# class BSON::Document

BSON Encodable and Decodable document

## Table of Contents

```
package BSON { class Document does Associative does Positional { ... } }
```

Use as

```
use BSON::Document;

# Document usage
my BSON::Document $d .= new;
$d<name> = 'Jose';
$d<address> = street => '24th', city => 'NY';
$d<keywords> = [<perl6 language programming>];

# Automatic generating subdocuments
$d.autovivify(True);
$d<a><b><c><d><e> = 10;

# Encoding and decoding
my Buf $b = $d.encode;
my BSON::Document $d2 .= new;
$d2.decode($b);
```

Document storage with Hash like behavior used mainly to communicate with a mongodb server. It can also be used as a serialized storage format. The main difference with the Hash is that this class keeps the input order of inserted key-value pairs which is important for the use with mongodb.

Every form of nesting with e.g. pairs is converted into a BSON::Document. Other classes might be needed to handle types such as Javascript, ObjectId and Binary. These classes are automatically loaded when `BSON::Document` is loaded.

E.g.

```
use BSON::Document;

my BSON::Document $d .= new;
$d<javascript> = BSON::Javascript.new(:javascript('function(x){return x;}'));
$d<datetime> = DateTime.now;
$d<rex> = BSON::Regex.new( :regex('abc|def'), :options<is>);
```

## SUPPORTED TYPES

The type which are currently supported are marked with a [x]. [-] will not be implemented and [ ] is a future thingy.

```
    Encoding perl6 to a BSON bytestream
    Perl6                BSON

[x] Num                  64-bit Double.
[x] Str                  UTF-8 string
[x] Seq                  Embedded document
[x] Pair                 Embedded document
[x] Hash                 Embedded document, throws an exception by default.
[x] Array                Array document
[x] BSON::Binary         All kinds of binary data
[x]                        Generic type
[ ]                        Function
[-]                        Binary old, deprecated
[-]                        UUID old, deprecated
[x]                        UUID
[x]                        MD5
[x] BSON::ObjectId       ObjectId
[x] Bool                 Boolean "true" / "false"
[x] DateTime             int64 UTC datetime, seconds since January 1st 1970
[x] Undefined type       Null value
[x] BSON::Regex          Regular expression for serverside searches
[x] BSON::Javascript     Javascript code transport with or whithout scope
[x] Int                  32-bit Integer if -2147483646 < n < 2147483647
                         64-bit Integer if -9,22337203685e+18 < n < 9,22337203685e+18
                         9,22337203685e+18 Fails at the moment if
                         larger/smaller with an exception X::BSON::ImProperUse.
                         Later it might be stored as a binary.
```

There are BSON specifactions mentioned on their site which are deprecated or used internally only. These are not implemented.

There are quite a few more perl6 container types like Rat, Bag, Set etc. Now binary types are possible it might be an idea to put these perl6 types into binary. There are 127 user definable types in that BSON binary specification, so place enough.

```
    Decoding a bytestream to perl6
    BSON                                 Perl6

[x] 64-bit Double                        Num
[x] UTF-8 string                         Str
[x] Embedded document.                   BSON::Document
[x] Array document                       Array
[x] All kinds of binary data             BSON::Binary
[x]   Generic type                       BSON::Binary
[ ]   Function                           BSON::Binary
[-]   Binary old, deprecated             BSON::Binary
[-]   UUID old, deprecated               BSON::Binary
[x]   UUID                               BSON::Binary
[x]   MD5                                BSON::Binary
[x] ObjectId                             BSON::ObjectId
[x] Boolean "true" / "false"             Bool
[x] int64 UTC datetime                   DateTime
[x] Null value                           Undefined type
[x] Regular expression                   BSON::Regex
[x] Javascript code                      BSON::Javascript
[x] 32 bit integers.                     Int
[x] 64 bit integers.                     Int

[ ]                                      FatRat
[ ]                                      Rat
```

## OPERATIONS

### postcircumfix:<{}>

```
$d{'full address'} = 'my-street 45, new york';
```

### postcircumfix:<<>>

```
$d<name> = 'Marcel Timmerman';
```

### postcircumfix:<[]>

```
use Test;
use BSON::Document;

my BSON::Document $d .= new;
$d<abc> = 10;                    # Key 'abc' on location 0
$d<def> = 11;                    # Key 'def' on location 1
$d[1] = 'Timezone of New York'; # Modify location 1 which is key 'def'
$d[100] = 'new data';           # New location 2 with generated key 'key100'.

is $d[100], $d<key100>, "Location 100 is same as pointed by 'key100'";
is $d.find-key('key100'), 2, 'Index is 2 instead of 100';
is $d[100], $d[2], "Location 100 is same as pointed by index 2";
is $d.find-key(100), 'key100', 'Check $d[100] to be key100';
```

Modify or create locations using an index into the document. When locations exist, data at that location is overwritten by the new data. Non-existent locations are set as the next free location in the document and a key is generated using the index prefixed with 'key' (depending on autovivify).

