Binary Separated Value

B. Bean, N. Mezher, J. Summers & D. Toomey
University of Massachusetts Lowell — Software Engineering II
Prof. James Daly

April 2020

Motivation

The use of databases as a means for storing information has become ubiquitous in every field concerning data. One of the most common methods for analyzing sets of data from databases is to export it to a `.csv` (comma separated value) file format in order to manipulate the data via a spreadsheet program or language libraries. But despite all its draws, the `.csv` format has some significant drawbacks as well. The format is too bulky and inefficient for many applications, and it relies on a comma delimiter to separate data which can be problematic [1]. Our improvements upon the `.csv` format allow for users and programs to more efficiently store and utilize complex data. The end goal is to expedite communication between programs and disparate systems.

This document first outlines a new file format termed Binary Separated Value (BSVX), with the file extension `.bsvx`. This is not to be confused with the `.bsv` file format, which is a BASIC BSave Graphics file. The tailing `x` was chosen for convenience as it makes the name of our format, `.bsvx`, wholly unique. This format of data is delimited with byte markers which begin each field telling the library what kind of data will be in the field and how long it will be. Using byte markers, instead of plaintext character delimiters, solves a key issue with the `.csv` format—strings including commas do not prematurely end a field. The Binary Separated Value format is processed through a proprietary Python library called BSVXPY.

One drawback of this style of implementation is the inability to parse and edit `.bsvx` files through a text editor. However, this issue is remedied through our proposed BSVX LibreOffice Calc Extension. LibreOffice is a free to use, open-source file editing platform similar to Microsoft Office. LibreOffice Calc is a program provided in the LibreOffice suite, and provides similar functionality to Microsoft's Excel program [4]. The BSVX LibreOffice Calc Extension would give LibreOffice Calc users the ability to read data from `.bsvx` files and export their spreadsheets to `.bsvx` files. Users would also have the ability to import, then convert `.csv` files into `.bsvx` files through the BSVX LibreOffice Calc Extension.

BSVX File Format Specification

Each `.bsvx` file contains a series of rows of headers or records. Each row begins with a byte marker denoting the type (i.e. header or record) and the number of fields within that row. Following the first byte marker of a row is a series of fields, each made up of two parts: a byte marker denoting the type and size of the data stored within the field, and the data itself. Some initial markers indicate that the size of the data is given in subsequent bytes. Once the length n is determined, those n bytes can be interpreted to match the field byte marker. Each row does not have to be the same length, the data can be jagged and parsers read as much data as is denoted by the first byte marker of each row.

At any time, the parser knows how many bytes it needs to read. There is never an instance where the parser needs to read bytes until it sees a particular character (as opposed to `.csv` or `.tsv` parsers, which look for commas or tabs respectively). Strings need neither end marks nor escape characters, and are stored in the UTF-8 format. The byte marker for strings denotes the number of bytes read, not the number of characters of the string itself. All numbers are stored in little endian order.

An abstract example of a `.bsvx` file row (header or record) looks like this:

3 Field	3 str	FOO	2 int	1000	4 Float	25.345

Table 1: An example of a `.bsvx` file row.

The same example of a `.bsvx` file row (header or record) but represented in hexadecimal:

0xAB Record	0x03 str	0x464F4F	0x91 int	0x01F4	0x9C Float	0x400395851EB851EB851EB851EB851EB8

Table 2: An example of a `.bsvx` file row in hex values.

The following table displays the implementation for each type of supported data in its own class. Bit ranges for each field are also provided; they are denoted by values ranging from 0 to 255. The first column gives the parser crucial context: what type of data follows the byte marker, and further, which *variant* on that type it is. For example, the short integer type is represented by numbers in the range 136-143. A 2 byte short integer is indicated by 138, 139 indicates a 3 byte integer, 140 indicates a 4 byte integer, etc. The second column illustrates how the range of values for a given type is affected by the magnitude of its offset. I.e. for a short integer, the second column entry is 136 + [0, 7]. The third column establishes the types of data that are supported, and the fourth column provides a brief description of each.

