Casbah (MongoDB + Scala Toolkit) Documentation

Release 2.7.0-RC2

10gen, Inc.

March 13, 2014

Welcome to the Casbah Documentation. Casbah is a Scala toolkit for MongoDB — We use the term "toolkit" rather than "driver", as Casbah is a layer on top of the official mongo-java-driver for better integration with Scala. This is as opposed to a native implementation of the MongoDB wire protocol, which the Java driver does exceptionally well. Rather than a complete rewrite, Casbah uses implicits, and *Pimp My Library* code to enhance the existing Java code.

Tutorial: Using Casbah A quick tutorial to get you started using Casbah.

User Guide The full guide to Casbah - covering installation, connecting, the query dsl , gridfs, and *everything* between.

ScalaDocs The complete ScalaDocs for Casbah along with SXR cross referenced source.

Changelog The recent changes to Casbah

Whats new in Casbah An indepth review of new features in MongoDB and Casbah

CHAPTER

ONE

HELP AND SUPPORT

For help and support using casbah please send emails / questions to the Casbah Mailing List on Google Groups. Also be sure to subscribe to the usergroup to get the latest news and information about Casbah.

Stackoverflow is also a great resource for getting answers to your casbah questions - just be sure to tag any questions with "casbah". Just don't forget to mark any answered questions as answered!

CHAPTER

TWO

CONTRIBUTING

The source is available on GitHub and contributions are always encouraged. Contributions can be as simple as minor tweaks to the documentation, feature improments or updates to the core.

To contribute, fork the project on GitHub and send a pull request.

CHAPTER

THREE

OFFLINE READING

Download the docs in either PDF or ePub formats for offline reading.

3.1 Tutorial: Using Casbah

This quick tutorial should get you up and running doing basic create, read, update, delete (CRUD) operations with Casbah.

3.1.1 Prerequisites

Please ensure you have downloaded and installed mongodb and have it running on its default host (localhost) and port (27107).

3.1.2 Getting started

The next step is to get and install sbt, create an sbt project and install casbah. I recommend using sbt-extras - a special sbt script for installing and running sbt.

1. Create a project directory: `mkdir casbah_tutorial && cd casbah_tutorial`
2. Install sbt-extras script:

```
curl https://raw.github.com/paulp/sbt-extras/master/sbt > sbt
chmod +ux sbt
```

2. Create an sbt build file: `build.sbt`:

```
name := "Casbah Tutorial"

version := "0.1"

scalaVersion := "2.10.3"

libraryDependencies += "org.mongodb" %% "casbah" % "2.7.0-RC2"
```

3. Run the console and test (sbt will automatically install the dependencies):

```
$ ./sbt console
scala> import com.mongodb.casbah.Imports._
import com.mongodb.casbah.Imports._
```

If you had any errors installing casbah please refer to the *installation guide*, otherwise you are ready to start using casbah!

3.1.3 Connecting to MongoDB

The first step of using Casbah is to connect to MongoDB. Remember, this tutorial expects MongoDB to be running on localhost on port 27017. MongoClient is the connection class.

Load the scala shell `./sbt console`

```
import com.mongodb.casbah.Imports._
val mongoClient = MongoClient("localhost", 27017)
```

There are various connection configuration options see the *connection guide* for more information.

Note: The scala repl has tab completion type: `mongoClient.<tab>` for a list of all the operations you can run on a connection.

3.1.4 Getting databases and collections

In MongoDB a database doesn't need to exist prior to connecting to it, simply adding documents to a collection is enough to create the database.

Try connecting to the "test" database and getting a list all the collections

```
val db = mongoClient("test")
db.collectionNames
```

If your database is new then `db.collectionNames` will return an empty `Set`, otherwise it will list the collections in the database.

The next step before starting to add, update and remove documents is to get a collection

```
val coll = db("test")
```

`coll` is the "test" collection in the "test" database. You are now ready to begin adding documents.

Note: If you had an existing "test" collection drop it first: `coll.drop()`

3.1.5 Doing CRUD operations

Inserting, reading, updating and deleting documents in MongoDB is simple. The `MongoDBObject` is a Map-like object that represents a MongoDB Document.

Create

Create two documents `a` and `b`:

```
val a = MongoDBObject("hello" -> "world")
val b = MongoDBObject("language" -> "scala")
```

Insert the documents:

```
coll.insert( a )
coll.insert( b )
```

Read

Count the number of documents in the "test" collection:

```
coll.count()
```

Use `find` to query the database and return an iterable cursor, then print out the string representation of each document:

```
val allDocs = coll.find()
println( allDocs )
for(doc <- allDocs) println( doc )
```

Note: You may notice an extra field in the document: `_id`. This is the primary key for a document, if you don't supply an `_id` an `ObjectId` will be created for you.

By providing a `MongoDBObject` to the `find` method you can filter the results:

```
val hello = MongoDBObject("hello" -> "world")
val helloWorld coll.findOne( hello )

// Find a document that doesn't exist
val goodbye = MongoDBObject("goodbye" -> "world")
val goodbyeWorld coll.findOne( goodbye )
```

Note: Notice that `find` returns a Cursor and `findOne` returns an `Option`.

Update

Now you have some data in MongoDB, how do you change it? MongoDB provides a powerful `update` method that allows you to change single or multiple documents.

First, find the scala document and add its platform:

```
val query = MongoDBObject("language" -> "scala")
val update = MongoDBObject("platform" -> "JVM")
val result = coll.update( query, update )

println("Number updated: " + result.getN)
for (c <- coll.find) println(c)
```

Warning: You will notice that the document is now missing `"language" -> "scala"`! This is because when using update if you provide a simple document it will replace the existing one with the new document. This is the most common gotcha for newcomers to MongoDB.

MongoDB comes with a host of update operators to modify documents. Casbah has a powerful *DSL* for creating these update documents. Lets set the language to scala for the JVM document:

```
val query = MongoDBObject("platform" -> "JVM")
val update = $set("language" -> "Scala")
val result = coll.update( query, update )

println( "Number updated: " + result.getN )
for ( c <- coll.find ) println( c )
```

Note: By default `update` will only update a single document - to update *all* the documents set the multi flag: `.update( query, update, multi=true)`.

Another useful feature of the `update` command is it also allows you to `upsert` documents on the fly. Setting `upsert=True` will insert the document if doesn't exist, otherwise update it:

```
val query = MongoDBObject("language" -> "clojure")
val update = $set("platform" -> "JVM")
```

```
val result = coll.update( query, update, upsert=true )

println( "Number updated: " + result.getN )
for (c <- coll.find) println(c)
```

Removing

The final part of the tutorial is removing documents. Remove is the similar to `find`, in that you provide a query of documents to match against:

```
val query = MongoDBObject("language" -> "clojure")
val result = coll.remove( query )

println("Number removed: " + result.getN)
for (c <- coll.find) println(c)
```

To remove all documents, provide a blank document to match all items in the database:

```
val query = MongoDBObject()
val result = coll.remove( query )

println( "Number removed: " + result.getN )
println( coll.count() )
```

Rather than iterating the collection and removing each document, its more efficient to drop the collection:

```
coll.drop()
```

3.1.6 Learning more about Casbah

If you got this far you've made a great start, so well done! The next step on your Casbah journey is the *full user guide*, where you can learn indepth about how to use casbah and mongodb.

3.2 User Guide

Casbah is a Scala toolkit for MongoDB — We use the term "toolkit" rather than "driver", as Casbah is a layer on top of the official mongo-java-driver for better integration with Scala. This is as opposed to a native implementation of the MongoDB wire protocol, which the Java driver does exceptionally well. Rather than a complete rewrite, Casbah uses implicits, and *Pimp My Library* code to enhance the existing Java code.