## METHODS

### method new

Defined as

```
multi method new ( List $l = () )
multi method new ( Pair $p )
multi method new ( Seq $s )
multi method new ( Buf $b )
```

Use as

```
my BSON::Document $d;

# empty document
$d .= new;

# Initialize with a Buf, Previously received from a mongodb server or
# from a previous encoding
$d .= new($bson-encoded-document);

# Initialize with a Seq
$d .= new: ('a' ... 'z') Z=> 120..145;

# Initialize with a List
$d .= new: ( a => 10, b => 11);
```

Initialize a new document.

### method perl

Defined as

```
method perl ( --> Str )
```

Return objects structure.

## method WHAT

Defined as

```
method WHAT ( --> Str )
```

Return type of the object. This is '(BSON::Document)'.

## method Str

Defined as

```
method Str ( --> Str )
```

Return type and location of the object.

## method autovivify

Defined as

```
submethod autovivify ( Bool $avvf = True )
```

By default it is set to `False` and will throw an exception with an message like 'Cannot modify an immutable Any' when an attempt is made like in the following.piece of code

```
my BSON::Document $d .= new;
$d<a><b> = 10;                    # Throw error
```

To have this feature one must turn this option on like so;

```
my BSON::Document $d .= new;
$d.autovivify(True);
$d<a><b> = 10;
```

NOTE: Testing for items will also create the entries if they weren't there.

## method accept-hash

Defined as

```
submethod accept-hash ( Bool $acch = True )
```

By default it is set to `False` and will throw an exception with a message like 'Cannot use hash values'. This is explicitly done to keep input order. When it is turned off try something like below to see what is meant;

```
my BSON::Document $d .= new;
$d.accept-hash(True);
$d<q> = {
  a => 120, b => 121, c => 122, d => 123, e => 124, f => 125, g => 126,
  h => 127, i => 128, j => 129, k => 130, l => 131, m => 132, n => 133,
  o => 134, p => 135, q => 136, r => 137, s => 138, t => 139, u => 140,
  v => 141, w => 142, x => 143, y => 144, z => 145
};

say $d<q>.keys;
# Outputs [x p k h g z a y v s q e d m f c w o n u t b j i r l]
```

## method find-key

Defined as

```
multi method find-key ( Int:D $idx --> Str )
multi method find-key ( Str:D $key --> Int )
```

Search for indes and find key or search for key and return index. It returns an undefined value if $idx or $key is not found.

```
use Test;
use BSON::Document;
my $d = BSON::Document.new: ('a' ... 'z') Z=> 120..145;

is $d<b>, $d[$d.find-key('b')], 'Value on key and found index are the same';
is $d.find-key(2), 'c', "Index 2 is mapped to key 'c'";
```

## method of

Defined as

```
method of ( )
```

Returns type of object. NOTE: I'm not sure if this is the normal practice of such a method. Need to investicate further

## method elems

Defined as

```
method elems ( --> Int )
```

Return the number of pairs in the document

## method kv

Defined as

```
method kv ( --> List )
```

Return a list of keys and values in the same order as entered.

```
use BSON::Document;
my $d = BSON::Document.new: ('a' ... 'z') Z=> 120..145;
say $d.kv;
# Outputs: [a 120 b 121 c 122 d 123 ... x 143 y 144 z 145]
```

## method pairs

Defined as

```
method pairs ( --> List )
```

Return a list of pairs in the same order as entered.

## method keys

Defined as

```
method keys ( --> List )
```

Return a list of keys in the same order as entered.

```
use BSON::Document;
my $d = BSON::Document.new: ('a' ... 'z') Z=> 120..145;
say $d.keys;
# Outputs: [a b c d ... x y z]
```

## method values

Defined as

```
method values ( --> List )
```

Return a list of value in the same order as entered.

```
use BSON::Document;
my $d = BSON::Document.new: ('a' ... 'z') Z=> 120..145;
say $d.values;
# Outputs: [120 121 122 123 ... 143 144 145]
```

## method modify-array

Defined as

```
method modify-array ( Str $key, Str $operation, $data --> List )
```

Use as

```
BSON::Document $d .= new:(docs => []);
$d.modify-array( 'docs', 'push', (a => 1, b => 2));
```

Modify an array in a document afterwards. This method is necessary to apply changes because when doing it directly like **$d<docs>.push: (c** = 2);> it wouldn't be encoded because the document object is not aware of these changes.

This is a slow method because every change will trigger an encoding procedure in the background. When a whole array needs to be entered then it is a lot faster to make the array first and then assign it to an entry in the document e.g;

```
BSON::Document $d .= new;
my $arr = [];
for ^10 -> $i {
  $arr.push($i);
}
$d<myarray> = $arr;
```

### method encode

Defined as

```
method encode ( --> Buf )
```

Encode entire document and return a BSON encoded byte buffer.

### method decode

Defined as

```
method decode ( Buf $data --> Nil )
```

Decode a BSON encoded byte buffer to produce a document. Decoding also takes place when providing a byte buffer to new().

## EXCEPTIONS

### X::Parse-document

This is thrown when the document doesn't parse correctly while encoding or decoding.

### X::NYS

Thrown when encoding stubles upon a variable which is not supported(yet)