Range	**Form**	**Name**	**Description**
0		Blank	Possible implementation: NULL or 'empty string'
1-127	1-127	Short str	UTF-8 Encoded string of byte length 1-127
128-135	128 + [0,7]	Long str	1-8 bytes giving the length of a str, followed by said str
136-143	136 + [0,7]	Short int	An integer in the range of 0-7 bytes
144-151	144 + [0,7]	Long int	A zig-zag encoded integer using 1-8 bytes
152-159	152 + [0,7]	Float	IEEE-754 format float: 0 = half precision, 1 = single, 2 = double, 3 = triple
160-167	160 + [0,7]	Blob	1-8 bytes giving the length of binary data in bytes, followed by said data
168-183	168 + [0,15]	Header	Beginning of header with 0-15 fields
184-191	184 + [0,7]	Long header	1-8 bytes giving the number of fields in the header
192-207	192 + [0,15]	Record	Beginning of record with 0-15 fields
208-215	208 + [0,7]	Long record	1-8 bytes giving the number of fields in the record
216-255		Reserved	For future use

Table 3: Data types supported by the BSVX file format specification.

Deliverables

The deliverables for this new file format include the aforementioned BSVX LibreOffice Calc Extension and BSVXPY Python library. The LibreOffice Calc extension should be capable of ultimately converting between `.csv` and `.bsvx` files without loss or adulteration of information. Similar libraries for languages such as Java, C++, or JavaScript are left as stretch goals.

The BSVX LibreOffice Calc Extension would allow spreadsheets to be saved to and read from `.bsvx` files. To illustrate the top-most point for user interaction with the BSVX LibreOffice Calc Extension, there is included a series of figures below. Figure 1 displays the default toolbar packaged with LibreOffice Calc. Figure 2 contrasts the differences between the default toolbar and a toolbar with the BSVX LibreOffice Calc Extension enabled. It can be seen that only two features are to-be added, in the form of two buttons. The proposed left button allows for importing, reading from, a `.bsvx` file and the proposed right button for exporting, saving to, a `.bsvx` file. Finally, Figure 3 provides a glance as to how the toolbar would look with the BSVX LibreOffice Calc Extension enabled.

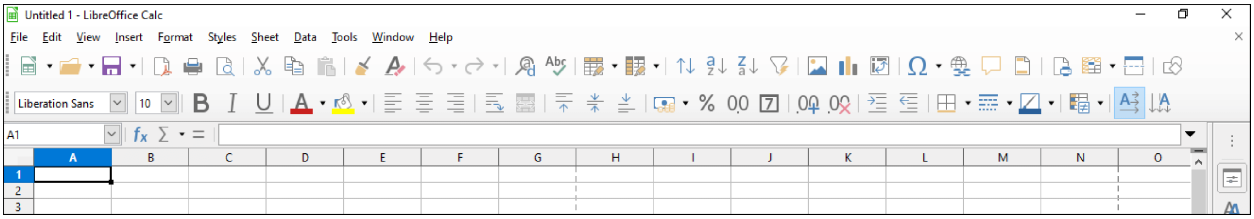

Figure 1: LibreOffice Calc's toolbar.

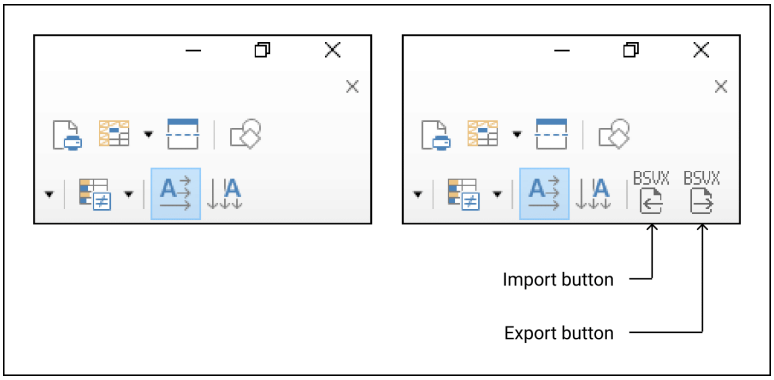

Figure 2: The BSVX LibreOffice Calc Extension would provide two additional buttons for importing and exporting.

