 * @param content The data for the new column.
 * @exception IOException Thrown if an error occurs while writing data to the
 * destination stream.
 */
public void write(String content) throws IOException {
    write(content, false);
}

public void writeComment(String commentText) throws IOException {
    checkClosed();
    checkInit();
    
    outputStream.write(userSettings.Comment);
    outputStream.write(commentText);
    if (useCustomRecordDelimiter) {
        outputStream.write(userSettings.RecordDelimiter);
    } else {

```java
outputStream.write(systemRecordDelimiter);
}

firstColumn = true;
}

/**
 * Writes a new record using the passed in array of values.
 * @param values Values to be written.
 * @param preserveSpaces Whether to preserve leading and trailing spaces in columns while writing out to the record or not.
 * @throws IOException Thrown if an error occurs while writing data to the destination stream.
 */
public void writeRecord(String[] values, boolean preserveSpaces)
    throws IOException {
    if (values != null && values.length > 0) {
        for (int i = 0; i < values.length; i++) {
            write(values[i], preserveSpaces);
        }
        endRecord();
    }
}

/**
 * Writes a new record using the passed in array of values.
 * @param values Values to be written.
 * @throws IOException Thrown if an error occurs while writing data to the destination stream.
 */
public void writeRecord(String[] values)
    throws IOException {
    writeRecord(values, false);
}
```

```
* Ends the current record by sending the record delimiter. 
* @exception IOException 
* Thrown if an error occurs while writing data to the 
* destination stream. 
*/

public void endRecord() throws IOException {
    checkClosed();

    checkInit();

    if (useCustomRecordDelimiter) {
        outputStream.write(userSettings.RecordDelimiter);
    } else {
        outputStream.write(systemRecordDelimiter);
    }

    firstColumn = true;
}

/** 
 * Clears all buffers for the current writer and causes any buffered data 
 * to be written to the underlying device. 
 * @exception IOException 
 * Thrown if an error occurs while writing data to the 
 * destination stream. 
*/

public void flush() throws IOException {
    outputStream.flush();
}
/**
 * Closes and releases all related resources.
 */
public void close() {
    if (!closed) {
        close(true);

        closed = true;
    }
}

/**
 *
 */
private void close(boolean closing) {
    if (!closed) {
        if (closing) {
            charset = null;
        }

        try {
            if (initialized) {
                outputStream.close();
            }
        } catch (Exception e) {
            // just eat the exception
        }

        outputStream = null;

        closed = true;
    }
}

/**
 *
 */
private void checkClosed() throws IOException {
    if (closed) {
        throw new IOException(
            "This instance of the CsvWriter class has already been closed.");
    }
}
/**
 * 
 */

protected void finalize() {
    close(false);
}

private class Letters {
    public static final char LF = '\n';
    public static final char CR = '\r';
    public static final char QUOTE = '"';
    public static final char COMMA = ',';
    public static final char SPACE = ' ';  
    public static final char TAB = '\t';
    public static final char POUND = '#';
    public static final char BACKSLASH = '\\';
    public static final char NULL = '\0';
}

private class UserSettings {
    // having these as publicly accessible members will prevent
    // the overhead of the method call that exists on properties
    public char TextQualifier;
    public boolean UseTextQualifier;
    public char Delimiter;
    public char RecordDelimiter;
    public char Comment;
    public int EscapeMode;
    public boolean ForceQualifier;

    public UserSettings() {
        TextQualifier = Letters.QUOTE;
    }
}
public static String replace(String original, String pattern, String replace) {
    final int len = pattern.length();
    int found = original.indexOf(pattern);

    if (found > -1) {
        StringBuffer sb = new StringBuffer();
        int start = 0;

        while (found != -1) {
            sb.append(original.substring(start, found));
            sb.append(replace);
            start = found + len;
            found = original.indexOf(pattern, start);
        }

        sb.append(original.substring(start));

        return sb.toString();
    } else {
        return original;
    }
}
G. TESTING DOCUMENTATION

This appendix outlines the testing carried out during the development of the system.

G.1 Component Testing

The program involves several different pieces of software, so it was important to test the interaction between these components, to ensure they were all communicating properly.

First embedding the user information in the source code checked the connection between the Bloomberg API and the Java program.

Then the user entered data off the command line checked that same connection to ensure that changeable data would be accepted.

