BSON 3.0.0 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', '~> 3.0'
```

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. 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 the `StringIO` bytes.

```
String.from_bson(string_io)
Int32.from_bson(string_io)
```

Core Ruby objects that are represented in the BSON specification and 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
```

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.

Compiling Regexes

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(io)
regex.pattern # Returns the pattern as a string.
regex.options # Returns the raw options as an int.
regex.compile # Returns the compiled Ruby Regexp object.
```