msgpack/msgpack

MessagePack specification

MessagePack is an object serialization specification like JSON.

MessagePack has two concepts: **type system** and **formats**.

Serialization is conversion from application objects into MessagePack formats via MessagePack type system.

Deserialization is conversion from MessagePack formats into application objects via MessagePack type system.

```
Serialization:
    Application objects
    -->  MessagePack type system
    -->  MessagePack formats (byte array)

Deserialization:
    MessagePack formats (byte array)
    -->  MessagePack type system
    -->  Application objects
```

This document describes the MessagePack type system, MessagePack formats and conversion of them.

Table of contents

Type system

- Types
    - **Integer** represents an integer
    - **Nil** represents nil
    - **Boolean** represents true or false
    - **Float** represents a IEEE 754 double precision floating point number including NaN and Infinity
    - **Raw**
        - **String** extending Raw type represents a UTF-8 string
        - **Binary** extending Raw type represents a byte array
    - **Array** represents a sequence of objects
    - **Map** represents key-value pairs of objects
    - **Extension** represents a tuple of type information and a byte array where type information is an integer whose meaning is

defined by applications or MessagePack specification
    - **Timestamp** represents an instantaneous point on the time-line in the world that is independent from time zones or calendars. Maximum precision is nanoseconds.

Limitation

- a value of an Integer object is limited from `-(2^63)` upto `(2^64)-1`
- maximum length of a Binary object is `(2^32)-1`
- maximum byte size of a String object is `(2^32)-1`
- String objects may contain invalid byte sequence and the behavior of a deserializer depends on the actual implementation when it received invalid byte sequence
    - Deserializers should provide functionality to get the original byte array so that applications can decide how to handle the object
- maximum number of elements of an Array object is `(2^32)-1`
- maximum number of key-value associations of a Map object is `(2^32)-1`

Extension types

MessagePack allows applications to define application-specific types using the Extension type. Extension type consists of an integer and a byte array where the integer represents a kind of types and the byte array represents data.

Applications can assign `0` to `127` to store application-specific type information. An example usage is that application defines `type = 0` as the application's unique type system, and stores name of a type and values of the type at the payload.

MessagePack reserves `-1` to `-128` for future extension to add predefined types. These types will be added to exchange more types without using pre-

shared statically-typed schema across different programming environments.

```
[0, 127]: application-specific types
[-128, -1]: reserved for predefined types
```

Because extension types are intended to be added, old applications may not implement all of them. However, they can still handle such type as one of Extension types. Therefore, applications can decide whether they reject unknown Extension types, accept as opaque data, or transfer to another application without touching payload of them.

Here is the list of predefined extension types. Formats of the types are defined at Formats section.

Name	Type
Timestamp	-1

Formats

Overview

format name	first byte (in binary)	first byte (in hex)
positive fixint	0xxxxxxx	0x00 - 0x7f
fixmap	1000xxxx	0x80 - 0x8f
fixarray	1001xxxx	0x90 - 0x9f
fixstr	101xxxxx	0xa0 - 0xbf
nil	11000000	0xc0
(never used)	11000001	0xc1
false	11000010	0xc2
true	11000011	0xc3
bin 8	11000100	0xc4
bin 16	11000101	0xc5

bin 32	11000110	0xc6
ext 8	11000111	0xc7
ext 16	11001000	0xc8
ext 32	11001001	0xc9
float 32	11001010	0xca
float 64	11001011	0xcb
uint 8	11001100	0xcc
uint 16	11001101	0xcd
uint 32	11001110	0xce
uint 64	11001111	0xcf
int 8	11010000	0xd0
int 16	11010001	0xd1
int 32	11010010	0xd2
int 64	11010011	0xd3
fixext 1	11010100	0xd4
fixext 2	11010101	0xd5
fixext 4	11010110	0xd6
fixext 8	11010111	0xd7
fixext 16	11011000	0xd8
str 8	11011001	0xd9
str 16	11011010	0xda
str 32	11011011	0xdb
array 16	11011100	0xdc
array 32	11011101	0xdd
map 16	11011110	0xde
map 32	11011111	0xdf
negative fixint	111xxxxx	0xe0 - 0xff

Notation in diagrams

one byte:

```
+--------+
|        |
+--------+

a variable number of bytes:
+========+
|        |
+========+

variable number of objects stored in MessagePack format:
+~~~~~~~~~~~~~~~~~+
|                 |
+~~~~~~~~~~~~~~~~~+
```

`X`, `Y`, `Z` and `A` are the symbols that will be replaced by an actual bit.

