MongoClient or how to connect in a new and better way

From driver version **1.2** we introduced a new connection Class that has the same name across all of our official drivers. This is to ensure that we present a recognizable front for all our API's. This does not mean that your existing application will break, but rather that we encourage you to use the new connection api to simplify your application development.

Furthermore, the new connection class **MongoClient** acknowledges all writes to MongoDB, in contrast to the existing connection class Db that has acknowledgements turned off. Let's take a tour of the MongoClient functions.

```
MongoClient = function(server, options);

MongoClient.prototype.open

MongoClient.prototype.close

MongoClient.prototype.db

MongoClient.connect
```

Outlined above is the complete MongoClient interface. The methods **open**, **close** and **db** work very similar to the existing methods on the **Db** class. The main difference is that the constructor is missing the **database name** from Db. Let's show a simple connection using **open** as a code example speaks a thousand words.

```
var MongoClient = require('mongodb').MongoClient
  , Server = require('mongodb').Server;

var mongoClient = new MongoClient(new Server('localhost', 27017));
mongoClient.open(function(err, mongoClient) {
  var db1 = mongoClient.db("mydb");

  mongoClient.close();
});
```

Notice that you configure the MongoClient just as you would have done the Db object. The main difference is that you access the db instances using the **db** method on the MongoClient object instead of using the Db instance directly as you would previously. MongoClient supports the same options as the previous Db instance you would have created.

So, with a minimal change in our app, we can apply the new MongoClient connection code. But there is

more and one direction you might consider int the future. That is the mongodb connection string.

The URL connection format

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]
```

The URL format is unified across official drivers from 10gen with some options not supported on some drivers due to natural reasons. The ones not supported by the Node.js driver are left out for simplicities sake.

Basic parts of the url

- **mongodb://** is a required prefix to identify that this is a string in the standard connection format.
- **username:password@** is optional. If given, the driver will attempt to login to a database after connecting to a database server.
- **host1** is the only required part of the URI. It identifies either a hostname, IP address, or unix domain socket
- **:portX** is optional and defaults to :27017 if not provided.
- **/database** is the name of the database to login to and thus is only relevant if the username:password@ syntax is used. If not specified the “admin” database will be used by default.
- **?options** are connection options. Note that if database is absent there is still a / required between the last host and the ? introducing the options. Options are name=value pairs and the pairs are separated by “&”. For any unrecognized or unsupported option, a driver should log a warning and continue processing. A driver should not support any options that are not explicitly defined in this specification. This is in order to reduce the likelihood that different drivers will support overlapping that differ in small but incompatible ways (like different name, different values, or different default value).

Replica set configuration:

- **replicaSet=name**
    - The driver verifies that the name of the replica set it connects to matches this name. Implies that the hosts given are a seed list, and the driver will attempt to find all members of the set.
    - No default value.

Connection Configuration:

- **ssl=true|false|prefer**

    - true: the driver initiates each connections with SSL
    - false: the driver initiates each connection without SSL
    - prefer: the driver tries to initiate each connection with SSL, and falls back to without SSL if it fails.
    - Default value is false.
- **connectTimeoutMS=ms**
    - How long a connection can take to be opened before timing out.
    - Current driver behavior already differs on this, so the default must be left to each driver. For new implementations, the default should be to never timeout.
- **socketTimeoutMS=ms**
    - How long a send or receive on a socket can take before timing out.
    - Current driver behavior already differs on this, so the default must be left to each driver. For new implementations, the default should be to never timeout.

Connection pool configuration:

- **maxPoolSize=n:** The maximum number of connections in the connection pool
    - Default value is 5

Write concern configuration:

More detailed information about write concerns can be found at
http://www.mongodb.org/display/DOCS/getLastError+Command

- **w=wValue**
    - For numeric values above 1, the driver adds { w : wValue } to the getLastError command.
    - wValue is typically a number, but can be any string in order to allow for specifications like “majority”
    - Default value is 1.
    - wValue == -1 ignore network errors
    - wValue == 0 no write acknowledgement
    - wValue == 1 perform a write acknowledgement
    - wValue == 2 perform a write acknowledgement across primary and one secondary
    - wValue == ‘majority’ perform a write acknowledgement across the majority of servers in the replicaset
    - wValue == ‘tag name’ perform a write acknowledgement against the replicaset tag name
- **wtimeoutMS=ms**
    - The driver adds { wtimeout : ms } to the getlasterror command.
    - Used in combination with w
    - No default value

- **journal=true|false**
  - true: Sync to journal.
  - false: the driver does not add j to the getlasterror command
  - Default value is false
- **fsync=true|false**
  - true: Sync to disk.
  - false: the driver does not add fsync to the getlasterror command
  - Default value is false
  - If conflicting values for fireAndForget, and any write concern are passed the driver should raise an exception about the conflict.

Auth options

- **authSource=string:** Used when the user for authentication is stored in another database using indirect authentication.
  - Default value is null

Read Preference

- **slaveOk=true|false:** Whether a driver connected to a replica set will send reads to slaves/secondaries.
  - Default value is false
- **readPreference=enum:** The read preference for this connection. If set, it overrides any slaveOk value.
  - Enumerated values:
  - primary
  - primaryPreferred
  - secondary
  - secondaryPreferred
  - nearest
  - Default value is primary
- **readPreferenceTags=string.** A representation of a tag set as a comma-separated list of colon-separated key-value pairs, e.g. **dc:ny,rack:1**. Spaces should be stripped from beginning and end of all keys and values. To specify a list of tag sets, using multiple readPreferenceTags, e.g. **readPreferenceTags=dc:ny,rack:1&readPreferenceTags=dc:ny&readPreferenceTags=**
  - Note the empty value, it provides for fallback to any other secondary server if none is available
  - Order matters when using multiple readPreferenceTags
  - There is no default value

MongoClient.connect

The url format can be used with MongoClient.connect. Where possible MongoClient will pick the best possible default values for options but they can be overridden. This includes setting **auto_reconnect to true** and **native_parser to true if it's available**. Below are some example on how to connect to a single server a replicaset and a sharded system using **MongoClient.connect**

The single server connection