3.2.1 Philosophy

Casbah's approach is intended to add fluid, Scala-friendly syntax on top of MongoDB and handle conversions of common types.

If you try to save a Scala List or Seq to MongoDB, we *automatically convert it* to a type the Java driver can serialise. If you read a Java type, we convert it to a comparable Scala type before it hits your code.

All of this is intended to let you focus on writing the best possible Scala code using Scala idioms. A great deal of effort is put into providing you the functional and implicit conversion tools you've come to expect from Scala, with the power and flexibility of MongoDB.

Casbah provides improved interfaces to *GridFS*, Map/Reduce and the core Mongo APIs. It also provides a *fluid query syntax* which emulates an internal DSL and allows you to write code which is more akin to what you would write in the JS Shell.

There is also support for easily adding new *serialisation/deserialisation* mechanisms for common data types (including Joda Time, if you so choose; with some caveats - See the *GridFS Section*).

3.2.2 Installation

Casbah is released to the Sonatype repository, the latest Casbah build as is 2.7.0-RC2 and supports the following scala versions: 2.9.3 and 2.10.3.

The easiest way to install the latest Casbah driver (2.7.0-RC2) is by using sbt - the Scala Build Tool.

Setting up via sbt

Once you have your sbt project setup - see the sbt setup guide for help there.

Add Casbah to sbt to your `./build.sbt` file:

```
libraryDependencies += "org.mongodb" %% "casbah" % "2.7.0-RC2"
```

Note: The double percentages (%%) is not a typo—it tells sbt that the library is crossbuilt and to find the appropriate version for your project's Scala version.

To test your installation load the sbt console and try importing casbah:

```
$ sbt console
scala> import com.mongodb.casbah.Imports._
import com.mongodb.casbah.Imports._
```

Problem solving

If sbt can't find casbah then you may have an older version of sbt and will need to add the sonatype resolvers to your *./build.sbt* file:

```
// For stable releases
resolvers += "Sonatype releases" at "https://oss.sonatype.org/content/repositories/releases"
// For SNAPSHOT releases
resolvers += "Sonatype snapshots" at "https://oss.sonatype.org/content/repositories/snapshots"
```

Alternative installation methods

You can install Casbah with maven, ivy or from source - see the *alternative install* documentation.

3.2.3 Alternative installation methods

Casbah is released to the Sonatype repository, the latest Casbah build as is 2.7.0-RC2 and supports the following scala versions: 2.9.3, 2.10.x.

Using Dependency/Build Managers

First, you should add the package repository to your Dependency/Build Manager. Our releases & snapshots are currently hosted at Sonatype; they should eventually sync to the Central Maven repository.:

```
https://oss.sonatype.org/content/repositories/releases/  /* For Releases */
https://oss.sonatype.org/content/repositories/snapshots/ /* For snapshots */
```

Set both of these repositories up in the appropriate manner - they contain Casbah as well as any specific dependencies you may require.

Setting Up Maven

You can add Casbah to Maven with the following dependency block.

Please substitute $SCALA_VERSION$ with your Scala version (we support 2.9.x+)

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>casbah_$SCALA_VERSION$</artifactId>
    <version> 2.7.0-RC2 </version>
    <type>pom</type>
<dependency>
```

Setting Up Ivy (w/ Ant)

You can add Casbah to Ivy with the following dependency block.

Please substitute $SCALA_VERSION$ with your Scala version (we support 2.9.x+)

```
<dependency org="org.mongodb" name="casbah_$SCALA_VERSION$" rev="2.7.0-RC2"/>
```

Setting up without a Dependency/Build Manager (Source + Binary)

All Dependencies Jar

As Casbah is published in multiple modules installing it manually can take time, especially as the dependencies change depending on the Scala version you are using. To simplify this you can download a single all inclusive jar for your scala version:

> http://oss.sonatype.org/content/repositories/releases/org/mongodb/casbah_2.10/2.6.2/casbah-alldep_2.10-2.7.0-RC2.jar http://oss.sonatype.org/content/repositories/releases/org/mongodb/casbah_2.9.2/2.6.2/casbah-alldep_2.9.3-2.7.0-RC2.jar

Once the jar is on your class path you will be able to use Casbah.

Building from source

You can always get the latest source for Casbah from the github repository:

```
$ git clone git://github.com/mongodb/casbah
```

The *master* branch is once again the leading branch suitable for snapshots and releases and should be considered (and kept) stable.

3.2.4 Casbah Modules

While Casbah has many stable of features, some users (such as those using a framework like Lift which already provides MongoDB wrappers) wanted access to certain parts of Casbah without importing the whole system. As a result, Casbah has been broken out into several modules which make it easier to pick and choose the features you want.

If you use the individual modules you'll need to use the import statement from each of these. If you use the import statement from the *casbah-core* module, everything except GridFS will be imported (not everyone uses GridFS so we don't load it into memory & scope unless it is needed).

The module names can be used to select which dependencies you want from maven/ivy/sbt, as we publish individual artifacts. If you import just *casbah*, this is a master pom which includes the whole system and will install all its dependencies, as such there is no single jar file for Casbah.

This is the breakdown of dependencies and packages:

Module	Package	Dependencies
Casbah Core **NOTES** Provides Scala-friendly wrappers to the Java Driver for connections, collections and MapReduce jobs	com.mongodb.casbah	casbah-commons and casbah-query along with their dependencies transitively
Casbah Commons **NOTES** Provides Scala-friendly DBObject & DBList implementations as well as Implicit conversions for Scala types	com.mongodb.casbah.commons	• mongo-java-driver, • nscala-time, • slf4j-api, • slf4j-jcl
Query DSL **NOTES** Provides a Scala syntax enhancement mode for creating MongoDB query objects using an Internal DSL supporting Mongo *$ Operators*	com.mongodb.casbah.query	casbah-commons along with their dependencies transitively
Gridfs **NOTES** Provides Scala enhanced wrappers to MongoDB's GridFS filesystem	com.mongodb.casbah.gridfs	casbah-commons and casbah-query along with their dependencies transitively

We cover the import of each module in their appropriate tutorials, but each module has its own *Imports* object which loads all of its necessary code. By way of example both of these statements would import the Query DSL:

```
// Imports core, which grabs everything including Query DSL
import com.mongodb.casbah.Imports._

// Imports just the Query DSL along with Commons and its dependencies
import com.mongodb.casbah.query.Imports._

// Import GridFS modules
import com.mongodb.casbah.gridfs.Imports._
```

3.2.5 Connecting to MongoDB

The core connection class is MongoClient. The casbah `MongoClient` class simply wraps the MongoClient Java class and provides a couple of scala helpers as well.

`MongoClient` is available in the global imports class:

```
import com.mongodb.casbah.Imports._
```

Simple connections

Below are some example connecting to MongoDB with Casbah:

```
// Connect to default - localhost, 27017
val mongoClient =  MongoClient()

// connect to "mongodb01" host, default port
val mongoClient =  MongoClient("mongodb01")

// connect to "mongodb02" host, port 42017
val mongoClient =  MongoClient("mongodb02", 42017)
```

MongoDB URI

As an alternative to providing host and port information, the mongodb URI format defines connections between applications and MongoDB. In Casbah the com.mongodb.casbah.MongoClientURI class handles string URI's:

```
val uri = MongoClientURI("mongodb://localhost:27017/")
val mongoClient =  MongoClient(uri)
```

Note: URI style strings supports all the various connection scenarios, such as connecting to replicasets or using authentication and as such its often considered easier to use.

The following examples show both the long hand way of connecting purely in code and the URI style.

Connecting to ReplicaSets / mongos

The java driver automatically determines if it is speaking to a replicaset or a mongos and acts accordingly.

List of ServerAddress instances