nil format

Nil format stores nil in 1 byte.

```
nil:
+--------+
|  0xc0  |
+--------+
```

bool format family

Bool format family stores false or true in 1 byte.

```
false:
+--------+
|  0xc2  |
+--------+

true:
+--------+
|  0xc3  |
+--------+
```

int format family

Int format family stores an integer in 1, 2, 3, 5, or 9 bytes.

```
positive fixnum stores 7-bit positive integer
+--------+
|0XXXXXXX|
+--------+

negative fixnum stores 5-bit negative integer
+--------+
|111YYYYY|
+--------+

* 0XXXXXXX is 8-bit unsigned integer
* 111YYYYY is 8-bit signed integer

uint 8 stores a 8-bit unsigned integer
+--------+--------+
|  0xcc  |ZZZZZZZZ|
+--------+--------+

uint 16 stores a 16-bit big-endian unsigned integer
+--------+--------+--------+
|  0xcd  |ZZZZZZZZ|ZZZZZZZZ|
+--------+--------+--------+

uint 32 stores a 32-bit big-endian unsigned integer
+--------+--------+--------+--------+--------+
|  0xce  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|
+--------+--------+--------+--------+--------+

uint 64 stores a 64-bit big-endian unsigned integer
+--------+--------+--------+--------+--------+--------+--------+--------+--
|  0xcf  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZ
+--------+--------+--------+--------+--------+--------+--------+--------+--

int 8 stores a 8-bit signed integer
+--------+--------+
|  0xd0  |ZZZZZZZZ|
+--------+--------+

int 16 stores a 16-bit big-endian signed integer
+--------+--------+--------+
```

```
|  0xd1  |ZZZZZZZZ|ZZZZZZZZ|
+--------+--------+--------+

int 32 stores a 32-bit big-endian signed integer
+--------+--------+--------+--------+--------+
|  0xd2  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|
+--------+--------+--------+--------+--------+

int 64 stores a 64-bit big-endian signed integer
+--------+--------+--------+--------+--------+--------+--------+--------+--
|  0xd3  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZ
+--------+--------+--------+--------+--------+--------+--------+--------+--
```

float format family

Float format family stores a floating point number in 5 bytes or 9 bytes.

```
float 32 stores a floating point number in IEEE 754 single precision floati
+--------+--------+--------+--------+--------+
|  0xca  |XXXXXXXX|XXXXXXXX|XXXXXXXX|XXXXXXXX|
+--------+--------+--------+--------+--------+

float 64 stores a floating point number in IEEE 754 double precision floati
+--------+--------+--------+--------+--------+--------+--------+--------+--
|  0xcb  |YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|YYYYYYYY|YY
+--------+--------+--------+--------+--------+--------+--------+--------+--

where
* XXXXXXXX_XXXXXXXX_XXXXXXXX_XXXXXXXX is a big-endian IEEE 754 single preci
  Extension of precision from single-precision to double-precision does not
* YYYYYYYY_YYYYYYYY_YYYYYYYY_YYYYYYYY_YYYYYYYY_YYYYYYYY_YYYYYYYY_YYYYYYYY i
  IEEE 754 double precision floating point number
```

str format family

Str format family stores a byte array in 1, 2, 3, or 5 bytes of extra bytes in addition to the size of the byte array.

```
fixstr stores a byte array whose length is upto 31 bytes:
+--------+========+
```

```
|101XXXXX|  data  |
+--------+========+

str 8 stores a byte array whose length is upto (2^8)-1 bytes:
+--------+--------+========+
|  0xd9  |YYYYYYYY|  data  |
+--------+--------+========+

str 16 stores a byte array whose length is upto (2^16)-1 bytes:
+--------+--------+--------+========+
|  0xda  |ZZZZZZZZ|ZZZZZZZZ|  data  |
+--------+--------+--------+========+

str 32 stores a byte array whose length is upto (2^32)-1 bytes:
+--------+--------+--------+--------+--------+========+
|  0xdb  |AAAAAAAA|AAAAAAAA|AAAAAAAA|AAAAAAAA|  data  |
+--------+--------+--------+--------+--------+========+

where
* XXXXX is a 5-bit unsigned integer which represents N
* YYYYYYYY is a 8-bit unsigned integer which represents N
* ZZZZZZZZ_ZZZZZZZZ is a 16-bit big-endian unsigned integer which represent
* AAAAAAAA_AAAAAAAA_AAAAAAAA_AAAAAAAA is a 32-bit big-endian unsigned integ
* N is the length of data
```

