
GT.M Bindings to NodeJS

Release 1.0

Luis Ibanez
OSEHRA

December 31, 2012

Abstract

This paper describes `m.js`, an Open Source library to integrate the GT.M database with the NodeJS language. The purpose of this integration is to facilitate the use of the GT.M database functionalities from server-side Javascript programs developed with NodeJS.

The source code is publicly available under the Apache 2.0 License, and it has been tested in Ubuntu Linux.

<https://github.com/OSEHR/m.js>

The bindings are implemented in C++, and are based on the C-Language API provided by GT.M, and the C++ API provided by NodeJS, resulting in direct connection between GT.M and NodeJS in the same process.

Contents

1	Introduction	2
2	GT.M C-API	2
3	NodeJS C++ API	3
4	Error Management	3
5	Configuration	4
6	How to Build	4
6.1	Download from Github	4
6.2	Install Dependencies	4
6.3	Setting the Environment	5
6.4	Configure and Build	5
6.5	A Quick Test	5
7	Examples	6
7.1	Example 1	6
7.2	Example 2	6

1 Introduction

M is both a Language and a Database. In this article, we are focused on the M Database, and in particular, on making possible to use the M Database from scripts written in the NodeJS language.

This follows the pattern of usage of other popular NoSQL databases, such as MongoDB for example[2], that can be used from many different programming languages, event though there might be a preferred language to take full advantage of the database.

The integration is done here for the particular case of the GT.M implementation of the M Database.

2 GT.M C-API

The C-API of GT.M implemented by fisglobal is described in great detail in the following pages:

http://tinco.pair.com/bhaskar/gtm/doc/books/pg/UNIX_manual/ch11s05.html

The `m.js` library presented in this article is built upon the functionalities provided by this existing GT.M C-API. In particular, it takes advantage of the functions

- `gtm_init()`
- `gtm_ci()`
- `gtm_exit()`

and through them, it implements a C++ interface to the M functionalities of

- Get
- Set
- Lock
- Kill
- Order
- Query
- Execute

This API is implented in a C++ class named “GTM” in the Source directory.

The connections to GT.M are defined in a file named `gtm_access.ci` also to be found in the Source directory.

3 NodeJS C++ API

Here we used the V8 Engine implementation of NodeJS[3]. This engine is written in C++ and therefore it also offers a C++ API to interface to it.

The NodeJS addons API is described in great detail in the following web page:

<http://nodejs.org/api/addons.html>

In particular, we took here the approach of wrapping C++ objects, as described in the page below

http://nodejs.org/api/addons.html#addons_wrapping_c_objects

The wrapping of the C++ object was implemented in a class called “NodeGTM”, that can also be found in the Source directory.

This class implements the functions

- Version
- About
- New
- Get
- Set
- Kill
- Order
- Query
- Execute
- Lock

That are wrapping the corresponding calls in the “GTM” class described in the previous section.

The purpose of the NodeGTM class is to integrate the GTM class with the NodeJS V8 engine. Therefore, most of its code is only dealing with the reformatting of datastructures to be passed as arguments and to be passed back as return values.

4 Error Management

Given that the integration is done as a C++ library, error management is implemented by taking advantage of Exceptions.

A C++ Exception class was customized by deriving from the `std::exception` class, and adding elements relevant to the GTM-NodeJS integration. This class is called `GTMException` and can also be found in the Source directory.

5 Configuration

The configuration of the library is done with CMake[1].

In particular, this involved the creation of CMake descriptions on how to discover at configuration time the elements of the packages:

- GTM
- NodeJS
- V8

This descriptions were implemented in the corresponding files

- FindGTM.cmake
- FindNodeJS.cmake
- FindV8.cmake

These files are looking for the libraries and headers of these packages in a standardize manner.

The files can be found in the `CMake` subdirectory.

6 How to Build

This build instructions are focused on Ubuntu Linux, which is the platform in which the library was developed and where it has been tested.

6.1 Download from Github

First you should download the source code from github with the command

```
git clone git://github.com/OSEHR/m.js.git
```

6.2 Install Dependencies

To prepare your build environment in Linux you should install the following packages:

- libv8-dev
- nodejs-dev

as well as installing GTM.

