e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:04/Issue:04/April-2022 Impact Factor- 6.752 www.irjmets.com

BSF: A GENERAL PURPOSE BINARY SERIALIZATION LIBRARY IN C

Shreyas Patil [*1], Bhushan Patil[*2], Himanshu Chadha[*3], Akanksha Akre[*4], Dr. Archana Mire[*5]

[*1,2,3,4,5]Department Of Computer Engineering, Terna Engineering College, Nerul, Navi Mumbai, Maharashtra, India.

ABSTRACT

We often find ourselves in situations where we need to save the state of an object beyond the lifetime of the application in which it is used. To achieve this goal, we use a technique called serialization. During this serialization process, the object's state is converted to a stream of bytes and then written to a file or sent over the network. This data can be de-serialized when needed. When de-serialized, it creates an exact clone of the object created. To solve this tedious problem of implementing manual serialization and de-serialization, we present BSF, a simple and efficient C header library for serializing and deserializing data. The BSF library allows users to easily serialize and de-serialize primitive and user-defined data types. This header-only library is implemented in the C programming language. BSF does not depend your software on the library and you can compile the source code (single file) directly into your own program. This document describes the technical details and implementation of the Universal Binary Serialization Library. A small demo and some example programs are provided in the Implementation section to demonstrate the use of this library.

Keywords: BSF, Serialization, De-Serialization, Endianness.

I. INTRODUCTION

In modern times we often save the status of the object in the Storage environment, and later reproduce the correct copy on the stage and to enable them to send an object to another application domain in one application domain. To achieve this goal, we use techniques known as serialization. In the calculation, serialization is a format that can store a data structure or object state (for example, in a file or memory data buffer) or transport (such as a computer network) and later reconstructed processes. In another computer environment). If a set of a series of bits are listed according to the serialization format, it can be used to generate the same replication of the source object. For many complex objects, such as creating a wide range of links, this process is not easy. This process of serializing objects is sometimes referred to as sorting. The reverse operation of extracting a data structure from a sequence of bytes is also known as unmarshalling.

Binary serialization is the process by which the state of an object is saved to a storage medium in binary format. When an object is serialized, its data is converted to a binary format containing a stream of bytes. This stream of bytes can be stored on any storage medium such as a file, disk, etc. or sent from one system to another over a network. Whenever you need this object, you can always convert the same stream of binary bytes back to the original object. The basic mechanism is to flatten the object(s) into a one-dimensional bit stream and replace that bit stream back to the original object(s). As with the Transporter of Star Trek, it converts to a complicated and flat sequence 1s and 0, then taking this sequence 1s and 0 (perhaps elsewhere to other times) and reconstructed something that is implemented by various programming frameworks. Other methods of serialization are provided. The two most important reasons for storing the status of an object in a storage environment can be re-created the correct copy and send the value of one application domain to the value of another application domain. C is considered a procedural computer programming language in consideration and is used as a building unit of many other programming languages. This is a trusted language because it provides a variety of types of data and operators to provide a huge platform for performing all types of tasks. C is very flexible. Or you could call it machine independent. This helps your code to run on any machine without any code changes.

BSF is a library implemented in the C programming language, designed to allow users to serialize and de-serialize data in a simple and efficient way. Data is stored in natural binary format. The API is small and tries to stay "out of the way". BSF can serialize many C data types, including structures.

II. LITERATURE REVIEW

The paper [1] "Objects serialized and disagreement with XML" Interference is possible, which is potentially enabled that the heterogeneous database has been a certain research issue for years in the database community. The old cooperation system tends to globalize the location and subsequent requirements of data and data access for the coexistence of new data stores, and data transmission between data stores is becoming increasingly important. The appearance of the markup language (XML), which is an independent representation of the database in the database (XML), provides the appropriate mechanism for transmitting data to the repository. This article describes the research activities of the CERN group in connection with the definition and implementation of serialization methods and database deserialization methods that can be used to replicate or migrate objects from the network between the CERN and the global center between the CERN and the global center. Contents of several objects. Residents in object-oriented databases.

In [2] "Objective Association of Net Framework" The author explains the use of serialization on the network. We describe the two most important reasons. Saving the state of an object on a storage medium so that an exact copy can be recreated at a later stage, and by value, sending the object from one application domain to another. It is also used remotely to pass objects by value from one application domain to another. This article provides an overview of serialization used in the Microsoft NET Framework.