```
val rs1 = new ServerAddress("localhost", 27017)
val rs2 = new ServerAddress("localhost", 27018)
val rs3 = new ServerAddress("localhost", 27019)
val mongoClient = MongoClient(List(rs1, rs2, rs3))
```

Note: The ServerAddress class isn't wrapped by casbah - so you have to call *new* eg: `new ServerAddress()`.

URI style connections

```
val uri = MongoClientURI("mongodb://localhost:27017,localhost:27018,localhost:27019/")
val mongoClient = MongoClient(uri)
```

Authentication

MongoDB currently provides two different authentication mechanisms. Challenge response and GSSAPI authentication (available in the subscriber edition). A commandline example of using GSSAPI authentication can be found in the `examples`.

MongoDBCredentials

```
// Challenge Response
val server = new ServerAddress("localhost", 27017)
val credentials = MongoCredential.createMongoCRCredential(userName, database, password)
val mongoClient = MongoClient(server, List(credentials))

// X.509 Protocol
val server = new ServerAddress("localhost", 27017)
val credentials = MongoCredential.createMongoX509Credential(userName)
val mongoClient = MongoClient(server, List(credentials))

// SASL PLAIN
val server = new ServerAddress("localhost", 27017)
val credentials = MongoCredential.createPlainCredential(userName, source, password)
val mongoClient = MongoClient(server, List(credentials))
```

```
// GSSAPI
val server = new ServerAddress("localhost", 27017)
val credentials = MongoCredential.createGSSAPICredential(userName)
val mongoClient = MongoClient(server, List(credentials))
```

Note: GSSAPI requires the kerberos to be configured correctly in java. Either via flags when running scala:

```
-Djava.security.krb5.realm=EXAMPLE.COM -Djava.security.krb5.kdc=kdc.example.com -Djavax.security.
```

or in scala:

```
System.setProperty("java.security.krb5.realm", "EXAMPLE.COM")
System.setProperty("java.security.krb5.kdc", "kdc.example.com")
System.setProperty("javax.security.auth.useSubjectCredsOnly", "false")
```

To change Service Name (SPN) with kerberos set the *mechanism property* on the credential eg:

```
val credential = MongoCredential.createGSSAPICredential(userName)
credential.withMechanismProperty(key, value)
```

URI style connections

```
// Challenge Response
val uri = MongoClientURI("mongodb://username:pwd@localhost/?authMechanism=MONGODB-CR")
val mongoClient =  MongoClient(uri)

// GSSAPI
val uri = MongoClientURI("mongodb://username%40domain@kdc.example.com/?authMechanism=MONGODB-GSSA
val mongoClient =  MongoClient(uri)
```

SSL connections

By default ssl is off for mongodb, but you can configure mongodb to enable ssl. Subscribers to the enterprise edition of mongodb have ssl support baked in.

MongoClientOptions

```
val options = MongoClientOptions(socketFactory=SSLSocketFactory.getDefault())
val client = MongoClient(serverName, options)
```

URI style connections

```
val uri = MongoClientURI("mongodb://localhost:27017/?ssl=true")
val mongoClient = MongoClient(uri)
```

Note: Ensure your keystore is configured correctly to validate ssl certificates

---

Connection Options

There are extra configuration options for connections, which cover setting the default write concern and read preferences to configuring socket timeouts.

For the more connection options see the mongodb connection reference.

Databases and Collections

To query mongodb you need a collection to query against. Collecions are simple to get from a connection, first get the database the collection is in, then get the collection:

```
val mongoClient = MongoClient()
val db = mongoClient("databaseName")
val collection = db("collectionName")
```

3.2.6 Querying

Query operations

As Casbah wraps the Java driver, so querying against MongoDB is essentially the same. The following methods for finding data:

- `find` Returns a cursor
- `findOne` Returns an Option - either Some(MongoDBObject) or None
- `findById` Returns an Option - either Some(MongoDBObject) or None
- `findAndModify` Finds the first document in the query, updates and returns it.
- `findAndRemove` Finds the first document in the query, removes and returns it.

The following methods insert and update data:

- `save` Saves an object to the collection
- `insert` Saves one or more documents to the collection
- `update` Updates any matching documents operation

For more information about create, read, update and delete (CRUD) operations in MongoDB see the core operations documentation.

The collection API documentation has a full list of methods and their signatures for interacting with collections.

Bulk Operations

Mongodb 2.6 introduces operations. The bulk operations builder can be used to construct a list of write operations to perform in bulk for a single collection. Bulk operations come in two main flavors.

1. Ordered bulk operations. These operations execute all the operation in order and error out on the first write error.
2. Unordered bulk operations. These operations execute all the operations in parallel and aggregates up all the errors. Unordered bulk operations do not guarantee order of execution.

An example of using the bulk api:

```
val collection = MongoClient()("test")("bulkOperation")

collection.drop()

val builder = collection.initializeOrderedBulkOperation
builder.insert(MongoDBObject("_id" -> 1))
builder.insert(MongoDBObject("_id" -> 2))
builder.insert(MongoDBObject("_id" -> 3))

builder.find(MongoDBObject("_id" -> 1)).updateOne($set("x" -> 2))
builder.find(MongoDBObject("_id" -> 2)).removeOne()
builder.find(MongoDBObject("_id" -> 3)).replaceOne(MongoDBObject("_id" -> 3, "x" -> 4))
```

```
val result = builder.execute()
```

For more information see the bulk operations documentation.

MongoDBObject

MongoDB queries work by providing a document to match against. The simplest query object is an empty one eg: `MongoDBObject()` which matches every record in the database.

MongoDBObject is a simple Map-like class, that wraps the Java driver DBObject and provides some nice Scala interfaces:

```
val query = MongoDBObject("foo" -> "bar") ++ ("baz" -> "qux")
```

DBObjects have a builder and as such you can also build MongoDBObjects that way:

```
val builder = MongoDBObject.newBuilder
builder += "foo" -> "bar"
builder += "baz" -> "qux"
val query = builder.result
```

Note: Remember to import casbah: `import com.mongodb.casbah.Imports._`

3.2.7 Query DSL

Casbah provides a rich fluid query syntax, that allows you to construct `DBObjects` on the fly using MongoDB query operators.

Query Selectors

Comparison Operators

- `$all` Matches arrays that contain all elements specified:

  ```
  "size" $all ("S", "M", "L")
  ```

- `$eq` Matches values that are equal to the value specified:

  ```
  "price" $eq 10
  ```

- `$gt` Matches values that are greater than the value specified in the query
- `$gte` Matches values that are equal to or greater than the value specified:

  ```
  "price" $gt 10
  "price" $gte 10
  ```

- `$in` Matches any of the values that exist in an array specified:

  ```
  "size" $in ("S", "M", "L")
  ```

- `$lt` Matches values that are less than the value specified in the query
- `$lte` Matches values that are less than or equal to the value specified:

  ```
  "price" $lt 100
  "price" $lte 100
  ```

- `$ne` Matches all values that are not equal to the value specified:

```
"price" $ne 1000
```

- `$nin` Matches values that **do not** exist in an array specified:

```
"size" $nin ("S", "XXL")
```

Logical Operators

- `$or` Joins query clauses with a logical `OR` returns all documents that match the conditions of either clause:

```
$or( "price" $lt 5 $gt 1, "promotion" $eq true )
$or( ( "price" $lt 5 $gt 1 ) :: ( "stock" $gte 1 ) )
```

- `$and` Joins query clauses with a logical `AND` returns all documents that match the conditions of both clauses:

```
$and( "price" $lt 5 $gt 1, "stock" $gte 1 )
$and( ( "price" $lt 5 $gt 1 ) :: ( "stock" $gte 1 ) )
```

- `$not` Inverts the effect of a query expression and returns documents that do *not* match the query expression:

```
"price" $not { _ $gte 5.1 }
```

- `$nor` Joins query clauses with a logical `NOR` returns all documents that fail to match both clauses:

```
$nor( "price" $eq 1.99 , "qty" $lt 20, "sale" $eq true )
$nor( ( "price" $lt 5 $gt 1 ) :: ( "stock" $gte 1 ) )
```

Element Operators

- `$exists` Matches documents that have the specified field:

```
"qty" $exists true
```

- `$mod` Performs a modulo operation on the value of a field and selects documents with a specified result:

```
"qty" $mod (5, 0)
```

- `$type` Selects documents if a field is of the specified type:

```
"size".$type[BasicDBList]
```

JavaScript Operators

- `$where` Matches documents that satisfy a JavaScript expression:

```
$where("function () { this.credits == this.debits }")
```

- `$regex` Selects documents where values match a specified regular expression. You can also use native regular expressions:

```
"foo" $regex "^bar$"
"foo" $eq "^bar$".r
```

Geospatial Operators

- `$geoWithin` Selects geometries within a bounding GeoJSON geometry:

```
// Create a GeoJson geometry document
var geo = MongoDBObject("$geometry" ->
        MongoDBObject("$type" -> "polygon",
          "coordinates" -> (((GeoCoords(74.2332, -75.23452),
                              GeoCoords(123, 456),
                              GeoCoords(74.2332, -75.23452))))))

// Example $geoWithin Queries
"location" $geoWithin(geo)
"location" $geoWithin $box ((74.2332, -75.23452), (123, 456))
"location" $geoWithin $center ((50, 50), 10)
"location" $geoWithin $centerSphere ((50, 50), 10)
```

- `$geoIntersects` Selects geometries that intersect with a GeoJSON geometry:

```
// Create a GeoJson geometry document
var geo = MongoDBObject("$geometry" ->
            MongoDBObject("$type" -> "polygon",
              "coordinates" -> (((GeoCoords(74.2332, -75.23452),
                                  GeoCoords(123, 456),
                                  GeoCoords(74.2332, -75.23452))))))
val near = "location" $geoIntersects geo
```

- `$near` Returns geospatial objects in proximity to a point:

```
"location" $near (74.2332, -75.23452)
```

- `$nearSphere` Returns geospatial objects in proximity to a point on a sphere:

```
"location" $nearSphere (74.2332, -75.23452)
```

Array Query Operators

- `$elemMatch` Selects documents if element in the array field matches all the specified `$elemMatch` conditions:

```
"colour" $elemMatch (MongoDBObject("base" -> "red", "flash" -> "silver")
```

- `$size` Selects documents if the array field is a specified size:

```
"comments" $size 12
```

Update DSL Operators

Field Operators

- `$inc` Increments the value of the field by the specified amount:

```
$inc("sold" -> 1, "stock" -> -1)
```

- `$rename` Renames a field:

```
$rename("color" -> "colour", "realize" -> "realise")
```

- `$setOnInsert` Sets the value of a field upon documentation creation during an upsert. Has no effect on update operations that modify existing documents:

```
$setOnInsert("promotion" -> "new")
```

- `$set` Sets the value of a field in an existing document

  $set("promotion" -> "sale", "qty" -> 100)

- `$unset` Removes the specified field from an existing document

  $unset("promotion")

Array Update Operators

- `$addToSet` Adds elements to an existing array only if they do not already exist in the set:

```
$addToSet("sizes" -> "L", "colours" -> "Blue")
$addToSet("sizes") $each ("S", "M", "L", "XL")
```

- `$pop` Removes the first or last item of an array:

```
$pop("sizes" -> "L")
```

- `$pull` Removes items from an array that match a query statement:

```
$pull("sizes" -> "L")
$pull("widgets" $gt 2 )
```

- `$pullAll` Removes multiple values from an array:

```
$pullAll("sizes" -> ("S", "XL"))
```

- `$push` Adds an item to an array:

```
$push("sizes" -> "L")
$push("widgets" $gt 2 )
$push("sizes") $each ("S", "M", "L", "XL")
```

- `$pushAll` *Deprecated.* Adds several items to an array:

```
$pushAll("sizes" -> ("S", "XL"))
```

Bitwise Operators

- `$bit` Performs bitwise `AND` and `OR` updates of integer values:

```
$bit("foo") and 5
```

For the full query syntax to MongoDB see the core docs at: docs.mongodb.org

3.2.8 Aggregation Framework

Overview

The MongoDB aggregation framework provides a means to calculate aggregated values without having to use map-reduce. While map-reduce is powerful, it is often more difficult than necessary for many simple aggregation tasks, such as totaling or averaging field values.

If you're familiar with SQL, the aggregation framework provides similar functionality to `GROUP BY` and related SQL operators as well as simple forms of "self joins." Additionally, the aggregation framework provides projection capabilities to reshape the returned data. Using the projections in the aggregation framework, you can add computed fields, create new virtual sub-objects, and extract sub-fields into the top-level of results.

Aggregation Syntax

Conceptually, documents from a collection pass through an aggregation pipeline, which transforms these objects as they pass through. For those familiar with UNIX-like shells (e.g. bash,) the concept is analogous to the pipe (i.e. |) used to string text filters together:

```
db.people.aggregate( [<pipeline>] )
db.runCommand( { aggregate: "people", pipeline: [<pipeline>] } )
```

See the aggregation reference for information about aggregation operations.

Aggregation By Example

First, consider a collection of documents named articles using the following format:

```
import com.mongodb.casbah.Imports._
val db = MongoClient()("test")
val coll = db("aggregate")
coll.drop()

coll += MongoDBObject("title" -> "Programming in Scala" ,
                      "author" -> "Martin",
                      "pageViews" ->  50,
                      "tags" ->  ("scala", "functional", "JVM") ,
                      "body" ->  "...")

coll += MongoDBObject("title" -> "Programming Clojure" ,
                      "author" -> "Stuart",
                      "pageViews" ->  35,
                      "tags" ->  ("clojure", "functional", "JVM") ,
                      "body" ->  "...")

coll += MongoDBObject("title" -> "MongoDB: The Definitive Guide" ,
                      "author" -> "Kristina",
                      "pageViews" ->  90,
                      "tags" ->  ("databases", "nosql", "future") ,
                      "body" ->  "...")
```

The following example aggregation operation pivots data to create a set of author names grouped by tags applied to an article. Call the aggregation framework by issuing the following command:

```
val db = MongoClient()("test")
val coll = db("aggregate")

val results = coll.aggregate(
  List(
    MongoDBObject("$project" ->
      MongoDBObject("author" -> 1, "tags" -> 1)
    ),
    MongoDBObject("$unwind" -> "$tags"),
    MongoDBObject("$group" ->
      MongoDBObject("_id" -> "$tags",
                    "authors" -> MongoDBObject("$addToSet" -> "$author")
      )
    )
  )
);
```

The results of the aggregation themselves can be accessed via `results`.

Aggregation Cursor Interface - new in casbah 2.7

MongoDB 2.6 adds the ability to return a cursor from the aggregation framework. To do that simply use *AggregationOptions* with the aggregation command:

```
val db = MongoClient()("test")
val coll = db("aggregate")
```

```
val aggregationOptions = AggregationOptions(AggregationOptions.CURSOR)
val results = coll.aggregate(
  List(
    MongoDBObject("$project" ->
      MongoDBObject("author" -> 1, "tags" -> 1)
    ),
    MongoDBObject("$unwind" -> "$tags"),
    MongoDBObject("$group" ->
      MongoDBObject("_id" -> "$tags",
                    "authors" -> MongoDBObject("$addToSet" -> "$author")
      )
    )
  ),
  aggregationOptions
);
```

Then the you can iterate the results of the aggregation as a normal cursor:

```
for (result <- results) println(result)
```

To learn more about aggregation see the aggregation tutorial and the aggregation reference documentation.

3.2.9 GridFS

`GridFS` is a specification for storing and retrieving files that exceed the BSON-document size limit of 16MB.

Instead of storing a file in a single document, GridFS divides a file into parts, or chunks, [1] and stores each of those chunks as a separate document. By default GridFS limits chunk size to 256k. GridFS uses two collections to store files. One collection stores the file chunks, and the other stores file metadata.

When you query a GridFS store for a file, the driver or client will reassemble the chunks as needed. You can perform range queries on files stored through GridFS. You also can access information from arbitrary sections of files, which allows you to "skip" into the middle of a video or audio file.

GridFS is useful not only for storing files that exceed 16MB but also for storing any files for which you want access without having to load the entire file into memory. For more information on the indications of GridFS, see faq-developers-when-to-use-gridfs.

Using GridFS in Casbah

GridFS is a separate package in Casbah and to use it you must import it explicitly. See the full gridfs api docs for more information about the package.

Example use case:

```
import java.io.FileInputStream
import com.mongodb.casbah.Imports._
import com.mongodb.casbah.gridfs.Imports._

// Connect to the database
val mongoClient = MongoClient()("test")

// Pass the connection to the GridFS class
val gridfs = GridFS(mongoClient)

// Save a file to GridFS
val logo = new FileInputStream("mongo.png")
val id = gridfs(logo) { f =>
    f.filename = "mongodb_logo.png"
    f.contentType = "image/png"
```

[1] The use of the term *chunks* in the context of GridFS is not related to the use of the term *chunks* in the context of sharding.

```
}

// Find a file in GridFS by its ObjectId
val myFile = gridfs.findOne(id.get.asInstanceOf[ObjectId])

// Or find a file in GridFS by its filename
val myFile = gridfs.findOne("mongodb_logo.png")

// Print all filenames stored in GridFS
for (f <- gridfs) println(f.filename)
```

Joda DateTime

Due to hardcoding in the Java GridFS driver the Joda Time serialization hooks break with GridFS. It tries to explicitly cast certain date fields as a `java.util.Date`. To that end, on all find ops we explicitly unload the Joda Time deserializers and reload them when we're done (if they were loaded before we started). This allows GridFS to always work but *MAY* cause thread safety issues - e.g. if you have another non-GridFS read happening at the same time in another thread at the same time, it may fail to deserialize BSON Dates as Joda DateTime - and blow up. Be careful — generally we don't recommend mixing Joda Time and GridFS in the same JVM at the moment.

3.2.10 Serialisation

As soon as you construct a `MongoClient` object, a few type conversions will be loaded automatically for you - Scala's built-in regular expressions (e.g. `"\\d{4}-\\d{2}-\\d{2}".r` will now serialize to MongoDB automatically with no work from you), as well as a few other things. The general idea is that common Java types (such as ArrayList) will be returned as the equivalent Scala type.

If you find you need to unload the default helpers, you can load and unload them easily:

```
import com.mongodb.casbah.commons.conversions.scala._
DeregisterConversionHelpers()
RegisterConversionHelpers()
```

Joda Time

Many Scala developers tend to prefer Joda time over JDK Dates, you can also explicitly enable serialization and deserialization of them (w/ full support for the Scala-Time wrappers) by an explicit call:

```
import com.mongodb.casbah.commons.conversions.scala._
RegisterJodaTimeConversionHelpers()
```

Once these are loaded, Joda Time (and Scala Time wrappers) will be saved to MongoDB as proper BSON Dates, and on retrieval / deserialisation all BSON Dates will be returned as Joda `DateTime` instead of a JDK Date (aka `java.util.Date`). Because this can cause problems in some instances, you can explicitly unload the Joda Time helpers:

```
import com.mongodb.casbah.commons.conversions.scala._
DeregisterJodaTimeConversionHelpers()
```

If you prefer Joda `LocalDateTime` - theres also a conversion helper for that:

```
import com.mongodb.casbah.commons.conversions.scala._
RegisterJodaLocalDateTimeConversionHelpers()

