Reading Bloomberg AIM History Files With aimreader.py

Victory Capital |

Reading BLOOMBERG AIM HISTORY DUMP FILES (Last Updated: 7/25/2019)

msimmons@vcm.com

2019

Table of Contents

[Bloomberg AIM History Files 2](#_Toc14961273)

[Background 2](#_Toc14961274)

[AIM History File Structure 2](#_Toc14961275)

[Reading the AIM History files 2](#_Toc14961276)

[Implementation: aimreader.py 3](#_Toc14961277)

[Using aimreader.py 4](#_Toc14961278)

# Bloomberg AIM History Files

## Background

USAA Portfolio Managers and Fixed Income Traders used Bloomberg AIM for order management prior to the VCM acquisition close on June 30th, 2019. Bloomberg exported 8 years (Jan 1, 2012 – June 30, 2019) of USAA Asset Management Company’s (AMCO) historical transaction data into 8 data files and transmitted them electronically to USAA. USAA included these files in the Official Documents and Records that were turned over to VCM at deal-close. Each of the 8 files contains AIM data for 1-year dating from 2012 to 2019, with the 2019 file containing 6 months (Jan 1 to June 30).

All 8 AIM History dump files are stored here: K:\VCM\_USAA\_IMPL\Book and Records - Compliance\Bloomberg\

## AIM History File Structure

The structure of the AIM files is defined in Bloomberg’s *Trade History File Dump Spec.pdf*. These files are text files which contain one transaction (executed order) record per row with no header or trailer records. The files contain commas between *most* fields – but they are not comma separated value (CSV) files and cannot easily be read as such.

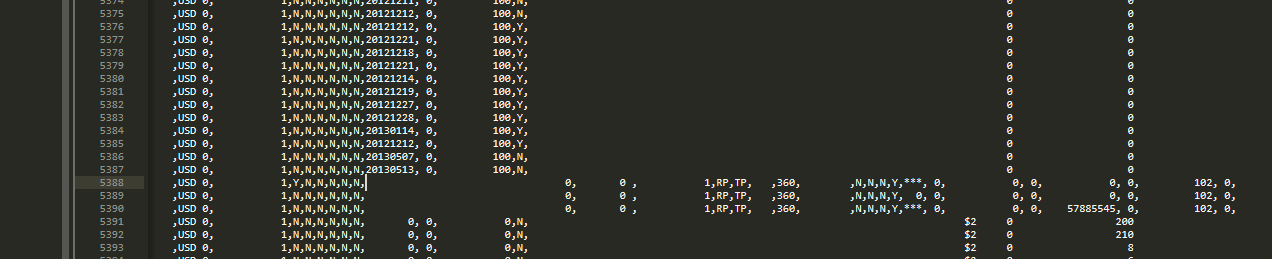


Figure 1: f2637dump.2012.0.ext sample data

Figure 1 is a sample of an actual AIM History file and illustrates how data “moves around” relative to commas. The reason for the “movement” is that Bloomberg is writing records with different definitions to the same output file. Bytes 0747 to 1533 can contain various layouts, but the bytes 0000 – 0746 and 2450 to the LF (CR LF on Windows systems) are relatively static.

## Reading the AIM History files

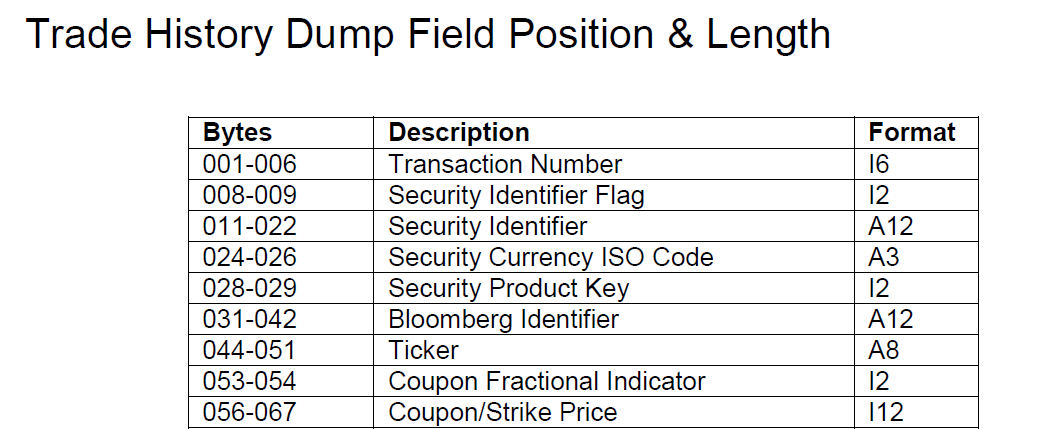
Per the Trade History File Dump Spec record layout documentation (Figure 2), the data fields in the file are defined by their byte locations. Finding data in the file is relatively easy by reading data byte by byte from the file. Understanding how to open and read files programmatically is not covered here since that process is well covered on the internet. Any programming language that allows File IO and SEEK operations will work for this. C, Java, .net, python and many other common languages all allow standard file access facility.