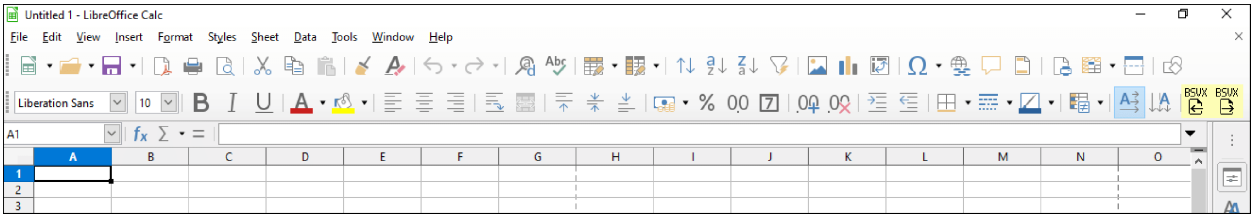

Figure 3: A mockup of LibreOffice Calc's toolbar with the BSVX LibreOffice Calc Extension enabled.

To talk more about how the BSVX LibreOffice Calc Extension would work behind the scenes, it is first necessary to speak about our Python library—BSVXPY. As a generality, our Python library is similar to the `.csv` Python library. A writer function is passed a series of fields representing a header row. Subsequent binary values are decoded based on the corresponding type casts provided by the header. The library then extracts each of the fields from the dictionary object and outputs them to the LibreOffice Calc spreadsheet in sequential order. The library also processes nested data structures within the `.bsvx` file, allowing for the recursive encoding and decoding of further dictionary objects.

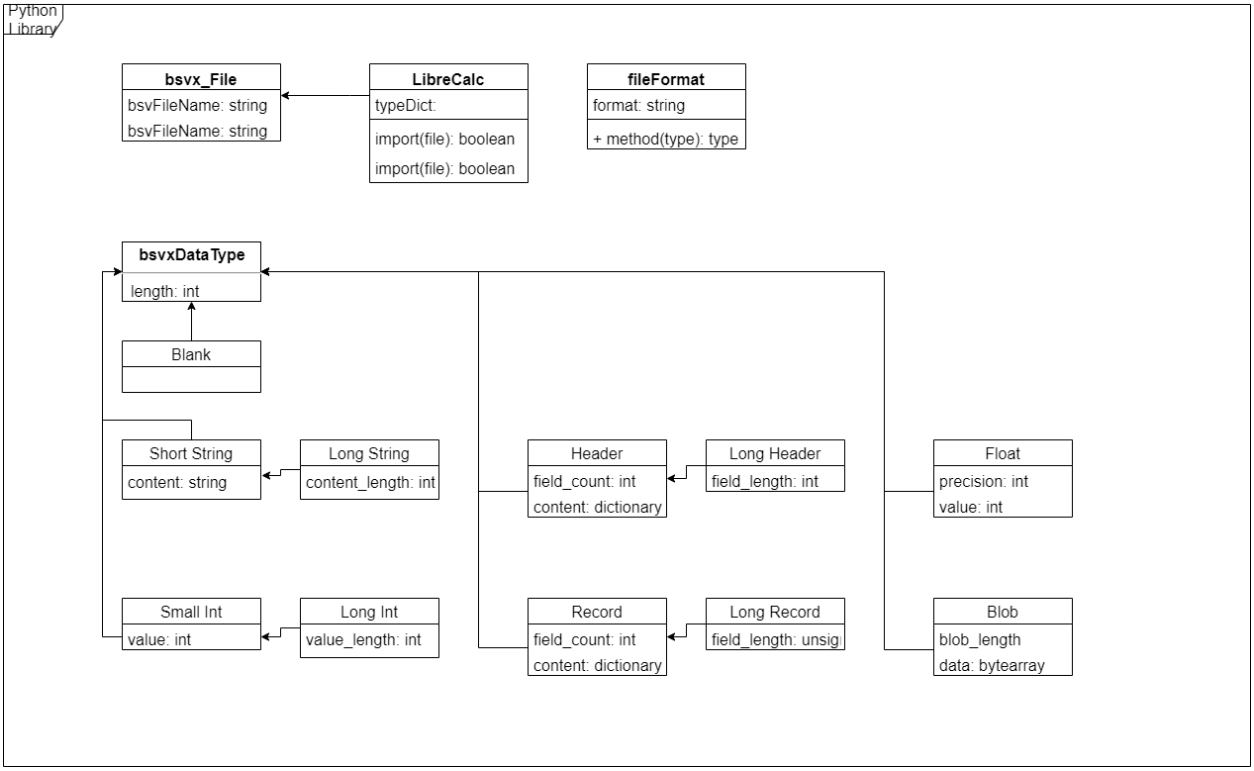

Figure 4: UML diagram outlining the classes and statstructures used in BSVXPY

With an understanding of how the BSVXPY Python library functions, we can return to an overview of the BSVX LibreOffice Calc Extension. First off, the LibreOffice Calc project allows developers to create extensions using Python, which would let us extend LibreOffice Calc's functionality to include `.bsvx` file format support using our BSVXPY Python library. We would also use the UNO Python library, as it is necessary for any LibreOffice Calc extension development.