In [3] presents a general pickling and minimization mechanism provided as a service similar to garbage collection. Pickling is used to externalize and internalize data. Minimization means sharing arbitrary data structures as much as possible. This paper introduces the concept of abstract storage as a formal basis for algorithms and analyzes design decisions for implementation aspects of minimization. The mechanism presented here is fully implemented in Alice's programming system. A general minimization mechanism is presented. We showed how Alice, a conservative extension of Standard ML, can use pickling in a type-safe way for a system of components. We introduced abstract storage as a general-purpose memory model to build a formal foundation for our algorithms. Unpacking and pickling is based on this model, which allows you to analyze and evaluate design decisions such as bottom-up or top-down unpacking, right-to-left or left-to-right traversal. Minimize can be used to reduce the size of pickled data. However, the general mechanism presented here appears to be suitable for other applications, such as efficiently representing types at runtime. Finally, we scaled the system with concurrency support in Alice.

In [4], the author explains why object serialization is not suitable for persistence in Java. Numerous code examples initially show serialization of 48 objects as easy to use, enticing developers to rely on them for robustness in more complex applications. Advanced uses of object serialization require considerable work from the programmer and are not obvious at first. Using the serialization of objects with static and temporary fields and discusses with "large characters of suction" within the framework of the multi-threaded program. Before processing, you need to be read from all chart objects. This article shows many support examples that guarantee the patience of inappropriate objects using a serialization mechanism of Java objects. The system looks simple on the surface, but there is a lot of results as a patience of patience. Programmers must specify the type with patience with patience for patience during compilation, while adopting this determination during execution.

This paper [5] compares six approaches to serialization of objects in high quality and quantitative aspects. Object serialization in Java, IDL, XStream, Protocol Buffers, Apache Avro, and MessagePack. With each approach, a generic example is serialized into a file and the file size is measured. Explore qualitative comparisons in terms of whether a schema definition is required, whether a schema compiler is required, whether serialization is ASCII or binary-based, and supported programming languages. It is clear that there is no better solution. Each solution is suitable for the context in which it was developed.

III. DESIGN AND ANALYSIS

Serialization is a process for converting data objects that exist in complex data structures to byte streams for deployment, transmission, and deploying physical devices. Computer systems may vary depending on the hardware architecture, OS, and addressing mechanisms. Internal binary data representations depend on each environment. Storage and data exchange between these different environments require the type of platform and language platform. All systems are reverse processes that create objects from the byte sequence, which is the desire process that is the opposite process, which is the opposite process that is transported to the target system only on the target system, only the serialized data. Reconstructed objects are the original object's clone. The choice of data serialization format for an application depends on factors such as the complexity of the data, the need to be human-readable, rate limitations, and storage space. XML, JSON, BSON, and YAML are the most commonly used data serialization formats.

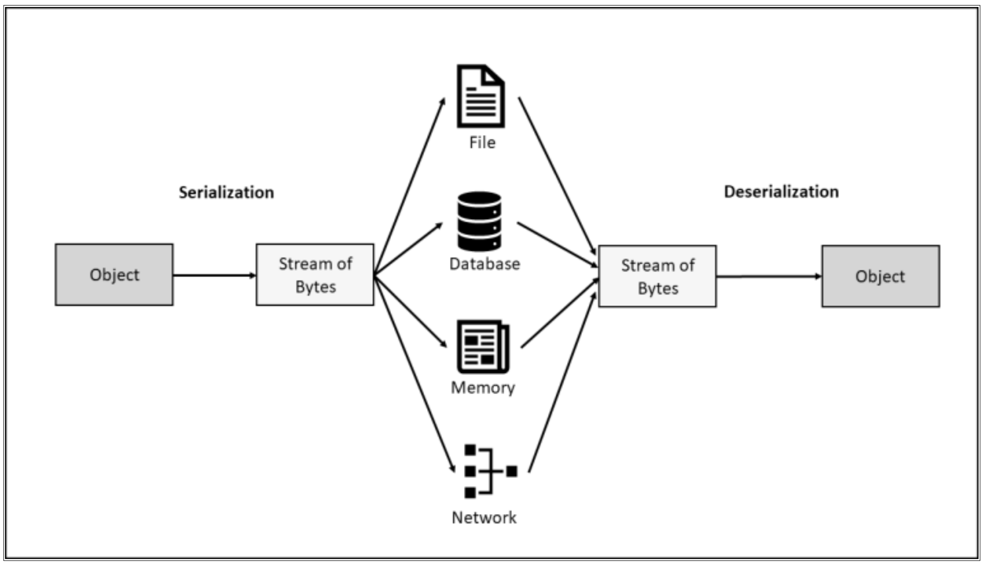

Figure 1: Serialization and De-serialization process

Serialization tools that allow this kind of data exchange are called portability. Portable serialization is not easy for the following reasons: Different processor architectures (little endian and big endian) can be difficult to port because they have different binary representations of numbers and other objects.