The original CSV File program tested the Java program’s communication with the company proprietary data system. When this was found to be successful the program was incorporated into the Java Request and Response program.

Each Java GUI was tested to ensure that the Java program was accessing the appropriate parts of the GUI and gaining the necessary information before proceeding to access the Bloomberg API and proprietary data system.
REFERENCES


BLUESEAL FINANCIAL LTD
Extraction of Bloomberg Data to study the effect of Economic Releases on different Asset Classes
User Manual
24-03-2014

Prepared by: Eve Anne Allen  Supervisor: Brett Houlding
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>NO.</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. OVERVIEW</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 About</td>
<td>1</td>
</tr>
<tr>
<td>2. MAIN PAGE</td>
<td>2</td>
</tr>
<tr>
<td>3. REQUEST GUI</td>
<td>3</td>
</tr>
<tr>
<td>3.1 Company</td>
<td>3</td>
</tr>
<tr>
<td>3.2 Start &amp; End Dates</td>
<td>4</td>
</tr>
<tr>
<td>3.3 Fields</td>
<td>5</td>
</tr>
<tr>
<td>3.4 Time Period</td>
<td>6</td>
</tr>
<tr>
<td>4. RESPONSE GUI</td>
<td>7</td>
</tr>
<tr>
<td>4.1 Data Retrieval &amp; Response</td>
<td>7</td>
</tr>
<tr>
<td>4.2 Write to a CSV File</td>
<td>8</td>
</tr>
</tbody>
</table>
1. OVERVIEW

1.1 Introduction

This user manual was created to help users understand the functionality of the program. The detailed manner in which it was written allows users of any level of technological skill to follow it.

1.2 About

This program was produced by Eve Anne Allen for BlueSeal Financial Ltd.
2. MAIN PAGE

The program is designed so that all necessary information is collected succinctly on the main page. This page will remain onscreen regardless of the other pages in the program that the users may be interacting with. This allows the users to send multiple data requests during a single session.
3 REQUEST GUI

3.1 Company

To create a new request, the user must enter a viable company’s equity in the text box, e.g. IBM US Equity; the Bloomberg API is case sensitive, however it follows the basic rule that all abbreviations are capital letters, as is the start of each word, everything else is lower case.
3.2 Start & End Dates

The start and end dates are a necessary part of the request as they state exactly which part of the Bloomberg API's historical data framework to access. As Bloomberg L.P is an American owned and operated company, the start and end dates are submitted in the “American style” i.e. YYYYMMDD rather than the “European style” of DDMMYYYY.

![Image of form with fields: Company: IBM US Equity, Start Date (YYYYMMDD): 20100101, End Date (YYYYMMDD): 20101231, Time Period: DAILY]
3.3 Fields

There are eight specific editable dropdown boxes for the field data in the program. This was stated by BlueSeal Financial Ltd. as the highest amount of fields they would ever need to access at any one time.

The user has two choices in how to input information into the “Fields” section. They can either choose from the list given (a list of the five most popular field choices):

Or they can manually enter a field they wish to access:
3.4 Time Period

The final piece of information required by the system is the time period. This is the amount of time between data releases that the user wishes to view. The Bloomberg API offers six different choices of time period, namely daily, weekly, monthly, quarterly, semi-annually, and yearly. These are the only viable options for the time period, therefore unlike the “Fields” section, the user is only able to choose one of the values from the dropdown box, and there is no option to manually enter a value for the time period.
4  RESPONSE GUI

4.1  Data retrieval and response

When the user clicks the “Results” button on the Request GUI, the Response GUI appears. If the user entered the following information and clicked the “Results” button:

![Example GUI](image)

The following Response GUI appears:

![Example GUI](image)

This GUI contains the relevant financial data given the users parameters. The date the data was originally released is also included to make it easier to view the changes in the market at the time the data was released.
4.2 Write to a CSV File

To write the financial data to a CSV File and store on the company's servers, the user must simply enter a title for the file in the textbox marked “Enter File Name” and click the button marked “Write to CSV”:

The file is saved to the company's servers in a pre-set destination folder created for the Bloomberg API data specifically.
When the CSV File is opened, it resembles the following:

![CSV File Example](image)

The file can be edited and manipulated in the same manner as any other CSV File in this format.