A novel problem with a strongly typed encoding is the inability to handle undefined data types such as graphs or algorithms that are often used in spreadsheet applications. To solve this, the `.bsvx` file format manages unknown data types with a catch-all data type—Blob—which acts as a polymorphic object. These Blobs store raw binary data imported from spreadsheets or `.csv` files. By relying on the raw binary for unknown data types, BSVXPY can accommodate proprietary encodings associated with any third party application. As a note, this introduces the risk of `.bsvx` files being limited to one spreadsheet application when using Blob data types. This is because third party applications may use internal encodings which are unknown to other programs.

To export data to a `.bsvx` file, the BSVX LibreOffice Calc Extension would call functions from the UNO Python library to read cell data from LibreOffice Calc. The BSVXPY Python library would then be used to convert that data into binary separated values. Once the data is appropriately converted, it would be written to a file with the extension `.bsvx` and as named by the user. Figure 5 depicts the data flow for the exporting feature.

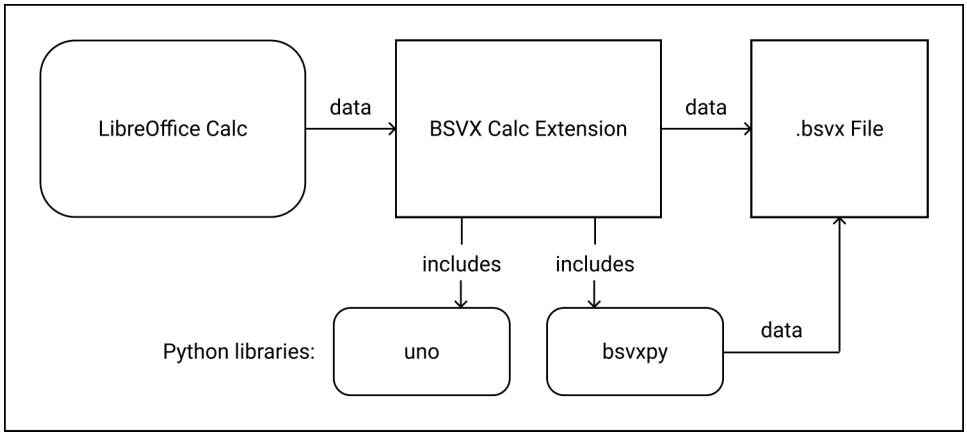

Figure 5: Scope of API and library calls for the BSVX LibreOffice Calc Extension in exporting data to a `.bsvx` file.

The importing feature would work the same way, but in reverse. The user would select a `.bsvx` file to import data from, and the BSVX LibreOffice Calc Extension would read data from that file, using BSVXPY and UNO to translate it from binary separated value data to cell data that LibreOffice Calc can read. The data flow for the importing feature is represented by Figure 6.

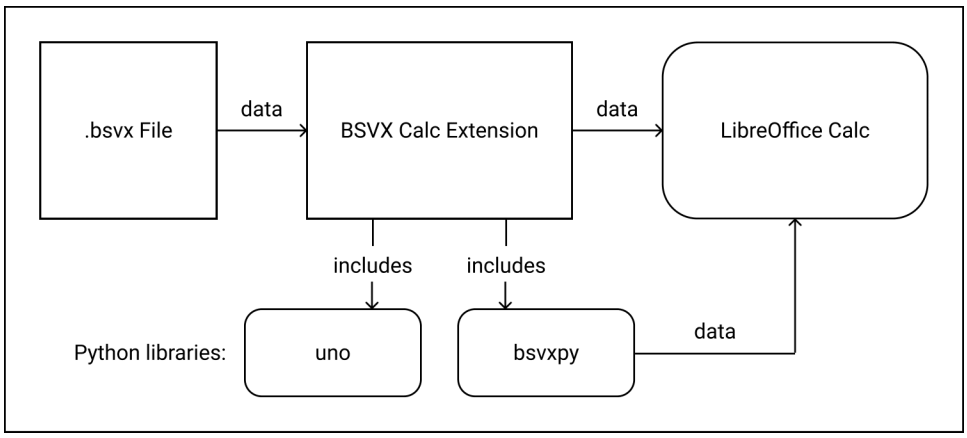

Figure 6: Scope of API and library calls for the BSVX LibreOffice Calc Extension in importing data from a `.bsvx` file.

