BSON 4.0.0 Tutorial

On this page

This tutorial discusses using the core Ruby BSON gem.

Installation

The BSON gem is hosted on Rubygems and can be installed manually or with bundler.

To install the gem manually:

```
gem install bson
```

To install the gem with bundler, include the following in your Gemfile:

```
gem 'bson', '~> 4.0'
```

The BSON gem is compatible with MRI 1.9.3, 2.0.x, 2.1.x, 2.2.x, JRuby 1.7.x, and Rubinius 2.5.x

BSON Serialization

Getting a Ruby object's raw BSON representation is done by calling `to_bson` on the Ruby object, which will return a `ByteBuffer`. For example:

```
"Shall I compare thee to a summer's day".to_bson
1024.to_bson
```

Generating an object from BSON is done via calling `from_bson` on the class you wish to instantiate and passing it a `ByteBuffer` instance.

```
String.from_bson(byte_buffer)
Int32.from_bson(byte_buffer)
```

Byte Buffers

In 4.0, BSON introduced the use of native byte buffers in both MRI and JRuby instead of using `StringIO`. This was done for performance improvements.

API

A `BSON::ByteBuffer` can be instantiated with nothing (for write mode) or with a string of raw bytes (for read mode).

```
buffer = BSON::ByteBuffer.new # a write mode buffer.
buffer = BSON::ByteBuffer.new(string) # a read mode buffer.
```

Writing to the buffer is done via the following API:

```
buffer.put_byte(value) # Appends a single byte.
buffer.put_cstring(value) # Appends a null-terminated string.
buffer.put_double(value) # Appends a 64-bit floating point.
buffer.put_int32(value) # Appends a 32-bit integer (4 bytes).
buffer.put_int64(value) # Appends a 64-bit integer (8 bytes).
buffer.put_string(value) # Appends a UTF-8 string.
```

Reading from the buffer is done via the following API:

```
buffer.get_byte # Pulls a single byte from the buffer.
buffer.get_bytes(value) # Pulls n number of bytes from the buffer.
buffer.get_cstring # Pulls a null-terminted string from the buffer.
buffer.get_double # Pulls a 64-bit floating point from the buffer.
buffer.get_int32 # Pulls a 32-bit integer (4 bytes) from the buffer.
buffer.get_int64 # Pulls a 64-bit integer (8 bytes) from the buffer.
buffer.get_string # Pulls a UTF-8 string from the buffer.
```

Convertig a buffer to it's raw bytes, for example to send over a socket, is done by simply calling to_s on the buffer.

```
buffer = BSON::ByteBuffer.new
buffer.put_string('testing')
socket.write(buffer.to_s)
```

Supported Objects

Core Ruby objects that have representations in the BSON specification and will have a `to_bson` method defined for them are: `Object`, `Array`, `FalseClass`, `Float`, `Hash`, `Integer`, `NilClass`, `Regexp`, `String`, `Symbol` (deprecated), `Time`, `TrueClass`.

In addition to the core Ruby objects, BSON also provides some special types specific to the specification:

`BSON::Binary`

This is a representation of binary data, and must provide the raw data and a subtype when constructing.

```
BSON::Binary.new(binary_data, :md5)
```

Valid subtypes are: `:generic`, `:function`, `:old`, `:uuid_old`, `:uuid`, `:md5`, `:user`.

`BSON::Code`

Represents a string of Javascript code.

```
BSON::Code.new("this.value = 5;")
```

`BSON::CodeWithScope`

Represents a string of Javascript code with a hash of values.

```
BSON::CodeWithScope.new("this.value = age;", age: 5)
```

`BSON::Document`

This is a subclass of a hash that stores all keys as strings but allows access to them with symbol keys.

```
BSON::Document[:key, "value"]
BSON::Document.new
```

`BSON::MaxKey`

Represents a value in BSON that will always compare higher to another value.

```
BSON::MaxKey.new
```

`BSON::MinKey`

Represents a value in BSON that will always compare lower to another value.

```
BSON::MinKey.new
```

`BSON::ObjectId`

Represents a 12 byte unique identifier for an object on a given machine.

```
BSON::ObjectId.new
```

`BSON::Timestamp`

Represents a special time with a start and increment value.

```
BSON::Timestamp.new(5, 30)
```

`BSON::Undefined`

Represents a placeholder for a value that was not provided.

```
BSON::Undefined.new
```

`BSON::Decimal128`

Represents a 128-bit decimal-based floating-point value capable of emulating decimal rounding with exact precision..

```
# Instantiate with a String
BSON::Decimal128.new("1.28")

# Instantiate with a BigDecimal
d = BigDecimal.new(1.28, 3)
BSON::Decimal128.new(d)
```

JSON Serialization

Some BSON types have special representations in JSON. These are as follows and will be automatically serialized in the form when calling `to_json` on them.

Object	JSON
`BSON::Binary`	`{ "$binary" : "\x01", "$type" : "md5" }`
`BSON::Code`	`{ "$code" : "this.v = 5 }`
`BSON::CodeWithScope`	`{ "$code" : "this.v = value", "$scope" : { v => 5 }}`
`BSON::MaxKey`	`{ "$maxKey" : 1 }`
`BSON::MinKey`	`{ "$minKey" : 1 }`
`BSON::ObjectId`	`{ "$oid" : "4e4d66343b39b68407000001" }`
`BSON::Timestamp`	`{ "t" : 5, "i" : 30 }`
`Regexp`	`{ "$regex" : "[abc]", "$options" : "i" }`

Special Ruby Date Classes

Ruby's `Date` and `DateTime` are able to be serialized, but when they are deserialized they will always be returned as a `Time` since the BSON specification only has a `Time` type and knows nothing about Ruby.

Using Regexes

Ruby regular expressions always have BSON regular expression's equivalent of 'm' on. In order for behavior to be preserved between the two, the 'm' option is always added when a Ruby regular expression is serialized to BSON.

There is a class provided by the bson gem, `Regexp::Raw`, to allow Ruby users to get around this. You can simply create a regular expression like this:

```
Regexp::Raw.new("^b403158")
```

This code example illustrates the difference between serializing a core Ruby `Regexp` versus a `Regexp::Raw` object:

```
regexp_ruby = /^b403158/
# => /^b403158/
regexp_ruby.to_bson
# => #<BSON::ByteBuffer:0x007fcf20ab8028>
_.to_s
# => "^b403158\x00m\x00"
regexp_raw = Regexp::Raw.new("^b403158")
# => #<BSON::Regexp::Raw:0x007fcf21808f98 @pattern="^b403158", @options="">
regexp_raw.to_bson
# => #<BSON::ByteBuffer:0x007fcf213622f0>
_.to_s
# => "^b403158\x00\x00"
```

Please use the `Regexp::Raw` class to instantiate your BSON regular expressions to get the exact pattern and options you want.

When regular expressions are deserialized, they return a wrapper that holds the raw regex string, but does not compile it. In order to get the Ruby `Regexp` object, one must call `compile` on the returned object.

```
regex = Regexp.from_bson(byte_buffer)
regex.pattern #=> Returns the pattern as a string.
regex.options #=> Returns the raw options as a String.
regex.compile #=> Returns the compiled Ruby Regexp object.
```