Xtensible Binary Format (XBF): An Efficient Self-Describing Binary Format

David Krauthamer

August 19, 2023

Table of Contents

Web Data Interchange Formats

1 Introduction

- Most common: JSON and XML
- Human-Readable and Self-Describing
- Alternatives: CSV, MessagePack, CBOR

Shortcomings of Current Formats

- Inefficiency of plain text.
- Repeated sending of metadata.
- Use of big-endian.

Plain Text Example

1 Introduction

Table: Sending A 32-bit Unsigned Integer

Number	ASCII Bytes	Binary Bytes
1	1	4
9999	4	4
7654321	7	4

Repeated Metadata

1 Introduction

```
<person>
  <name>John Jackson</name>
  <age>25</age>
</person>
```

Table: XML Space Utilization

	Bytes Used	Percent
Metadata	45	76%
Data	14	24%
Total	59	100

Big Endian Example: 32-bit Unsigned Integer

Number To Send: 1027

Sender (Little Endian)			
3	4	0	0

On the Wire (Big Endian)			
0	0	4	3

Recipient (Little Endian)			
3	4	0	0

Improvements For A New Format

- When possible, send metadata only a single time (mainly for homogenous lists).
- Allow sending metadata to be optional, such as if a client already has it.
- Utilize little-endian to eliminate endian conversions on common architectures.
- Prioritize a simple type system.

Table of Contents

Primitives and Primitive Metadata

- Boolean
- U8, U16, U32, U64, U128, U256
  - Equivalent to uint*_t types in C
- I8, I16, I32, I64, I128, I256
  - Equivalent to int*_t types in C
- F32, F64
  - IEEE 754 floating point numbers
  - Equivalent to float and double in C
- Bytes (Non-UTF-8 Bytes)
- String (UTF-8 Bytes)

Type	Value
Boolean	0
U8	1
U16	2
U32	3
U64	4
U128	5
U256	6

Type	Value
I8	7
I16	8
I32	9
I64	10
I128	11
I256	12
F32	13
F64	14
Bytes	15
String	16

Primitive Example

2 Format Design

- 16-Bit Unsigned Integer: 1024

Disc	1024	
2	0	4

- String: ”hello”

Disc	length							h	e	l	l	o
16	2	0	0	0	0	0	0	104	101	108	108	111

Vectors (Homogenous Lists)

2 Format Design

Values

- Homogenous list of values that has a known length.
- Length is sent as a part of the type, not in metadata.
- Inner type Information is *not* sent as a part of the value.

Metadata

- Single byte discriminant value is sent (String + 1).
- Followed by internal type metadata (can continue recursively).
- Length is *not* sent as a part of the metadata.

Vector Example

2 Format Design

Vector of 16-Bit Unsigned Integers: [42, 1024]

DV = Discriminant of a Vector
D16 = Discriminant of a 16-Bit Unsigned Integer

DV	D16	length							42		1024	
17	2	2	0	0	0	0	0	0	42	0	0	4

Structs (Aggregate Types)

- An aggregate type containing a name as well as named fields.
- May not contain duplicate field names.
- Fields are sent in sequence in the order they are listed in metadata.
- Serialized value does *not* include:
  - Struct name.
  - Number of fields.
  - Field names.
  - Field types.

Struct Metadata

- Single byte discriminant value is sent (Vector + 1).
- Followed by Struct name.
- Next, number of fields as an unsigned 16-bit integer.
- Last, pairs of field names and types.
  - May continue recursively with nested types of Structs or Vectors.
- All names are sent with the same format as primitive Strings.

Struct Metadata Example

2 Format Design

DS	struct name length								P	o	i	n	t	fields	
18	5	0	0	0	0	0	0	0	80	111	105	110	116	2	0

first name length								f	i	r	s	t	DF
5	0	0	0	0	0	0	0	102	105	114	115	116	7

last name length								l	a	s	t	DL
4	0	0	0	0	0	0	0	108	97	115	116	7

```
struct Point {
  first: i8,
  last: i8
}
```

DS = Struct Discriminant
DF = Field ”first” discriminant
DL = Field ”last” discriminant

Struct Value Example

2 Format Design

first	last
42	251

```
let my_point = Point {
  first: 42,
  last: -5,
};
```

Table of Contents

Encoder and Decoder Reference Implementation

3 Evaluation

- Written in Rust
  - Efficient code generation similar to C and C++
  - Built-in tagged unions
  - Strong memory safety guarantees
- Extensively tested
  - Unit and integration tests
  - 100 percent code coverage

Performance Test

3 Evaluation

Multi-threaded Server

- Downloads and parses 1 year of Sony stock history.
- Records the original CSV file size in bytes.
- Records the size in bytes when stored natively in memory.
- Waits to receive requests.

Client

- Formats: CSV, MessagePack, CBOR, JSON, XML, XBF
- For each data format:
  - Client sends 100 requests to the server.
  - Records average time to receive a response.
  - Records number of bytes read.

Table of Contents

Data

4 Results and Conclusions

Format	Avg Time (ms)	Bytes Read	Overhead (bytes)	Percent Overhead
CSV	18.93	16,411	1,823	11.1
MessagePack	11.22	15,565	977	6.22
CBOR	16.95	25,507	10,919	42.8
JSON	21.91	31,180	16,592	53.2
XML	21.87	43,699	29,111	66.6
XBF	11.32	14,686	98	0.67

Original CSV data size recorded by the server: 17,160 bytes.
Native data size recorded by the server: 14,558 bytes.

CSV Size Difference

4 Results and Conclusions

CSV Size Difference

- Original data always used 8 digits of precision.
- CSV library used only writes the required digits of precision.
- Example: 83.500000 becomes 83.5

Time Performance

- XBF was faster than every format except MessagePack.
- Formats that operated in binary were faster, even when sending more data (CBOR).
- XBF reference implementation was optimized for memory usage, more could be done for speed.

Size Performance

4 Results and Conclusions

- XBF required the least amount of bytes.
- Indicates metadata deduplication worked as intended.
- XBF can be even smaller with optimization techniques from MessagePack
  - Coercion to smaller integer sizes when possible.
  - Differently sized variable length types.
- Could be possible to have a ”negative” overhead.

Use Cases

4 Results and Conclusions

- Internet of Things and other low bandwidth scenarios.
- High volume applications.

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Thank you for listening!
Any questions?