bin format family

Bin format family stores an byte array in 2, 3, or 5 bytes of extra bytes in addition to the size of the byte array.

```
bin 8 stores a byte array whose length is upto (2^8)-1 bytes:
+--------+--------+========+
|  0xc4  |XXXXXXXX|  data  |
+--------+--------+========+

bin 16 stores a byte array whose length is upto (2^16)-1 bytes:
+--------+--------+--------+========+
|  0xc5  |YYYYYYYY|YYYYYYYY|  data  |
+--------+--------+--------+========+

bin 32 stores a byte array whose length is upto (2^32)-1 bytes:
+--------+--------+--------+--------+--------+========+
```

```
|  0xc6  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|  data  |
+--------+--------+--------+--------+--------+========+

where
* XXXXXXXX is a 8-bit unsigned integer which represents N
* YYYYYYYY_YYYYYYYY is a 16-bit big-endian unsigned integer which represent
* ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ is a 32-bit big-endian unsigned integ
* N is the length of data
```

array format family

Array format family stores a sequence of elements in 1, 3, or 5 bytes of extra bytes in addition to the elements.

```
fixarray stores an array whose length is upto 15 elements:
+--------+~~~~~~~~~~~~~~~~~+
|1001XXXX|    N objects    |
+--------+~~~~~~~~~~~~~~~~~+

array 16 stores an array whose length is upto (2^16)-1 elements:
+--------+--------+--------+~~~~~~~~~~~~~~~~~+
|  0xdc  |YYYYYYYY|YYYYYYYY|    N objects    |
+--------+--------+--------+~~~~~~~~~~~~~~~~~+

array 32 stores an array whose length is upto (2^32)-1 elements:
+--------+--------+--------+--------+--------+~~~~~~~~~~~~~~~~~+
|  0xdd  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|    N objects    |
+--------+--------+--------+--------+--------+~~~~~~~~~~~~~~~~~+

where
* XXXX is a 4-bit unsigned integer which represents N
* YYYYYYYY_YYYYYYYY is a 16-bit big-endian unsigned integer which represent
* ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ is a 32-bit big-endian unsigned integ
    N is the size of a array
```

map format family

Map format family stores a sequence of key-value pairs in 1, 3, or 5 bytes of extra bytes in addition to the key-value pairs.

```
fixmap stores a map whose length is upto 15 elements
+--------+~~~~~~~~~~~~~~~~~+
|1000XXXX|   N*2 objects   |
+--------+~~~~~~~~~~~~~~~~~+

map 16 stores a map whose length is upto (2^16)-1 elements
+--------+--------+--------+~~~~~~~~~~~~~~~~~+
|  0xde  |YYYYYYYY|YYYYYYYY|   N*2 objects   |
+--------+--------+--------+~~~~~~~~~~~~~~~~~+

map 32 stores a map whose length is upto (2^32)-1 elements
+--------+--------+--------+--------+--------+~~~~~~~~~~~~~~~~~+
|  0xdf  |ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|ZZZZZZZZ|   N*2 objects   |
+--------+--------+--------+--------+--------+~~~~~~~~~~~~~~~~~+

where
* XXXX is a 4-bit unsigned integer which represents N
* YYYYYYYY_YYYYYYYY is a 16-bit big-endian unsigned integer which represent
* ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ is a 32-bit big-endian unsigned integ
* N is the size of a map
* odd elements in objects are keys of a map
* the next element of a key is its associated value
```