// Remove the helper
DeregisterJodaLocalDateTimeConversionHelpers()
```

Custom Conversion Helpers

Writing your own conversion helper is relatively easy, simply provide a `BSON.addEncodingHook(encodeType, transformer)` and a `BSON.removeEncodingHooks(encodeType)`.

See the casbah conversions for an example of creating your own.

3.3 Casbah Examples

Alongside the *quick tutorial* there are a few example commandline programs demonstrating using Casbah in various scenarios.

The following examples can be seen on github :

- **mongoexport** An example of the mongoexport program.
- **mongoimport** An example of the mongoimport program.
- **parallelscan** An example of using the parallelScan collection method with futures to handle the processing of the cursor.

3.4 Changelog

3.4.1 Changes in Version 2.7.0

- Added *Casbah Examples* to highlight using Casbah in various scenarios
- Added Collection helper method to support the parallelCollectionScan command (SCALA-139)
- Fixed getAs[Type](“key”) to stop invalid casts to Some(value) (SCALA-136)
- Support vargs for getAs[Type](“keys”) (SCALA-134)
- Support vargs for as[Type](“Keys”) (SCALA-134) (pr/#61)
- Fixed issue with OpLog matching (pr/#63)
- Register the core Serialization helpers only once (SCALA-129)
- Removed scala 2.9.1 and 2.9.2 support, casbah utilises *scala.concurrent.duration* which was added in scala 2.9.3
- Updated nscala-time to 0.6.0 and specs
- Added support for Server automatically abort queries/commands after user-specified time limit (SCALA-118)
- Added support for Aggregation returning a cursor *AggregationCursor* (SCALA-117)
- Added support for *allowDiskUse: true* to the top-level of an aggregate command with the AggregationOptions helper
- Added support *$out* aggregation pipeline operator (SCALA-130)
- Casbah WriteConcern’s *valueOf* now returns an Option (SCALA-127)
- Support the use of a different SPN for Kerberos Authentication (SCALA-103)
- Support SASL PLAIN authentication (SCALA-101)
- Support MONGODB-X509 authentication (SCALA-112)
- Updated Mongo Java Driver to 2.12

3.4.2 Changes in Version 2.6.5

- Updated Mongo Java Driver to 2.11.4

3.4.3 Changes in Version 2.6.4

- Updated Scala 2.10 series to 2.10.3

3.4.4 Changes in Version 2.6.3

- Fixed JodaGridFS when registered helpers are on (SCALA-113)
- Updated Mongo Java Driver to 2.11.3

3.4.5 Changes in Version 2.6.2

- Fixed MongoClientURI Implicit
- Added support for Joda-Time LocalDate serialisation (SCALA-111, #59)
- Added aggregate collection helper (SCALA-110)
- Added $each support to $pull (SCALA-109)
- Updated to the latest Java driver 2.11.2 (SCALA-106)
- Added $eq operator (SCALA-105)
- Fixed $where dsl operator (SCALA-97)

3.4.6 Changes in Version 2.6.1

- Fixed $pushAll and $pullAll casting of iterables (SCALA-54)
- Fixed MongoCollection string representation (SCALA-96)
- Fixed support for jsScope (SCALA-43) (#44)
- Publish casbah.commons test helpers
- Added suport $setOnInsert to the query dsl

3.4.7 Changes in Version 2.6.0

- Added suport for GSSAPI SASL mechanism and MongoDB Challenge Response protocol
- Updated support for latest Java driver 2.11.1

3.4.8 Changes in Version 2.5.1

- Added 2.10.1 support
- Removed reference to scala-tools (SCALA-78)
- Added 2.9.3 support (SCALA-94)
- Removed Specs2 and Scalaz dependencies outside test (SCALA-93)
- Fixed 2.10 support, no need for -Yeta-expand-keeps-star compile flag (SCALA-89)
- Fixed distinct regression (SCALA-92)

- Fixed test data import - now in tests :)

3.4.9 Changes in Version 2.5.0

- Added support for Scala 2.10.0
- Dropped support for Scala 2.9.0
- Dropped support for Scala 2.8.X
- Updated support for latest Java driver 2.10.1
- Added support for the new MongoClient connection class
- Removed scalaj.collections dependency
- Updated to nscala-time
- Updated the build file
- Added unidoc and updated documentation
- Migrated documentation theme
- Updated MongoDBList to handle immutable params
- Maven Documentation fix (SCALA-71)
- MongoOpLog - uses new MongoClient and defaults to replciaSet oplog database

3.4.10 Changes in Version 2.4.1

- Fixed QueryDSL imports for "default" (com.mongodb.casbah.Imports) import so that bareword ops like $set and $inc are available.

3.4.11 Changes in Version 2.4.0

- Hide BasicDBList; now, getAs and As and related will always return a MongoDBList which is a Seq[_]. Enjoy!
- This is an API breakage - you should *never* get back a BasicDBList from Casbah anymore, and asking for one will cause a ClassCastException. This brings us more in line with sane Scala APIs

3.4.12 Changes in Version 2.3.0

BT/Maven Package change. Casbah is now available in: "org.mongodb" %% "casbah" % "2.3.0"

- Update mongo-java-driver to 2.8.0 release
- Updated to Mongo Java Driver 2.8.0-RC1
- Changed some tests to run sequentially to avoid shared variable races.
- JodaGridFS wasn't properly checked in before.
- Updated MongoOptions to sync up with options provided in Java Driver.
- Pre-Beta milestone (linked against unreleased Java Driver release)
- Dropped Scala 2.8.0 support...
  - 2.1.5-1 is the final Casbah release for 2.8.0; please migrate to Scala 2.8.1 or higher
- SCALA-62: Simple solution - hack the date type on the base class.

  - There is now a JodaGridFS implementation which works cleanly with Joda DateTime and will return them to you

- Backport casbah-gridfs from 3.0
  - Fixes SCALA-45: Allow filename and contentType to be nullable
    - Retrieving filename or contentType on a GridFS File now returns Option[String] when fetched
    - To facilitate sane usage, the loan-pattern/execute-around-resource methods now return the _id of the created file as Option[AnyRef]
- Backports to casbah-core from 3.0
  - SCALA-70: Removed type alias to com.mongodb.WriteConcern and made method args for it explicit, as it was causing a fun post-compile (aka "library compiles, user code doesn't") implosion.
  - added socketKeepAlive option
  - Fixes SCALA-45: Allow filename and contentType to be nullable
  - Retrieving filename or contentType on a GridFS File now returns Option[String] when fetched
  - To facilitate sane usage, the loan-pattern/execute-around-resource methods now return the _id of the created file as Option[AnyRef]
- Backports for QueryDSL
  - Major cleanups and bugfixes to the DSL, it's heavily and fully tested now and much faster/cleaner
  - Added support for $and bareword operator
  - SCALA-30, SCALA-59 - $or is not properly accepting nested values esp. from other DSL constructors
    - Introduced proper type class filter base to fix $or, will implement across other operators next.
  - SCALA-59 - Fix Bareword Query Operators to better target accepted values; should only accept KV Tuple Pairs or DBObjects returned from Core Operators
    - Complete test suites for $and and $nor although they need to be updated to more appropriate contextual examples rather than just "compiles properly"
    - New code logic, fixed $or, $and and $nor for proper nested list operations
    - New :: list cons operator on MongoDBObject to create MongoDBLists on th fly (esp. for DSL)
    - Typesafety kungfu from @jteigen
      - enforce at compile time that type parameters used for casting are not Nothing
      - enforce $pushAll & $pullAll arguments can be converted to Iterable at compile time
      - switched to a type class (AsQueryParam) for queryparams to avoid code duplication
- SCALA-69: Maps saved to DBObject are now eagerly converted to a DBObject, from factory, builder and put methods.
- Always return MongoDBList from Factories/Builders instead of Seq[Any]
- Backports from Casbah 3.0
  - Refactor collections (MongoDBList and MongoDBObject)
  - Use CanBuildFrom properly to compose more appropriate Collection objects
  - As part of above, you should get seq-like objects back from MongoDBList builders & factories instead of the previous BasicDBList; this is part of attempting to "Hide" DBList and let people work with List/Seq
  - SCALA-69: Immediately upon saving any None's will be converted to null inside the DBObject for proper fetching later.
  - Add toString, hashCode and equals methods to DBObject

  - New, refactored tests for DBObject and DBList
    - More typesafety kungfu from @jteigen
      - enforce at *compile time* that type parameters used for casting (as, getAs, getAsOrElse) are not Nothing
- Backport Test Helpers
  - New MongoDB "smart" test helpers for Specs2 and ScalaTest (Thanks Bill Venners for the latter)
- Added SBT Rebel cut, local runner

3.4.13 Changes in Version 2.1.5.0

- Added support for Scala 2.9.0-1 ... As this is a critical fix release against 2.9.0.final, 2.9.0.final is not supported. (Note that SBT, etc requires the artifact specified as 2.9.0-1, not 2.9.0.1)
- Apart from BugFixes this will be the last Casbah release which supports Scala 2.8.0; all future releases will require Scala 2.8.1+ (See 2.8.0 EOL Announcement)
- [2.9.0 only] Adjusted dynamic settings to build against 2.9.0-1 and Casbah 2.1.5.0
- [2.9.0 only] Prototype "Dynamic" module (You must enable Scala's support for Dynamic)
- [2.9.0 only] I seem to have missed project files for SBT and casbah-dynamic
- [2.9.0 only] Tweaks and adjustments to get this building and testing solidly on 2.9.0-1
- Disabled a few tests that weren't passing and known to be 'buggy' in specs1. These are fixed for the upcoming 2.2. release on specs2; they are test bugs rather than Casbah bugs.
- RegEx *not was just flat out wrong - was producing {"foo": {"foo": /<regex>/}} instead of {"foo": {*"not":{//}}
- Added a getAsOrElse method

3.4.14 Changes in Version 2.1.0

- SCALA-22 Added a dropTarget boolean option to rename collection, which specifies behavior if named target collection already exists, proxies JAVA-238
- Removed resetIndexCache, which has also been removed from the Java Driver
- SCALA-21 Added "set metadata" method to match Java Driver (See Java-261)
- SCALA-20 Updated to Java Driver 2.5