Figure 2: Trade History File Dump Spec.pdf

To read the data of interest, in this example lets read Ticker in bytes 044 – 051. Starting from the beginning if the file (byte 0), move a file pointer to byte 043 (044 –1), then read in 8 bytes. The file pointer will be at byte 051 when the read operation finishes. To read the next row, the file pointer needs to be moved to the end of the current row, the byte index of the last byte in the row needs to be stored, then the pile pointer needs to be advances to the next row. Note: in almost all programming languages, this is done with a readline() method. to read the next line’s Ticker, add (044-1) to the value stored in the variable containing the index of end of the previous row.

For example, if the row had 1000 bytes in it (Lf and CR LF characters are ignored here for simplicity) the algorithm to read the Ticker from each row in the file is:

Set index = 0;

While there are rows in the file to read {

Seek to the byte before Ticker (index + 43)

Read length of the Ticker (8 bytes) and store the data in a variable

Write the contents of the Ticker variable to an output file

Use readline() to move the file pointer to the next line

Use the current byte position -1 to know the length of the previous line, index = current position - 1

}

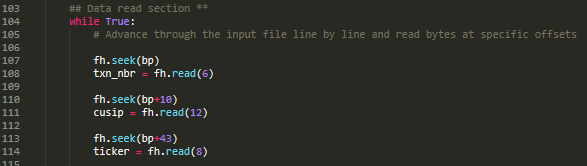
Any data at any byte position in the AIM History files can be read using this basic algorithm. Figure 3 shows the beginning of the while loop and code for reading transaction number, cusip, and ticker from the AIM History file. Note: “bp” is a “byte pointer” variable which is keeping track of the byte index as the code progresses through the lines in the input file.

Figure 3: aimreader.py code sample

## Implementation: aimreader.py

The implementation of this process has been done in python in a program named *aimreader.py*.

Usage: aimreader.py <input file> <ASOF date 'YYYYMMDD'> <output file (optional)>

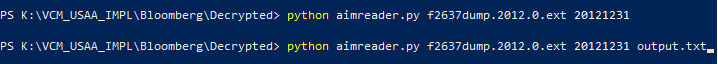


Figure 4: aimreader.py command and arguments

The first argument is required and is the name of the AIM history file you want to convert. Ex: f2637dump.2012.0.ext

The second argument is required and is the "ASOF Date" for the data in the file. It should be in YYYMMDD format, although it will take any string. This string will be added to the ASOF\_DATE at the beginning of every record and provides a way to identify all the data in a single load. This is the same functionality that an ETL number would provide and it can be used to back data out and reloaded in a database later.

The third argument is optional and is an output file name. If no name is specified, then ".dsv" is appended to the input file name and that becomes the output file name. Ex: f2637dump.2012.0.ext.dsv

The default output file delimiter is a pipe, but you can specify whatever you like in the delim = "|" at the top of the file aimreader.py file.

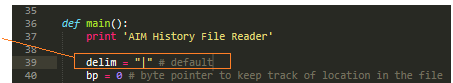


Figure 5: aimreader.py code for delimiter

## Using aimreader.py

The aimreader.py program needs the following to run:

1) Python 2.7 or greater needs to be installed on the machine where it will run.

2) A user path var needs to be set to the python.exe so it can run from mapped drives.

3) aimreader.py needs to be in the same folder as the AIM history dump file you want to read. At the time this was written we only needed to convert 8 files and there was not a lot more ROI to be had by handling file paths.

To run the program:

* Open a windows CMD window (or PowerShell in Win 10)
* Go to the folder with the Aim files, type in “python aimreader.py <input file> <data date> and hit enter

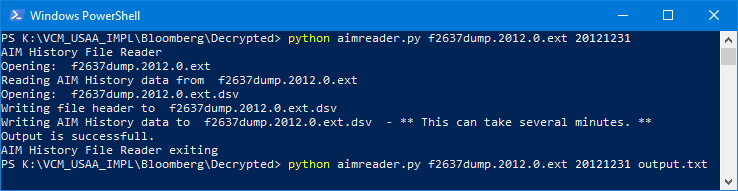


Figure 6: aimreader.py execution and console output

If the run is successful, then the “Output is successful.” message will display. If there are any errors, then an error message as to the cause of the error will display, along with brief info on what to do to fix the issue.