This functionality would also allow the user to convert from `.csv` to `.bsvx`. If the user chose to import a `.csv` file into LibreOffice Calc, and export that file to a `.bsvx` file, that same data would become available in both files. Likewise, if the user chose to import a `.bsvx` file and export that file to a `.csv` file, that same data would become available. These changes would not significantly affect the overall architecture of LibreOffice Calc. The same base functionality of LibreOffice Calc would still be provided to the user once this extension was installed, just with the addition of importing and exporting features for the `.bsvx` format.

Interface

The current implementation of BSVXPY includes interface classes to hold data, as well as providing the user with the ability to read and write using files.

The `Reader` class is designed to take in any `.bsvx` file and create a map of BSVXPY classes for the user to operate with. This is done by reading in the hexadecimal (hex) text of a `.bsvx` file and iterating through the data, creating classes as it traverses. When the file is first opened, the class reads in the first two hex

characters on the first line to know how many fields to read in. A loop is then called to iterate over `i` columns of data. Each record contains hex values telling the `Reader` what class it is. `Reader` then creates a `bsvxDataType` class for the corresponding type by passing in the hex representation of the data to the class' constructor. Once the class is instantiated, it is appended to a map containing the string representation of the data as well as the class itself.

`Writer` is a class designed to write any `bsvxDataType` object to a file. Its implementation is simple, and is mostly handled by the `bsvxDataType` subclasses. Each subclass contains its own unique `write()` function, which is implemented differently depending on how the data is formatted. Each class you pass to `Writer`'s `writerow()` function will be written to the file in that order.

Approach

As the specifications for the `.bsvx` format have already been articulated, our main goal was to implement novel importing and exporting functionality in LibreOffice Calc. Example `.bsvx` files have been created to accurately account for testing basic functionality as well as edge cases. Initial builds emphasized basic functionality, such as correctly reading and writing basic data types. Nested and blob types were handled after initial testing was completed. In every step of the process, we attempted to optimize memory and computational efficiency to improve our performance outcomes.

One important test of the `.bsvx` format, and its corresponding extension, was its capacity to maintain full integrity of the data after conversion to and from the `.csv` format. A user with minimal experience in LibreOffice and some experience with spreadsheets should be able to perform these conversions without loss of data or corruption of its ordering. We accepted some loss of formatting and style, provided the order of the values and the values themselves were maintained.

The project was expected to take approximately two months to complete, with production wrapping in late April 2020. Costs were minimal, if nonexistent, as our developers were paid in "experience."

Development

In developing the BSVXPY Python library, our team took a number of precautions. At the highest level, our project was split into three different repositories on GitHub, so that BSVX's components were entirely separate. The BSVXPY repository holds our BSVXPY Python library. The BSVX4CALC repository would hold our BSVX LibreOffice Calc Extension. Lastly, the DOCS repository contains efforts related to this document and generating the results shown later in this document.

To further isolate BSVX, the development of the BSVXPY Python library required the use of a Python virtual environment. This ensured that the user did not install the incomplete module on their default Python environment. Specifics regarding the initialization and launch of a Python virtual environment, within the context of this project, can be found in the README portion of the BSVXPY repository.

To ensure correctness, to the best of the BSVX developers, Travis-CI was utilized. Travis-CI allows for integration with GitHub repositories, and helped to automatically test BSVX's components. As for the tests themselves, the PYTEST package was utilized. The PYTEST package allowed the BSVX developers to easily and quickly construct unit and systems tests for the BSVXPY Python library. Different testing files were associated for each data type supported by the BSVX file format specification so that each component's functionality could be tested separated if needed. If all tests that were tested—typically all tests created—passed, Travis-CI would assign our repository a passing mark. Failing Travis-CI marks enabled our developers to retroactively fix problems with some context as to where they happened. This continuous integration process made development safe and incremental.

Schedule and Milestones

Date	Goal
February 14, 2020	Finish initial background research, Draft BSV Proposal deliverable.
February 26, 2020	Finalize project's architecture, draft Project Architecture deliverable
March 4, 2020	**Midterm Milestone:** Present progress and evaluate stretch goals
March 18, 2020	Implement csv/bsvx backend
April 1, 2020	Draft the Initial Results deliverable
April 27, 2020	**Final Milestone:** Submit Final Report and Present findings to class

Table 4: A schedule of our project's events.

