BIPF, a Serialization Format for JSON-like Data Types [2023-07-14]

BIPF (Binary In-Place Format) is a serialization format for JSON-like data types that is optimized for in-place access. It is used in some versions of Secure Scuttlebut, including tinySSB.

This document focuses on the bit-level format and also includes test vectors. The authoritative places for BIPF are `https://github.com/ssbc/bipf-spec` as well as the repo for a Javascript implementation `https://github.com/ssbc/bipf` that includes the rationale for BIPF. JSON is documented in `https://www.rfc-editor.org/rfc/rfc8259`.

Supported Data Types

A simplified version of the JSON schema is captured by the following EBNF grammar:

```
JSON_VAL   ::= JSON_ATOM | JSON_LIST | JSON_DICT
JSON_ATOM  ::= 'true' | 'false' | 'null' | INT_VAL | DOUBLE_VAL | STR_VAL
JSON_LIST  ::= '[' JSON_VAL (',' JSON_VAL)* ']'
JSON_DICT   ::= '{' STR_VAL ':' JSON_VAL (',' STR_VAL ':' JSON_VAL)* '}'
```

BIPF is slightly more expressive in that it supports byte arrays, not only strings. Moreover, any atom can be used as a key for a dictionary, not only strings:

```
BIPF_VAL   ::= BIPF_ATOM | BIPF_LIST | BIPF_DICT
BIPF_ATOM  ::= 'true' | 'false' | 'null' | INT_VAL | DOUBLE_VAL | STR_VAL | BYTES_VAL
BIPF_LIST  ::= '[' BIPF_VAL (',' BIPF_VAL)* ']'
BIPF_DICT  ::= '{' BIPF_ATOM ':' BIPF_VAL (',' BIPF_ATOM ':' BIPF_VAL)* '}'
```

Serialization

For each of the 7 atomic and 2 structured types, a bit pattern is assigned. These type identifiers are later used in the encoding of a corresponding value:

```
STRING  : 0 (000) // utf8 encoded string
BYTES   : 1 (001) // raw byte sequence
INT     : 2 (010) // signed LEB128-encoded integer
DOUBLE  : 3 (011) // IEEE 754-encoded double precision floating point
LIST    : 4 (100) // sequence of bipf-encoded values
DICT    : 5 (101) // sequence of alternating bipf-encoded key and value
BOOLNULL: 6 (110) // 1 = true, 0 = false, no value means null
EXTENDED: 7 (111) // custom type. Specific type should be indicated by varint at start of buffer
```

Note that the `BOOLNULL` bit pattern is used for three different atom types.

BIPF values are serialized with a TYPE-LENGTH-VALUE (TLV) encoding. To this end, T and L are combined into a single integer value called *tag* which is encoded with signed LEB128, see `https://en.wikipedia.org/wiki/LEB128`, also known as `varint`. The encoded *tag* is then prepended to the bytes of the encoding of the value V proper.

```
enc(V)   ::= concat( tag(V.type,V.length), V.bytes )
tag(t,l) ::= LEB128( l<<3 + t )
```

LEB128 encoding is done by producing one encoded byte for each 7 input bit group, starting with the least significant group of 7 bits. Signed values are assumed to be represented in two's complement. All encoded bytes will have the most significant bit set, except the last byte. Zero is encoded as byte 0x00.

Examples

operation	memory content of result
bipf(null)	06
bipf(false)	0e00
bipf(true)	0e01
bipf(123)	0a7b
bipf(-123)	0a85
bipf("¥€$!")	39c2a5e282ac2421
bipf(#ABCD#)	11abcd
bipf([123,true])	240a7b0e01
bipf({123:false})	250a7b0e00
bipf({#ABCD#:[123,null]})	3d11abcd1c0a7b06

Notes

BIPF as described in `https://github.com/ssbc/bipf-spec` uses a fixed-length encoding for integer values (4 bytes, little endian, two's complement). With space concerns in mind, tinySSB uses the LEB128 (varint) representation, which is already used for encoding the length of a BIPF value, hence has no additional programming cost.

This document located at `https://PREFIX/tinySSB-2023/DOC/bipf.pdf`