ext format family

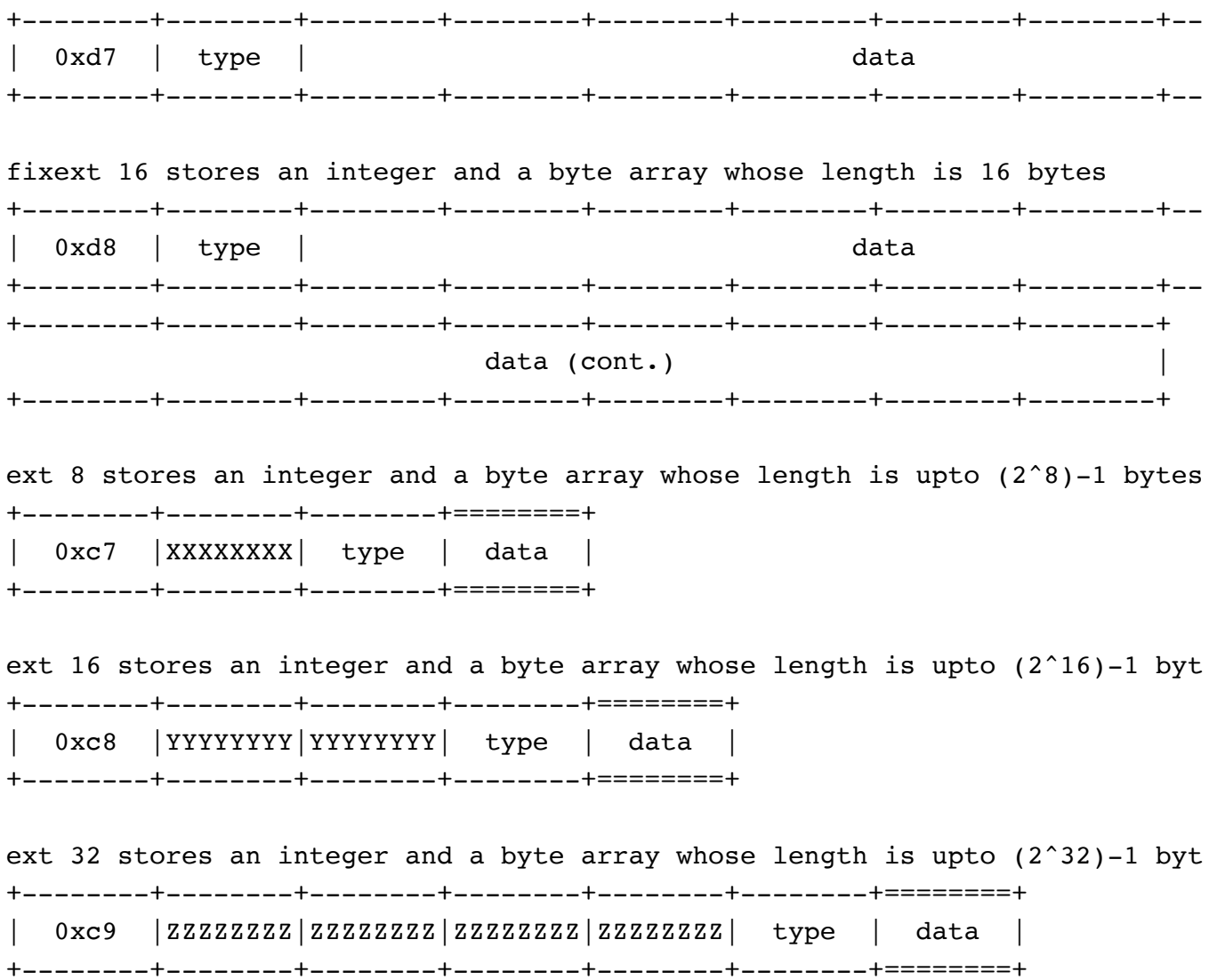
Ext format family stores a tuple of an integer and a byte array.

```
fixext 1 stores an integer and a byte array whose length is 1 byte
+--------+--------+--------+
|  0xd4  |  type  |  data  |
+--------+--------+--------+

fixext 2 stores an integer and a byte array whose length is 2 bytes
+--------+--------+--------+--------+
|  0xd5  |  type  |       data      |
+--------+--------+--------+--------+

fixext 4 stores an integer and a byte array whose length is 4 bytes
+--------+--------+--------+--------+--------+--------+
|  0xd6  |  type  |                data               |
+--------+--------+--------+--------+--------+--------+

fixext 8 stores an integer and a byte array whose length is 8 bytes
```



where
* XXXXXXXX is a 8-bit unsigned integer which represents N
* YYYYYYYY_YYYYYYYY is a 16-bit big-endian unsigned integer which represent
* ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ_ZZZZZZZZ is a big-endian 32-bit unsigned integ
* N is a length of data
* type is a signed 8-bit signed integer
* type < 0 is reserved for future extension including 2-byte type informati

Timestamp extension type

Timestamp extension type is assigned to extension type `-1`. It defines 3 formats: 32-bit format, 64-bit format, and 96-bit format.

timestamp 32 stores the number of seconds that have elapsed since 1970-01-0
in an 32-bit unsigned integer:

```
+--------+--------+--------+--------+--------+--------+
|  0xd6  |   -1   |   seconds in 32-bit unsigned int  |
+--------+--------+--------+--------+--------+--------+

timestamp 64 stores the number of seconds and nanoseconds that have elapsed
in 32-bit unsigned integers:
+--------+--------+--------+--------+--------+--------+--------+--------+--
|  0xd7  |   -1   |nanoseconds in 30-bit unsigned int |  seconds in 34-bit
+--------+--------+--------+--------+--------+--------+--------+--------+--

timestamp 96 stores the number of seconds and nanoseconds that have elapsed
in 64-bit signed integer and 32-bit unsigned integer:
+--------+--------+--------+--------+--------+--------+--------+
|  0xc7  |   12   |   -1   |nanoseconds in 32-bit unsigned int |
+--------+--------+--------+--------+--------+--------+--------+
+--------+--------+--------+--------+--------+--------+--------+--------+
                    seconds in 64-bit signed int                        |
+--------+--------+--------+--------+--------+--------+--------+--------+
```