Challenges and Risks

One of the primary risks was the possibility that there were undiscovered ambiguities in the format specification. These were dealt with by tightening the specification to account for ambiguities, and updating the reference implementation. Additionally, parsing `.csv` files for conversion to `.bsvx` files, and vice-versa, involved numerous pitfalls. While there was an agreed upon standard format for `.csv` files, it didn't come about until 2005 and many `.csv` files still did not conform to it strictly. This complicated our attempt to ensure integrity and continuity between conversions for *all* `.csv` and `.bsvx` files. For instance, when converting a `.bsvx` file consisting of several strings of comma characters, our library had to ensure that none of the commas ended up being misinterpreted as delimiters. Properly following the specification ensured consistency and prevented this from happening, but rigorous testing with a myriad of files was necessary.

One challenge fundamental to the `.bsvx` format itself was deciding how to handle Blob objects. It is not always clear what type of data a Blob should be deserialized as. We took the stance that the user should have context of the Blob's contents, so anything outside of the confines of the BSVX specification is not BSVX's responsibility. LibreOffice Calc may contain methods to interpret unknown fields upon reading the file but more research on this is necessary. Another challenge was that the framework/API for both LibreOffice and LibreOffice Calc was unfamiliar to our developers; it took them some time to learn how to engage with LibreOffice Calc.

Results

In diagnosing the successfulness of our project and of BSVX's subcomponents themselves, we created a script that generates a number of key figures. This script can be found in the scripts folder of our `docs` repository along with usage instructions in the form of a README. We offer this means of figure self-generation such that it is clear and self-provable that all figures in this document are fair.

An aspect integral to BSVX is the compression of information attained through storing fields in hexadecimal. In comparing relative sizes of equivalent `.csv` and `.bsvx` files, it is demonstrably clear that `.bsvx` files are smaller in file weight.

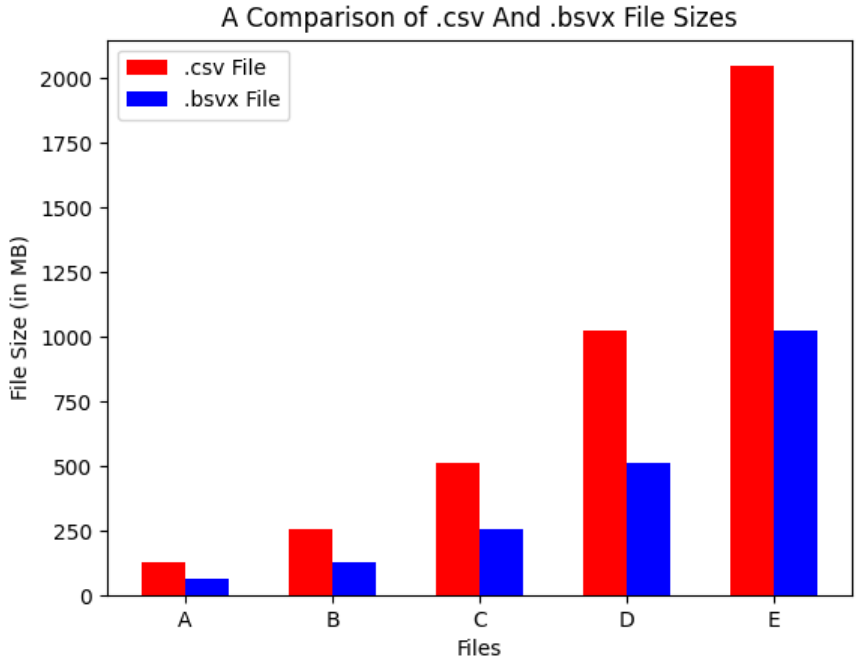

Figure 7: A simplified version of a graphic that will be in the final version of our report.

Comma delimiter issues are another problem BSVX sought to solve. Displayed below is a graph of expected versus actual fields for equivalent `.csv` and `.bsvx` files. `.bsvx` files match the expected number of fields, whereas `.csv` files decidedly do not. In fact, `.csv` files develop an increasingly larger margin of error with larger file sizes.