```
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect("mongodb://localhost:27017/integration_test", function(err
  test.equal(null, err);
  test.ok(db != null);

  db.collection("replicaset_mongo_client_collection").update({a:1}, {b:1}, {up
    test.equal(null, err);
    test.equal(1, result);

    db.close();
    test.done();
  });
});
```

A replicaset connect using no acknowledgment by default and readPreference for secondary

```
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect("mongodb://localhost:30000,localhost:30001/integration_tes
  test.equal(null, err);
  test.ok(db != null);

  db.collection("replicaset_mongo_client_collection").update({a:1}, {b:1}, {up
    test.equal(null, err);
    test.equal(1, result);

    db.close();
    test.done();
  });
});
```

A sharded connect using no acknowledgment by default and readPreference for secondary

```
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect("mongodb://localhost:50000,localhost:50001/integration_tes
  test.equal(null, err);
  test.ok(db != null);

  db.collection("replicaset_mongo_client_collection").update({a:1}, {b:1}, {up
    test.equal(null, err);
    test.equal(1, result);

    db.close();
    test.done();
  });
});
```

Notice that when connecting to the sharded system it's pretty much the same url as for connecting to the replicaset. This is because the driver itself figures out if it's a replicaset or a set of Mongos proxies it's connecting to. No special care is needed to specify if it's one or the other. This is in contrast to having to use the **ReplSet** or **Mongos** instances when using the **open** command.

MongoClient connection pooling

A Connection Pool is a cache of database connections maintained by the driver so that connections can be re-used when new connections to the database are required. To reduce the number of connection pools created by your application, we recommend calling **MongoClient.connect once** and reusing the database variable returned by the callback:

```
var express = require('express');
var mongodb = require('mongodb');
var app = express();

var MongoClient = require('mongodb').MongoClient;
var db;

// Initialize connection once
MongoClient.connect("mongodb://localhost:27017/integration_test", function(err
  if(err) throw err;

  db = database;

  // Start the application after the database connection is ready
  app.listen(3000);
  console.log("Listening on port 3000");
});

// Reuse database object in request handlers
app.get("/", function(req, res) {
  db.collection("replicaset_mongo_client_collection").find({}, function(err, d
```

```
    docs.each(function(err, doc) {
      if(doc) {
        console.log(doc);
      }
      else {
        res.end();
      }
    });
  });
});
```

MongoClient.connect options

The connect function also takes a hash of options divided into db/server/replset/mongos allowing you to tweak options not directly supported by the unified url string format. To use these options you do pass in a hash like this:

```
var MongoClient = require('mongodb').MongoClient;

