Binary Separated Value Proposal

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

February 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` file 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 will 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 will outline the proposal of a new format called Binary Separated Value (`.bsvx`). This format of data will be 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. Instead of using a character delimitation in plaintext form, using a byte marker solves the issue of having commas in string fields, avoiding parsing errors. One drawback of this style of implementation is the inability to parse and edit `.bsvx` files through a text editor, but this issue is remedied through the BSVX LibreOffice Calc Extension. The Binary Separated Value format will be processed through a proprietary Python library.

LibreOffice is a free to use, open-source file editing platform similar to Microsoft Office. Calc is a program provided in the LibreOffice suite, and provides similar functionality to Microsoft's Excel program [4]. The proposed BSVX Calc Extension will give Calc users the ability to read data from a `.bsvx` file, and export their spreadsheets in `.bsvx` format. This will also allow users the functionality of converting a `.csv` file into a `.bsvx` file through the Calc program.

BSVX Approach

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 will indicate that the size of the data will be 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 will only 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 will never be an instance where it needs to read bytes until it sees a particular byte (like in the case of `.csv` or `.tsv` looking for a comma or tab to end on). Strings will neither need end marks nor escape characters, and will be stored in 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

The following chart will be used to implement each type of supported data in its own class and denote the bit range each field will be denoted by, ranging from 0 to 255. The first column gives the parser crucial context: what type of data will follow the byte marker, and further, which *variant* on that type it will be. For example, the small integer (int) type is represented by numbers in the range 136-143. A 2-byte small int is indicated by 138, 139 is for a 3 byte int, 140 denotes a 4 byte int, 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 small int, the second column entry is 136 + [0, 7]. The third column establishes the types of data that are supported and the fourth 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]	Small 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

Deliverables

The proposed deliverables for this new format include a LibreOffice Calc extension to read from and write to .bsvx files, and a python library to for doing the same. This extension should be capable of converting between `.csv` and `.bsvx` without loss or adulteration of the information stored in the files. Similar libraries for languages such as Java, C++ and C#, or JavaScript are left as stretch goals.

The Calc extension will allow spreadsheets to be saved to and read from `.bsvx` files. In most cases, the first row of the spreadsheet will be saved as a header, and subsequent rows will be saved as records. When reading, both headers and records will be read the same, but may affect whether a row is included in sorting.

Our Python library will be similar to the `.csv` library. A writer function will be 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 will then extract each of the fields from the dictionary object and output them to the Calc spreadsheet in sequential order. The Library will also process nested data structures within the `.bsvx` file allowing for the recursive encoding/decoding of further dictionary objects.

Advanced options allow for serializing objects directly, accessing their fields rather than first converting the object to a dict and serializing the dict.

Versions of this library for other languages will provide similar functionality to the Python library and will use reflection to support serializing and deserializing objects directly. Added support for reflection allows for the `.bsvx` filetype to encode more complex data structures and data types not explicitly encoded in the `.bsvx` specifications. Similarly, child objects will be serialized as a blob whose contents are themselves `.bsvx` files.

Approach

As the specifications for the `.bsvx` file format have already been articulated, our main goal is to implement novel import/export functionality in Libre Calc. Example `.bsvx` files will be manually created to accurately

account for testing basic functionality as well as edge cases. Initial builds will emphasize basic functionality such as correctly reading/writing basic data types, and will move to nested and blob types after initial testing is completed. Lastly, we'll attempt to optimize memory and computational efficiency to improve performance outcomes.

One important test of the `.bsvx` format and its corresponding extension will be its capacity to maintain full integrity of the data after conversion to and from `.csv` format. Our deliverable should enable a user with minimal experience in LibreOffice and some experience with spreadsheets to perform these conversions without loss of data or corruption of its ordering. Some loss of formatting and style is acceptable as long as the order of the values and the values themselves are maintained. The point of `.bsvx` is not aesthetics, it is purely function.

The project is expected to approximately two months to complete, with production wrapping in late April 2020. Costs are to be kept minimal, if nonexistent, as our developers are being paid in "experience."

Schedule and Milestones

Date	**Goal**
February 19, 2020	Finish background research, draft Object and Interaction UML diagrams
February 26, 2020	Finalize list of required development tools, testing environment, lock LibreOffice version for internal development Decompose development steps and and assign tasks
March 4, 2020	Finalize and submit our User Manual
March 6, 2020: Midterm Milesome	Draft LibreOffice Calc interface Draft .bsvx interpreter with the expected functionality of converting .csv to .bsvx file type (one way)
March 18, 2020	Finalize Calc interface Draft .bsvx import functionality (.bsvx to .csv file type)
April 1, 2020	Merge Calc interface and .bsvx interpreter with stretch goal of eliminating .csv middle-step
April 17, 2020	Draft Final Report of findings
April 27, 2020: Final Milestone	Submit Final Report and Present findings to class

Challenges and Risks

One of the primary risks is the possibility that there are undiscovered ambiguities in the format specification. These will be 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, could involve numerous pitfalls. While there is an agreed upon standard format for `.csv` files, it didn't come about until 2005 and many `.csv` files still do not conform to it strictly. This will complicate our attempt to ensure integrity and continuity between conversions for *all* `.csv` and `.bsvx` files. For instance, when converting a a `.bsvx` file consisting of several strings of comma characters, our library would have to ensure that none of the commas end up being misinterpreted as delimiters. Properly following the specification will ensure consistency and prevent this from happening, but rigorous testing with a myriad of files will be necessary.

One challenge fundamental to the `.bsvx` format itself is deciding how to handle blob objects. It isn't always clear what type of data they should be deserialized as. Perhaps Calc has a method to interpret unknown fields upon reading the file; more research will be done on this. Another challenge is that the framework/API for both LibreOffice and Calc is unfamiliar, and it will take some time to learn.

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.

Binary Separated Value The filetype 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 of a .bsvx byte stream and reconstruction of the original data.

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

Serialization The process of encoding an object as a .bsvx format 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] James Daly 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