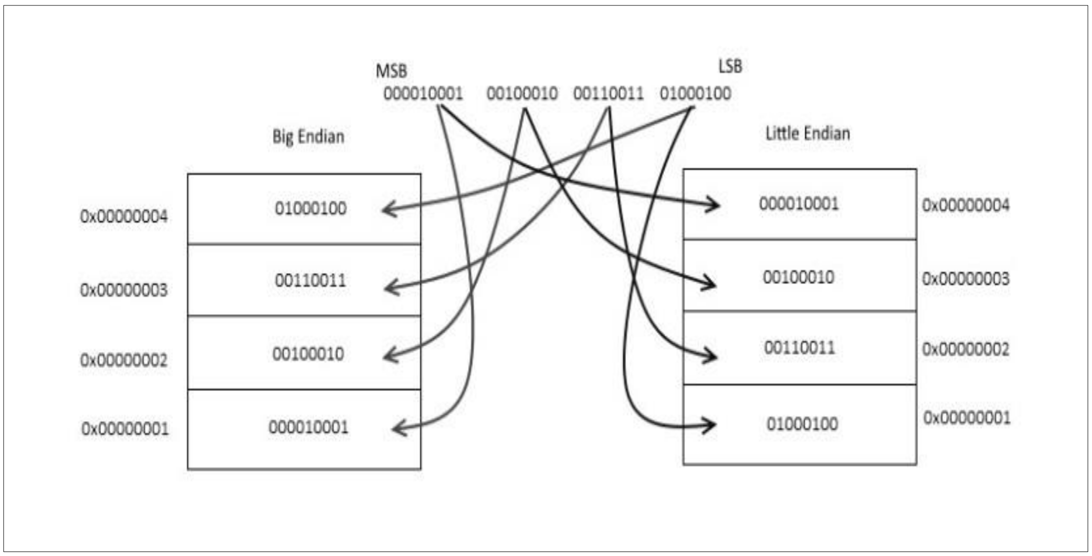

Figure 2: Understanding the Endianness

Endianness is the byte order or sequence of words of digital data in computer memory. The order of degrees is often expressed as big-endian (BE) or little-endian (LE). In a big-endian system, the high-order byte of a word is stored at the lowest memory address and the low-order byte is stored at the highest. In contrast, in little-endian systems, the least significant byte is stored at the least significant address. Binary endianness is a supported feature on many computer architectures that have switchable endianness when fetching and storing data or fetching instructions. Other spells are commonly referred to as middle-endian or mixed-endian.

BSF is a library for serializing C data, the data is stored in natural binary format. The API is small and tries to stay "in the way". BSF can serialize many C data types, including structures. BSF makes a convenient file format.

For example, suppose a program needs to store a list of user names and identifiers. This can be expressed using the format string A(si). If your program requires two such lists (example- one for regular users and one for administrators), it can be expressed as A(si)A(si). This structured data is easy to read and write with BSF.

Expressing type: BSF "datatype" is explicitly specified as a format string. There is no ambiguity about data types stored in BSF. Some examples are:

- A(is) - variable-length array of integer-string pairs
- A(is)A(is) - two such arrays that are, completely independent of each other
- S(ci) - structure containing a char and integer
- S(ci)# - fixed-length array
- A(A(i))- nested array, an array of integer arrays

BSF Image: A BSF image is a serialized form of BSF that is stored in a memory buffer, file, or written to a file descriptor. An image is a strictly defined binary buffer consisting of two sections: a header and data. The header encodes the image's length, format string, endianness, and other flags. The data section contains compressed data. BSF images created on one type of CPU are generally portable to another type of CPU when BSF is used correctly. Given that BSF is a binary format, this might come as a surprise. But BSF has been carefully designed to work. Each format character has a type associated with an explicit size. For integer and floating point types where "endianness" or endianness varies from processor to processor, BSF automatically and transparently adjusts endianness (if necessary) during decompression.

API Concepts: To work with BSF, you need to know how to call API functions and the basic concepts of format strings, arrays, and indexes. Creating a BSF is always the first step, and releasing it is always the last step. In the meantime, either compress and decompress the BSF (if you are serializing the data) or download and decompress the BSF image (if you are deserializing the data).