  - See Release Notes: http://groups.google.com/group/mongodb-user/browse_thread/thread/a693ad4fdf9c3731/931f46f7213b6775?show_docid=931f46f7213b6775
- SCALA-21 - Update GridFS to use DBObject views. Holding back full bugfix until we have a 2.5 build to link against
- Example adjustments to filter by start time and namespace
- SCALA-10 - And this is why we unit test. Size was returning empty for cursor based results as it wasn't pulling the right value. Fixed, calling cursor.size.
- Added an alternative object construction method for MongoDBObject with a list of pairs, rather than varargs [philwills]
- Making scaladoc for MongoURI more explicit. Note that the wiki markup for lists isn't actually implemented in scaladoc yet. [philwills]
- Refactor Collection and Cursors using Abstract types, explicit 'DBObject' version is always returned from DB, Collection etc now. Those wanting to use typed versions must code the flip around by hand. !!! BREAKING CHANGE, SEE CODE / EXAMPLES

- SCALA-10 Updated MapReduce interfaces to finish 1.8 compatibility
  - Renamed MapReduceError to MapReduceException; MapReduceError is a non exception which represents a failed job
  - Changed MapReduceResult to automatically proxy 'results' in inline result sets
- Added missing methods to GridFSDBFile necessary to access the underlying datastream
- Fixed setter/getter of option on cursor
- For several reasons changed backing trait of DBList PML from Buffer to LinearSeq
- Moved to new MapReduce functionality based on MongoDB 1.7.4+ !!! You must now specify an output mode.
  - See http://blog.evilmonkeylabs.com/2011/01/27/MongoDB-1_8-MapReduce/
- MapReduce failures shouldn't throw Error which can crash the runtime
- New MapReduceSpec updates to include tests against new MongoDB MapReduce logic

3.4.15 Changes in Version 2.0.2

- Fixed the MongoDBOBject 'as' operator to return the proper type, instead of Any. (philwills)

3.4.16 Changes in Version 2.0.1

- SCALA-16: Added a few additional validation tests against getAs and as on MongoDBObject
- SCALA-17 - Fixed syntax of $within and its nested operators, unit test passes

3.4.17 Version 2.0 / 2011-01-03

Notable Changes since Casbah 1.0.8.1:

- Ownership Change: Casbah is now an officially supported MongoDB Driver
  - All bugs should be reported at http://jira.mongodb.org/browse/SCALA
  - Package Change: Casbah is now `com.mongodb.casbah` (See migration guide)
  - Documentation (ScalaDocs, Migration Guide & Tutorial) is available at http://mongodb.github.com/casbah
- Casbah is now broken into several submodules - see http://mongodb.github.com/casbah/migrating.html
- Casbah releases are now published to http://scala-tools.org
- SBT Build now publishes -sources and -javadoc artifacts
- Added heavy test coverage
- ++ additivity operator on MongoDBObject for lists of tuple pairs
- Updates to Java Driver wrappings
  - Casbah now wraps Java Driver 2.4 and fully supports all options & interfaces including Replica Set and Write Concern support
  - added a WriteConcern helper object for Scala users w/ named & default args
  - added findAndModify / findAndRemove
- Stripped out support for implicit Product/Tuple conversions as they're buggy and constantly interfere with other code.

- Migrated Conversions code from core to commons, repackaging as com.mongodb.casbah.commons.conversions
  - Moved loading of ConversionHelpers from Connection creation to instantiation of Commons' Implicits (This means conversions are ALWAYS loaded now for everyone)
- Switched off of configgy to slf4j as akka did
  - Added SLF4J-JCL Bindings as a +test* dependency (so we can print logging while testing without forcing you to use an slf4j implementation yourself)
  - Moved Logger from core to commons
- Massive improvements to Query DSL:
  - Added new implementations of $in, $nin, $all and $mod with tests. $mod now accepts non-Int numerics and aof two differing types.
  - Full test coverage on DSL (and heavy coverage on other modules)
  - Migrated $each to a now functioning internal hook on $addToSet only exposed in certain circumstances
  - Various cleanups to Type constraints in Query DSL
  - Full support for all documented MongoDB query operators
  - Added new $not syntax, along with identical support for nested queries in $pull
  - Valid Date and Numeric Type boundaries introduced and used instead of Numeric (since Char doesn't actually workwith Mongo and you can't double up type bounds)
  - Added full support for geospatial query.
  - Resolved an issue where the $or wasn't being broken into individual documents as expected.
  - DSL Operators now return DBObjects rather than Product/Tuple (massive fixes to compatibility and performance result)
  - Added @see linkage to each core operator's doc page
- GridFS Changes:
  - GridFS' 'files' now returned a MongoCursor not a raw Java DBCursor
  - GridFS findOne now returns an Option[_] and detects nulls like Collection
- Added "safely" resource loaning methods on Collection & DB
  - Given an operation, uses write concern / durability on a single connection and throws an exception if anything goes wrong.
- Culled casbah-mapper. Mapper now lives as an independent project at http://github.com/maxaf/casbah-mapper
- Bumped version of scala-time to the 0.2 release
- Added DBList support via MongoDBList, following 2.8 collections
- Adjusted boundaries on getAs and expand; the view-permitting Any was causing ambiguity issues at runtime with non AnyRefs (e.g. AnyVal).
- Fixed an assumption in expand which could cause runtime failure
- Updated MongoDBObject factory & builder to explicitly return a type; some pieces were assuming at runtime that it was a MongoDBObjectBuilder$anon1 which was FUBAR

3.4.18 Changes in Version 1.0.7.4

- Fixed some issues w/ GridFS libraries attempting to call toMap in iteration, which isn't implemented on the Java side; added custom toString methods on the GridFS files [BWM]
- Cleaned up log spam [BWM / MA]
- Added serialization hook for MongoDBObject to help catch any nested instances [MA]
- Cleaned up some stray references to java.lang.Object, replaced with AnyRef for good Scala coding practices [BWM]

3.4.19 Changes in Version 1.0.7

- Updated reference to Configgy to have a Scala version attached; this was causing issues on some mixed-version users' systems.
- Corrected massive stupidity from lack of testing on my part and disabled ScalaJDeserializers - in most cases these caused runtime ClassCastExceptions. *SERIALIZERS* still in place - Deserializers were just plain a bad idea.

3.4.20 Changes in Version 1.0.5

- Due to oddities and ambiguities, stripped the type parameter apply[A] method from MongoDBObject. If you want a cast return, please use MongoDBObject.getAs[A]. This should minimize odd runtime failures.
- Added toplevel detection in MongoDBObject's +=/put methods to try and convert a MongoDBObject value to DBObject for you.
- Added "Product" arguments to $pushAll - this means you can pass a Tuple-style list, where previously it required an Iterable ($pushAll ("foo" -> (5, 10, 23, "spam", eggs") should now work).
- Updated to scalaj-collection 1.0 release, built against 2.8.0 final
- Added a new ScalaJ-Collection based Deserializer and Serializer layer. All base types supported by ScalaJ collection now use asJava / asScala to cleanly ser/deser where possible. This excludes Comparator/Comparable and Map types for sanity reasons. See com.novus.casbah.mongodb.conversions.scala.ScalaConversions for detail. Please report bugs if this breaks your code - it's nascent and a bit naive!
- New Committer - Max Afonov
- Removed the BitBucket Mirror; we're purely on GitHub now. Bug tracker linked from Github page.
- Created a user mailing list - http://groups.google.com/group/mongodb-casbah-users

3.4.21 Changes in Version 1.0.2

- Changed $in, $notin, $all to always generate an array in Any* mode
- Added default type alias import for com.mongodb.DBRef & Casbah's MongoDB class

3.4.22 Changes in Version 1.0.1

- Updated externals to link against 2.8.0 final - 1.0 release had some RC/Beta built externals. (scalaj-collection is still linked against Beta)
- Added an Object interface, MongoDBAddress, for static construction of DBAddress instances.
- Added type aliases in MongoTypeImports for all Casbah companion objects - please report any odd behavior this causes.

- Added MapReduceCommand to BaseImports

3.4.23 Version 1.0

- GridFS enhanced via Loan Pattern
- Full support for MongoDB Query operators via fluid syntax (now with lots of testing to minimize breakage)
- Added support for Scala 2.8-style Map interaction w/ DBObject. Builder pattern, +=, etc.
- Tutorial Available

3.5 Whats new in Casbah

3.5.1 Casbah 2.7 and Mongo DB 2.6 Features

MongoDB 2.6 introduces some new powerful features that are reflected in the 2.7.0 driver release. These include:

- Aggregation cursors
- Per query timeouts **maxTimeMS**
- Ordered and Unordered bulk operations
- A parallelCollectionScan command for fast reading of an entire collection
- Integrated text search in the query language

Moreover the driver includes a whole slew of minor and major bug fixes and features. Some of the more noteworthy changes include.

- Added extra type checking so that `MongoDBObject.getAs[Type]("key")` better protects against invalid type casting so there are fewer scenarios where it will an invalid `Some(value)`.
- Extended helpers for `MongoDBObject.as[Type]("Keys"*)` and `MongoDBObject.getAs[Type]("keys" _*)` for easier fetching of nested MongoDBObjects.
- Fixed issue with OpLog matching - thanks to Brendan W. McAdams for the pull request.
- Register the core Serialization helpers only once - thanks to Tuomas Huhtanen for the pull request.
- Updated nscala-time to 0.6.0 and specs

Please see the *full changelog* and *upgrade documentation*.

Let's look at the main things in 2.6 features one by one.

3.5.2 Aggregation cursors

MongoDB 2.6 adds the ability to return a cursor from the aggregation framework. To do that simply use `AggregationOptions` with the aggregation command:

```
val collection = MongoClient()("test")("aggregate")

val aggregationOptions = AggregationOptions(AggregationOptions.CURSOR)
val results = collection.aggregate(
  List(
    MongoDBObject("$project" ->
      MongoDBObject("author" -> 1, "tags" -> 1)
    ),
    MongoDBObject("$unwind" -> "$tags"),
    MongoDBObject("$group" ->
      MongoDBObject("_id" -> "$tags",
```

```
                    "authors" -> MongoDBObject("$addToSet" -> "$author")
      )
    )
  ),
  aggregationOptions
);
```

Then the you can iterate the results of the aggregation as a normal cursor:

```
for (result <- results) println(result)
```

To learn more about aggregation see the aggregation tutorial and the aggregation reference documentation.

3.5.3 maxTime

One feature that has requested often is the ability to timeout individual queries. In MongoDB 2.6 it's finally arrived and is known as maxTimeMS. In Casbah support for `maxTimeMS` is via an argument or via the query api but is called `maxTime` and takes a Duration

Let's take a look at a simple usage of the property with a query:

```
val collection = MongoClient()("test")("maxTime")
val oneSecond = Duration(1, SECONDS)

collection.find().maxTime(oneSecond)
collection.count(maxTime = oneSecond)
```

In the examples above the **maxTimeMS** is set to one second and the query will be aborted after the full second is up.

3.5.4 Ordered/Unordered bulk operations

Under the covers MongoDB is moving away from the combination of a write operation + get last error (GLE) and towards a write commands api. These new commands allow for the execution of bulk insert/update/remove operations. The bulk api's are abstractions on top of this that server to make it easy to build bulk operations. Bulk operations come in two main flavors.

1. Ordered bulk operations. These operations execute all the operation in order and error out on the first write error.
2. Unordered bulk operations. These operations execute all the operations in parallel and aggregates up all the errors. Unordered bulk operations do not guarantee order of execution.

Let's look at two simple examples using ordered and unordered operations:

```
val collection = MongoClient()("test")("bulkOperation")
collection.drop()

// Ordered bulk operation
val builder = collection.initializeOrderedBulkOperation
builder.insert(MongoDBObject("_id" -> 1))
builder.insert(MongoDBObject("_id" -> 2))
builder.insert(MongoDBObject("_id" -> 3))

builder.find(MongoDBObject("_id" -> 1)).updateOne($set("x" -> 2))
builder.find(MongoDBObject("_id" -> 2)).removeOne()
builder.find(MongoDBObject("_id" -> 3)).replaceOne(MongoDBObject("_id" -> 3, "x" -> 4))

val result = builder.execute()

// Unordered bulk operation - no guarantee of order of operation
val builder = collection.initializeUnOrderedBulkOperation
```

```
builder.find(MongoDBObject("_id" -> 1)).removeOne()
builder.find(MongoDBObject("_id" -> 2)).removeOne()

val result2 = builder.execute()
```

For older servers than 2.6 the API will down convert the operations. However it's not possible to down convert 100% so there might be slight edge cases where it cannot correctly report the right numbers.

3.5.5 parallelScan

The **parallelCollectionScan** command is a special command targeted at reading out an entire collection using multiple cursors. Casbah adds support by adding the `MongoCollection.parallelScan(options)` method:

```
val collection = MongoClient()("test")("parallelScan")
collection.drop()

for(i <- 1 to 1000) collection += MongoDBObject("_id" -> i)

val cursors = collection.parallelScan(ParallelScanOptions(3, 200))

for (cursor <- cursors) {
  while (cursor.hasNext) {
    println(cursor.next())
  }
}
```

This optimizes the IO throughput from a collection.

3.5.6 Integrated text search in the query language

Text indexes are now integrated into the main query language and enabled by default:

```
val collection = MongoClient()("test")("textSearch")
collection.drop()
collection.ensureIndex( MongoDBObject("content" -> "text") )

collection += MongoDBObject("_id" -> 0, "content" -> "textual content")
collection += MongoDBObject("_id" -> 1, "content" -> "additional content")
collection += MongoDBObject("_id" -> 2, "content" -> "irrelevant content")

// Find using the text index
val result1 =  collection.find($text("textual content -irrelevant")).count

// Find using the $language operator
val result2 =  collection.find($text("textual content -irrelevant") $language "english").count

// Sort by score
val result3 = collection.findOne($text("textual content -irrelevant"), "score" $meta)
```

3.6 Upgrade

3.6.1 Version 2.7.0

- In order to meet Scala Style conventions (http://docs.scala-lang.org/style/naming-conventions.html#parentheses) the following 0 argument methods have had parentheses added if there are side effects or removed as there are no side effects:

  com.mongodb.casbah.MongoClient:

  - dbNames()
  - databaseNames()
  - getVersion
  - getConnectPoint
  - getAddress
  - getAllAddresses
  - getOptions

com.mongodb.casbah.MongoConnection:

  - dbNames()
  - databaseNames()
  - getVersion
  - getConnectPoint
  - getAddress
  - getAllAddresses
  - getOptions

com.mongodb.casbah.MongoDB:

  - collectionNames()
  - stats()
  - getName
  - getOptions

com.mongodb.casbah.MongoCollection:

  - getLastError()
  - lastError()

com.mongodb.casbah.MongoCursor:

  - count()

com.mongodb.casbah.map_reduce.MapReduceResult:

  - toString()

com.mongodb.casbah.util.MongoOpLog:

  - next()

com.mongodb.casbah.gridfs.GridFSDBFile:

  - toString()

- OpLog util fixes - Opid is now an option.
- Scala 2.9.1 and 2.9.2 are no longer supported
- Updated nscala-time to 0.6.0
- *com.mongodb.casbah.MongoCollection.findOne* signatures have to changed to a main defaulted method
- *com.mongodb.casbah.MongoCollection.mapReduce* may now throw an exception if maxDuration is exceeded
- *com.mongodb.casbah.MongoCollection.count* and *com.mongodb.casbah.MongoCollection.getCount* signatures have changed to add *Duration* but are defaulted.

- *com.mongodb.casbah.WriteConcern.valueOf* now returns an Option as an invalid name would return a *null*. Any code relying on it should be updated.
- *com.mongodb.casbah.commons.MongoDBList.getAs* now returns None if the type cast is invalid
- *com.mongodb.casbah.commons.MongoDBObject.getAs* now returns None if the type cast is invalid

3.6.2 Version 2.6.1

The com.mongodb.casbah.commons.test dependencies are now marked in the test classpath, so to install:

```
"org.mongodb" %% "casbah-commons" % "2.6.2" % "test"
```

3.6.3 Version 2.6.0

No upgrade needed.

3.6.4 Version 2.5.1

Scala 2.10

The *-Yeta-expand-keeps-star* compiler flag is no longer required.

3.6.5 Version 2.5.0

Scala 2.10

Because of how scala 2.10 handles repeated parameters you may need to build with the *-Yeta-expand-keeps-star* flag to upgrade your codebase.

CHAPTER

FOUR

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*