# Specification Version 1.0

BSON is a binary format in which zero or more key/value pairs are stored as a single entity. We call this entity a *document*.

The following grammar specifies version 1.0 of the BSON standard. We've written the grammar using a pseudo-BNF syntax. Valid BSON data is represented by the `document` non-terminal.

## Basic Types

The following basic types are used as terminals in the rest of the grammar. Each type must be serialized in little-endian format.

| | |
|---|---|
| `byte` | 1 byte (8-bits) |
| `int32` | 4 bytes (32-bit signed integer, two's complement) |
| `int64` | 8 bytes (64-bit signed integer, two's complement) |
| `double` | 8 bytes (64-bit IEEE 754 floating point) |

## Non-terminals

The following specifies the rest of the BSON grammar. Note that quoted strings represent terminals, and should be interpreted with C semantics (e.g. `"\x01"` represents the byte `0000 0001`). Also note that we use the `*` operator as shorthand for repetition (e.g. `("\x01"*2)` is `"\x01\x01"`). When used as a unary operator, `*` means that the repetition can occur 0 or more times.

| | |
|---|---|
| `document ::= int32 e_list "\x00"` | BSON Document. int32 is the total number of bytes comprising the document. |
| `e_list   ::= element e_list` | |
| `           \| ""` | |
| `element  ::= "\x01" e_name double` | Floating point |
| `           \| "\x02" e_name string` | UTF-8 string |
| `           \| "\x03" e_name document` | Embedded document |
| `           \| "\x04" e_name document` | Array |
| `           \| "\x05" e_name binary` | Binary data |
| `           \| "\x06" e_name` | Undefined — *Deprecated* |
| `           \| "\x07" e_name (byte*12)` | ObjectId |
| `           \| "\x08" e_name "\x00"` | Boolean "false" |
| `           \| "\x08" e_name "\x01"` | Boolean "true" |
| `           \| "\x09" e_name int64` | UTC datetime |
| `           \| "\x0A" e_name` | Null value |
| | Regular expression - The first |

| | | |
|---|---|---|
| | \| "\x0B" e_name cstring cstring | cstring is the regex pattern, the second is the regex options string. Options are identified by characters, which must be stored in alphabetical order. Valid options are 'i' for case insensitive matching, 'm' for multiline matching, 'x' for verbose mode, 'l' to make \w, \W, etc. locale dependent, 's' for dotall mode ('.' matches everything), and 'u' to make \w, \W, etc. match unicode. |
| | \| "\x0C" e_name string (byte*12) | DBPointer — *Deprecated* |
| | \| "\x0D" e_name string | JavaScript code |
| | \| "\x0E" e_name string | Deprecated |
| | \| "\x0F" e_name code_w_s | JavaScript code w/ scope |
| | \| "\x10" e_name int32 | 32-bit Integer |
| | \| "\x11" e_name int64 | Timestamp |
| | \| "\x12" e_name int64 | 64-bit integer |
| | \| "\xFF" e_name | Min key |
| | \| "\x7F" e_name | Max key |
| e_name | ::= cstring | Key name |
| string | ::= int32 (byte*) "\x00" | String - The int32 is the number bytes in the (byte*) + 1 (for the trailing '\x00'). The (byte*) is zero or more UTF-8 encoded characters. |
| cstring | ::= (byte*) "\x00" | Zero or more modified UTF-8 encoded characters followed by '\x00'. The (byte*) MUST NOT contain '\x00', hence it is not full UTF-8. |
| binary | ::= int32 subtype (byte*) | Binary - The int32 is the number of bytes in the (byte*). |
| subtype | ::= "\x00" | Generic binary subtype |
| | \| "\x01" | Function |
| | \| "\x02" | Binary (Old) |
| | \| "\x03" | UUID (Old) |
| | \| "\x04" | UUID |
| | \| "\x05" | MD5 |
| | \| "\x80" | User defined |
| code_w_s | ::= int32 string document | Code w/ scope |

## Notes

- Array - The document for an array is a normal BSON document with integer values for the keys, starting with 0 and continuing sequentially. For example, the array ['red', 'blue'] would be encoded as the document {'0': 'red', '1': 'blue'}. The keys must be in ascending numerical order.
- UTC datetime - The int64 is UTC milliseconds since the Unix epoch.
- Timestamp - Special internal type used by MongoDB replication and sharding. First 4 bytes are an increment, second 4 are a timestamp.
- Min key - Special type which compares lower than all other possible BSON element values.
- Max key - Special type which compares higher than all other possible BSON element values.
- Generic binary subtype - This is the most commonly used binary subtype and should be the 'default' for drivers and tools.
- The BSON "binary" or "BinData" datatype is used to represent arrays of bytes. It is somewhat analogous to the Java notion of a ByteArray. BSON binary values have a *subtype*. This is used to indicate what kind of data is in the byte array. Subtypes from zero to 127 are predefined or reserved. Subtypes from 128-255 are user-defined.
    - \x02 Binary (Old) - This used to be the default subtype, but was deprecated in favor of \x00. Drivers and tools should be sure to handle \x02 appropriately. The structure of the binary data (the byte* array in the binary non-terminal) must be an int32 followed by a (byte*). The int32 is the number of bytes in the repetition.
    - \x03 UUID (Old) - This used to be the UUID subtype, but was deprecated in favor of \x04. Drivers and tools for languages with a native UUID type should handle \x03 appropriately.
    - \x80-0xff "User defined" subtypes. The binary data can be anything.
- Code w/ scope - The int32 is the length in bytes of the entire code_w_s value. The string is JavaScript code. The document is a mapping from identifiers to values, representing the scope in which the string should be evaluated.