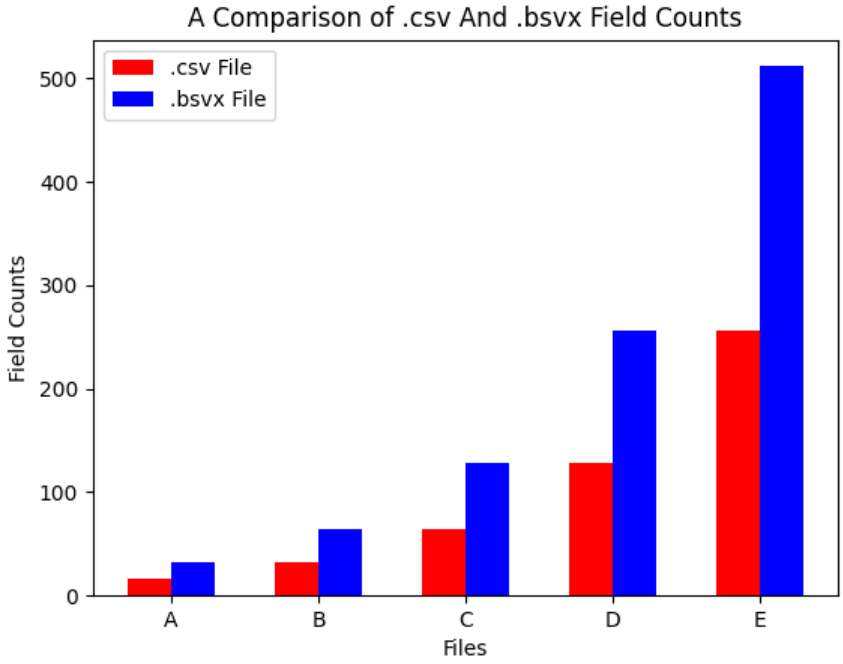

Figure 8: A simplified version of a graphic that will be in the final version of our report.

From here we can see that the two main problems with `.csv` files, as previously noted, have been solved with our implementation of the BSVX file format specification.

Discussion

In the original product specification, the goal of BSVX was to create a new Python library rivalling the Python CSV library all while providing the same functionality set. This included handling data with Python, writing and reading to and from files, and providing the user with a LibreOffice Calc extension that could be used to import data to and from spreadsheets.

However, we—like most everyone else—did not foresee the COVID-19 pandemic. With the pandemic's timing, we were forced to change our implementation plan. And with the team's shift to remote development, significant working time was lost. In the beginning of the shift, progress was halted on most repositories. New communication channels had to be created too, which further cut into development time. As a result, the BSVX team decided to no longer pursue development of the BSVX LibreOffice Calc Extension. We believed it necessary to limit the scope of our project to maintain code quality. The BSVX LibreOffice Calc Extension had been a stretch goal for our team regardless, and the team thought it best to diverge resources towards the BSVXPY Python library. An outline for the implementation of a BSVX LibreOffice Calc Extension remains in this document, and is freely open for development.

Glossary

.bsvx The file extension associated with the Binary Separated Value file format.

.csv Comma-separated values file format often used for databases and spreadsheets.

.tsv Tab-separated values file format used for databases and spreadsheets.

Binary Separated Value The file format associated with the encoding protocols laid out in this document.

Comma delimiter Practice of using the ‘,’ character as a field separator to differentiate records in a file. Instances of a comma are always interpreted as a delimiter unless they appear in doubles quotes e.g. “1,0”.

Deserialization The process of decoding a `.bsvx` byte stream into its original data.

LibreOffice Calc An open-source application for manipulating spreadsheets. Developed and maintained by The Document Foundation.

Serialization The process of encoding data into a `.bsvx` byte stream.

References

[1] Coleman, Larry. "Why do we keep using CSV?" Software Engineering Stack Exchange, 14 Feb. 2011. https://softwareengineering.stackexchange.com/questions/47838/why-do-we-keep-using-csv

[2] "CSV File Reading and Writing." Python Docs, 12 Feb. 2020. https://docs.python.org/3/library/csv.html

[3] Daly, James and Meiners, Chad. "Binary Separated Value." UMass Lowell, 30 Jan. 2020.

[4] Guthrie, Gordon. "How to Work With LibreOffice Calc." TechRadar, 23 July 2012. https://www.techradar.com/news/world-of-tech/roundup/how-to-work-with-libreoffice-calc-1089870