Table 1. Order of Usage

No.	When Serializing	When De-Serializing
1.	BSF_map()	BSF_map()
2.	BSF_pack()	BSF_load()
3.	BSF_dump()	BSF_unpack()
4.	BSF_free()	BSF_free()

BSF_map: The only way to create a BSF is to call BSF_map(). The first argument is a format string. Here is a list of arguments required for specific characters in the format string. This function creates a mapping between the elements of a format string and a C program variable given an address. Later, the C variable is read/written when the BSF is packed or unpacked. This function returns BSF_node* on success and NULL on failure.

BSF_pack: The pack() method packs data into BSF.

BSF_dump: After packing the BSF, use BSF_dump() to write the BSF image to a file, memory buffer, or file descriptor. The mode is as follows. The last mode is to query the output size without actually generating a dump.

BSF_load: This API reads a previously generated BSF image from a file, memory buffer, or file descriptor and prepares it for subsequent decompression. The format string specified in the previous call to BSF_map() is checked for equality with the format string stored in the BSF image.

BSF_unpack: The BSF_unpack() function unpacks data from the BSF. When the data is unpacked, it is copied to the C program variable originally specified in BSF_map().

BSF_free: The last step for any BSF is to free it using BSF_free(). Its only argument is the BSF_node* to be freed.

BSF_jot: This is a quick way to create a BSF. It can be used instead of the usual "card, pack, dump and free" life cycle. With BSF_jot, all these steps are done. This only works for simple types, i.e. types without an A (...) in the format string.

BSF has no dependencies on any library other than the C system library. Just copy the BSF source into your project so there are no dependencies. Alternatively, you can build BSF as a library and link it into your program.