- Timestamp 32 format can represent a timestamp in [1970-01-01 00:00:00 UTC, 2106-02-07 06:28:16 UTC) range. Nanoseconds part is 0.
- Timestamp 64 format can represent a timestamp in [1970-01-01 00:00:00.000000000 UTC, 2514-05-30 01:53:04.000000000 UTC) range.
- Timestamp 96 format can represent a timestamp in [-584554047284-02-23 16:59:44 UTC, 584554051223-11-09 07:00:16.000000000 UTC) range.
- In timestamp 64 and timestamp 96 formats, nanoseconds must not be larger than 999999999.

Pseudo code for serialization:

```
struct timespec {
    long tv_sec;  // seconds
    long tv_nsec; // nanoseconds
} time;
if ((time.tv_sec >> 34) == 0) {
    uint64_t data64 = (time.tv_nsec << 34) | time.tv_sec;
```

```
    if (data64 & 0xffffffff00000000L == 0) {
        // timestamp 32
        uint32_t data32 = data64;
        serialize(0xd6, -1, data32)
    }
    else {
        // timestamp 64
        serialize(0xd7, -1, data64)
    }
}
else {
    // timestamp 96
    serialize(0xc7, 12, -1, time.tv_nsec, time.tv_sec)
}
```

Pseudo code for deserialization:

```
ExtensionValue value = deserialize_ext_type();
struct timespec result;
switch(value.length) {
case 4:
    uint32_t data32 = value.payload;
    result.tv_nsec = 0;
    result.tv_sec = data32;
case 8:
    uint64_t data64 = value.payload;
    result.tv_nsec = data64 >> 34;
    result.tv_sec = data64 & 0x00000003ffffffffL;
case 12:
    uint32_t data32 = value.payload;
    uint64_t data64 = value.payload + 4;
    result.tv_nsec = data32;
    result.tv_sec = data64;
default:
    // error
}
```

Serialization: type to format conversion

MessagePack serializers convert MessagePack types into formats as following:

source types	output format
Integer	int format family (positive fixint, negative fixint, int 8/16/32/64 or uint 8/16/32/64)
Nil	nil
Boolean	bool format family (false or true)
Float	float format family (float 32/64)
String	str format family (fixstr or str 8/16/32)
Binary	bin format family (bin 8/16/32)
Array	array format family (fixarray or array 16/32)
Map	map format family (fixmap or map 16/32)
Extension	ext format family (fixext or ext 8/16/32)

If an object can be represented in multiple possible output formats, serializers SHOULD use the format which represents the data in the smallest number of bytes.

Deserialization: format to type conversion

MessagePack deserializers convert MessagePack formats into types as following:

source formats	output type
positive fixint, negative fixint, int 8/16/32/64 and uint 8/16/32/64	Integer
nil	Nil
false and true	Boolean
float 32/64	Float
fixstr and str 8/16/32	String
bin 8/16/32	Binary
fixarray and array 16/32	Array
fixmap map 16/32	Map

fixext and ext 8/16/32	Extension

Future discussion

Profile

Profile is an idea that Applications restrict the semantics of MessagePack while sharing the same syntax to adapt MessagePack for certain use cases.

For example, applications may remove Binary type, restrict keys of map objects to be String type, and put some restrictions to make the semantics compatible with JSON. Applications which use schema may remove String and Binary types and deal with byte arrays as Raw type. Applications which use hash (digest) of serialized data may sort keys of maps to make the serialized data deterministic.

Implementation guidelines

Upgrading MessagePack specification

MessagePack specification is changed at this time. Here is a guideline to upgrade existent MessagePack implementations:

- In a minor release, deserializers support the bin format family and str 8 format. The type of deserialized objects should be same with raw 16 (== str 16) or raw 32 (== str 32)
- In a major release, serializers distinguish Binary type and String type using bin format family and str format family
    - At the same time, serializers should offer "compatibility mode" which doesn't use bin format family and str 8 format

```
MessagePack specification
Last modified at 2017-08-09 22:42:07 -0700
Sadayuki Furuhashi © 2013-04-21 21:52:33 -0700
```