# class MongoDB::Database

Operations on a MongoDB database

## Table of Contents



```
package MongoDB { class Database { ... } }
```

# Synopsis

```
# Initialize
my MongoDB::Client $client .= new(:uri<mongodb://>);
my MongoDB::Database $database = $client.database('mydatabase');

# Drop database
$database.run-command: dropDatabase => 1;
```

# Readonly attributes

## name

```
has Str $.name;
```

Stored name of the database.

## client

```
has MongoDB::Client $.client;
```

Client used by database. Set when creating object.

## read-concern

```
has BSON::Document $.read-concern;
```

The read-concern is a structure to have some control over the read operations to which server the operations are directed to. Default is an empty structure. The structure will be explained elsewhere.

# Methods

## new

```
submethod BUILD (
  ClientType:D :$client, Str:D :$name, BSON::Document :$read-concern
)
```

Create a database object. Can be called directly although not done often, e.g.

```
my MongoDB::Database $d .= new( $client, 'my_db');
```

When read-concern is absent it is taken from $client.

## collection

```
method collection (
  Str $name, BSON::Document :$read-concern
  --> MongoDB::Collection
)
```

Select collection and return a collection object. When the collection is new it will only be created when data is inserted.

When read-concern is absent it is taken from its $database.

## run-command

```
multi method run-command (
  BSON::Document:D $command,
  BSON::Document :$read-concern
  --> BSON::Document
)

multi method run-command ( |command-capture --> BSON::Document )
```

Run a command against the database. For proper handling of this command it is nesseary to study the documentation on the MongoDB site. A good starting point is at this page.

The command argument is a BSON::Document or List of Pair of which the latter might be more convenient. Mind the comma's when describing list of one Pair! This is very important see e.g. the following perl6 REPL interaction;

```
> 123.WHAT.say
(Int)
> (123).WHAT.say
(Int)
> (123,).WHAT.say     # Only now it becomes a list
(List)

> (a => 1).WHAT.say
(Pair)
> (a => 1,).WHAT.say  # Again, with comma it becomes a list
(List)
```

See also Perl6 docs here and here

First example to insert a document using a BSON::Document See also information here.

```
# Method 1. With info from http://perldoc.perl.org/perlhist.html
# There are tests using the Test package
#
# Insert a document into collection 'famous_peaople'
BSON::Document $req .= new: (
  insert => 'famous_people',
  documents => [
    BSON::Document.new((
      name => 'Larry',
      surname => 'Walll',
      languages => BSON::Document.new((
        Perl0 => 'introduced Perl to my officemates.',
        Perl1 => 'introduced Perl to the world',
        Perl2 => 'introduced Henry Spencer\'s regular expression package.',
        Perl3 => 'introduced the ability to handle binary data.',
        Perl4 => 'introduced the first Camel book.',
        Perl5 => 'introduced everything else,'
                 ~ ' including the ability to introduce everything else.',
        Perl6 => 'A perl changing perl event, Dec 24, 2015'
      )),
    )),
  ]
);

# Run the command with the insert request
BSON::Document $doc = $database.run-command($req);
is $doc<ok>, 1, "Result is ok";
is $doc<n>, 1, "Inserted 1 document";
```

As you can see above, it might be confusing how to use the round brackets (). Normally when a (sub)method or sub is called you have positional and named arguments. A named argument is like a pair. So to provide a pair as a positional argument, the pair must be enclosed between an extra pair of round brackets. E.g. **$some-array.push(($some-key => $some-value));**. There is a nicer form using a colon ':' e.g. **$some-array.push: ($some-key => $some-value);**. This is done above on the first line. However, this is not possible at the inner calls because these round brackets also delimit the pairs in the list to the new() method.

The second method is easier using List of Pair not only for the run-command but also in place of nested BSON:Document's. Now we use findAndModify (see documentation here) to repair Larry's surname which should be Wall instead of Walll.

```
# Method 2 using List of Pair. We need to repair our spelling mistake of
# mr Walls name
#
# Directly use run-command instead of making a request BSON::Document
my BSON::Document $doc = $database.run-command: (
  findAndModify => 'famous_people',
  query => (surname => 'Walll'),
  update => ('$set' => (surname => 'Wall')),
);

is $doc<ok>, 1, "Result is ok";
is $doc<value><surname>, 'Walll', "Old data returned";
is $doc<lastErrorObject><updatedExisting>,
   True,
   "Existing document updated";
```

Please also note that mongodb uses query selectors such as $set above and virtual collections like $cmd. Because they start with a '$' these must be protected against evaluation by perl using single quotes.