IV. RESULT AND ANALYSIS

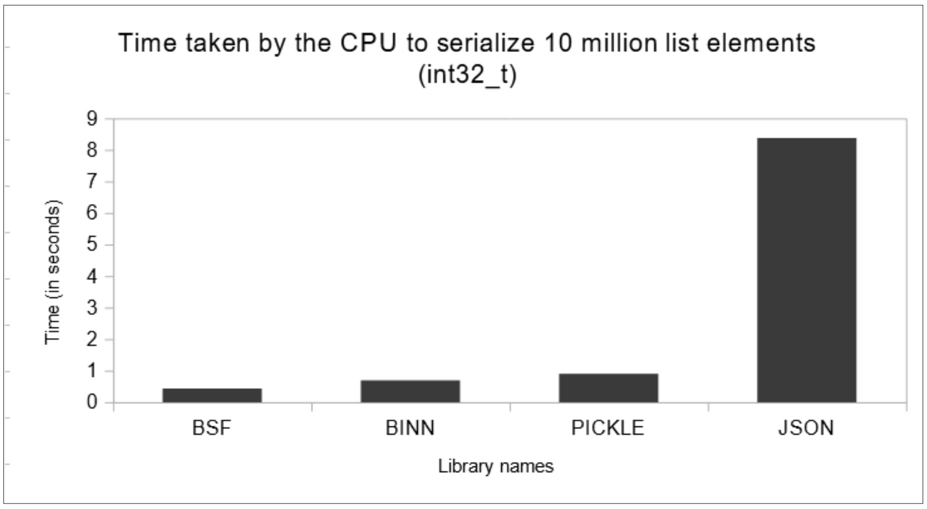

Figure 3: Comparison between Serialization time

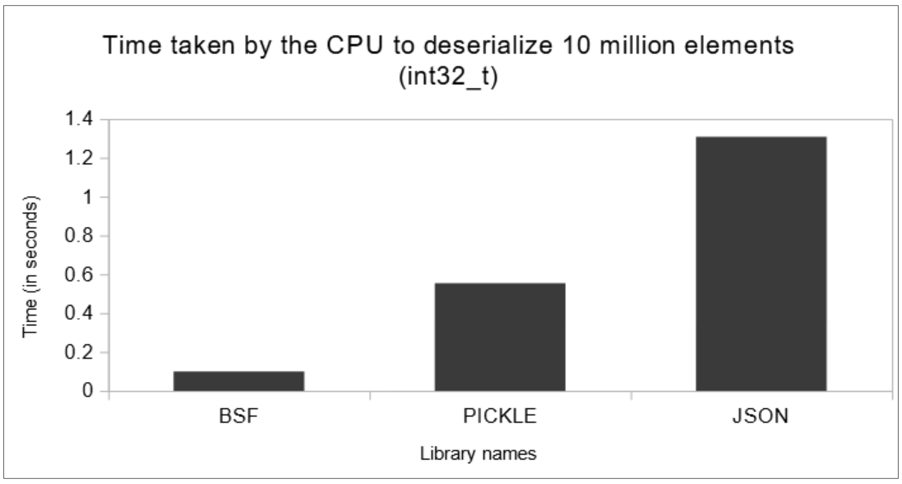

Figure 4: Comparison between Serialization size

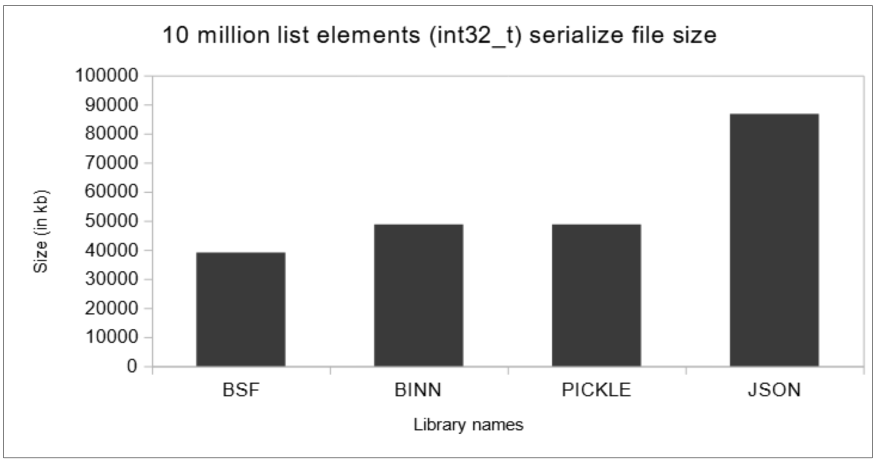

Figure 5: Comparison between De-serialization time

V. CONCLUSION

The main goal of developing this library is to provide simple and efficient data exchange and ease of creation and loading. The minimization method also helps to significantly reduce the serialization size. Serialization using binary serialization methods has proven to be an efficient mechanism for consolidating objects into binary streams for transport to other systems/networks. Binary serialization is much more efficient than other forms of serialization because it saves memory and bandwidth as well as system processing time.

The most compact serialization format is binary format, but these archives are difficult to use to exchange information between modules written in other programming languages. Text formats, such as JSON or XML, are portable and self-interactive, but serialization and serialization processes require additional data processing, and archives can increase much more space than binaries and slower if necessary.

The serialization library described in this article is simply a useful, simple and effective example of the data serialization library. With this library, development, and data transfer process, it becomes much more convenient and efficient. BSF is not dependent on libraries except system libraries. Users can simply copy the BSF source code to their project.

We are currently planning to do an in-depth evaluation to improve the performance of our library. Another area of future work is to change the semantics of serialization algorithms to improve performance. Because serialization is a critical process in applications exchanging messages, identify and study any external processes or conditions in hardware or software that can affect performance.

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude towards our guide Dr. Archana Mire, Prof. Pramila Mate and Prof. Randeep Kahlon for their help, guidance and encouragement, they provided during the project development. This work would have not been possible without their valuable time, patience and motivation. We thank them for making our stint thoroughly pleasant and enriching. It was great learning and an honor being their student. We are deeply thankful to Dr. Archana Mire (HOD of Computer Department) and entire team in the Computer Department. They supported us with scientific guidance, advice and encouragement, they were always helpful and enthusiastic and this inspired us in our work. We take the privilege to express our sincere thanks to Dr. L. K. Ragha our Principal for providing the encouragement and much support throughout our work.

VI. REFERENCES

[1] N. Bhatti and V. Hassan, "Serialization and Deserialization of Objects Using XML," Tata McGrawhill, Advances in Data Management, Volume 1, 2000.

[2] Pete Obermeyer and Jonathan Hawkins, "Serialization of Objects in the NET Framework," Microsoft Corporation, 2001.

[3] Guido Tack and Leif Kornstadt, "General Pickling and Minimization", Science Direct, Electronic Note in Theoretical Computer Science, Institute of Programming Systems Saarland University, Germany, 2006.

[4] Huw Evans, "Why Object Serialization is Inappropriate for Providing Persistence in Java", University of Glasgow Computer Science, Glasgow, G12 8RZ, UK.

[5] Kazuaki Maeda, "Comparative Survey of Object Serialization Techniques and the Programming Supports".