MongoClient.connect("mongodb://localhost:27017/integration_test_?", {
    db: {
      native_parser: false
    },
    server: {
      socketOptions: {
        connectTimeoutMS: 500
      }
    },
    replSet: {},
    mongos: {}
  }, function(err, db) {
  test.equal(null, err);
  test.ok(db != null);

  db.collection("replicaset_mongo_client_collection").update({a:1}, {b:1}, {up
    test.equal(null, err);
    test.equal(1, result);

    db.close();
    test.done();
  });
});
```

Below are all the options supported for db/server/replset/mongos.

db: A hash of options at the db level overriding or adjusting

functionality not supported by the url

- **w**, {Number/String, > -1 ll 'majority'} the write concern for the operation where < 1 is no acknowledgment of write and w >= 1 or w = 'majority' acknowledges the write
- **wtimeout**, {Number, 0} set the timeout for waiting for write concern to finish (combines with w option)
- **fsync**, (Boolean, default:false) write waits for fsync before returning
- **journal**, (Boolean, default:false) write waits for journal sync before returning
- **readPreference** {String}, the preferred read preference (ReadPreference.PRIMARY, ReadPreference.PRIMARY_PREFERRED, ReadPreference.SECONDARY, ReadPreference.SECONDARY_PREFERRED, ReadPreference.NEAREST).
- **native_parser** {Boolean, default:false}, use c++ bson parser.
- **forceServerObjectId** {Boolean, default:false}, force server to create _id fields instead of client.
- **pkFactory** {Object}, object overriding the basic ObjectID primary key generation.
- **serializeFunctions** {Boolean, default:false}, serialize functions.
- **raw** {Boolean, default:false}, perform operations using raw bson buffers.
- **recordQueryStats** {Boolean, default:false}, record query statistics during execution.
- **retryMiliSeconds** {Number, default:5000}, number of milliseconds between retries.
- **numberOfRetries** {Number, default:5}, number of retries off connection.
- **bufferMaxEntries** {Number, default: -1}, sets a cap on how many operations the driver will buffer up before giving up on getting a working connection, default is -1 which is unlimited.

server: A hash of options at the server level not supported by the url.

- **readPreference** {String, default:null}, set's the read preference (ReadPreference.PRIMARY, ReadPreference.PRIMARY_PREFERRED, ReadPreference.SECONDARY, ReadPreference.SECONDARY_PREFERRED, ReadPreference.NEAREST)
- **ssl** {Boolean, default:false}, use ssl connection (needs to have a mongod server with ssl support)
- **slaveOk** {Boolean, default:false}, legacy option allowing reads from secondary, use **readPrefrence** instead.
- **poolSize** {Number, default:1}, number of connections in the connection pool, set to 1 as default for legacy reasons.
- **socketOptions** {Object, default:null}, an object containing socket options to use (noDelay:(boolean), keepAlive:(number), connectTimeoutMS:(number), socketTimeoutMS:(number))
- **logger** {Object, default:null}, an object representing a logger that you want to use, needs to support functions debug, log, error **({error:function(message, object) {}, log:function(message, object) {}, debug:function(message, object) {}})**.
- **auto_reconnect** {Boolean, default:false}, reconnect on error.
- **disableDriverBSONSizeCheck** {Boolean, default:false}, force the server to error if the BSON

message is to big

replSet: A hash of options at the replSet level not supported by the url.

- **ha** {Boolean, default:true}, turn on high availability.
- **haInterval** {Number, default:2000}, time between each replicaset status check.
- **reconnectWait** {Number, default:1000}, time to wait in milliseconds before attempting reconnect.
- **retries** {Number, default:30}, number of times to attempt a replicaset reconnect.
- **rs_name** {String}, the name of the replicaset to connect to.
- **socketOptions** {Object, default:null}, an object containing socket options to use (noDelay:(boolean), keepAlive:(number), connectTimeoutMS:(number), socketTimeoutMS:(number))
- **readPreference** {String}, the preferred read preference (ReadPreference.PRIMARY, ReadPreference.PRIMARY_PREFERRED, ReadPreference.SECONDARY, ReadPreference.SECONDARY_PREFERRED, ReadPreference.NEAREST).
- **strategy** {String, default:null}, selection strategy for reads choose between (ping and statistical, default is round-robin)
- **secondaryAcceptableLatencyMS** {Number, default:15}, sets the range of servers to pick when using NEAREST (lowest ping ms + the latency fence, ex: range of 1 to (1 + 15) ms)
- **connectArbiter** {Boolean, default:false}, sets if the driver should connect to arbiters or not.

mongos: A hash of options at the mongos level not supported by the url.

- **socketOptions** {Object, default:null}, an object containing socket options to use (noDelay:(boolean), keepAlive:(number), connectTimeoutMS:(number), socketTimeoutMS:(number))
- **ha** {Boolean, default:true}, turn on high availability, attempts to reconnect to down proxies
- **haInterval** {Number, default:2000}, time between each replicaset status check.