6.3 Setting the Environment

The programs that use these bindings must be able to find at runtime the `gtm_access.ci` file.

To this end, GT.M uses the environment variable `GTMCi`, that must be set to the absolute path of the `gtm_access.ci` file.

For example as:

```
export GTMCi=/home/ibanez/bin/m.js/bin/gtm_access.ci
```

The other standard GT.M environment variables must be also set as usual. In particular

- `gtm_dist`
- `gtmgbldir`

6.4 Configure and Build

As with typical CMake configurations, one must start by creating a directory to host the binary build. This is referred to as the `BINARY` directory.

From that binary directory we can invoke `ccmake` and pass to it as argument the path to the directory where we put the source code of `m.js`. For example

```
ccmake /src/m.js
```

CMake should be able to find the V8 and NodeJS components, and might ask you for the location of `gtm_dist`.

You can provide this information, and then use the “c” key to configure and then the “g” key to generate Makefiles.

Once you return to the command line prompt, you can simply invoke “make” and the build should be complete in about 30 seconds.

6.5 A Quick Test

The build process should generate in the `BINARY` directory a `lib` subdirectory where you will find among other things, the files:

- `libnodegtm.so`
- `runNode.sh`

The first one is the shared library that contains the GT.M to NodeJS interface.

The second file is a helper file to setup environment, copies two example files and executes them by calling `node`.

7 Examples

This section presents two simple examples on how to interact with GT.M from Javascript using the NodeJS bindings. You will find these two examples in the `Testing` subdirectory of the `m.js` project.

7.1 Example 1

In this simple example, we create a connection to the database and then exercise the basic functionalities of setting and getting a Global value.

```

1 // load gtm module
2 var gtm = require('./gtm');
3
4 // create gtm database connection
5 var db = new gtm.Database();
6
7 console.log("\n");
8 console.log('Node.js Version: ' + process.version);
9
10 console.log('Version: ' + db.version());
11
12 console.log('About: ' + db.about());
13
14 // Set a global
15 db.set('^Person', "John Lennon");
16
17 // Get the global
18 var name = db.get('^Person');
19
20 console.log('^Person ' + name);
21
22 // Kill the global
23 db.kill('^Person');
24
25 db.execute('write $ZVERSION,!');
```

7.2 Example 2

In this example we exercise the GT.M functionalites of: set, get, order, query and kill.

```

1 // load gtm module
2 var gtm = require('./gtm');
3
4 // create gtm database connection
5 var db = new gtm.Database();
6
7 console.log("\n");
8 console.log('Node.js Version: ' + process.version);
9
10 console.log('Version: ' + db.version());
11
12 console.log('About: ' + db.about());
```

```

13
14 db.lock('^Fibonacci');
15
16 // Set a global
17 db.set('^Fibonacci(1)','1');
18 db.set('^Fibonacci(2)','1');
19 db.set('^Fibonacci(3)','2');
20 db.set('^Fibonacci(4)','3');
21 db.set('^Fibonacci(5)','5');
22 db.set('^Fibonacci(6)','8');
23
24 // Get the index that follows 4, it should be 5.
25 var nextindex = db.order('^Fibonacci(4)');
26
27 // Set a global
28 db.set('^Person("name","address","phone)","John Lennon");
29
30 // Get the index that follows "address", it should be "phone".
31 var nextentry = db.query('^Person("name","address)');
32
33 // Set a global
34 db.set('^Apoptosis','Life is short');
35
36 // Kill the global
37 db.kill('^Apoptosis');
38
39 // modify the database
40 db.set('^Capital','Paris');
41
42 // query the database
43 var capital = db.get('^Capital');
44 console.log( capital );

```

8 Conclusions

A similar interface for NodeJS exist for the Intersystems Cache implementation of M. This binding is provided by the “Globals” project

<http://globalsdb.org>

Ideally, the GT.M binding presented in this article should evolve to provide an API that is consistent with the one provided by Globals, and therefore enable developers to write Javascript programs that interface to M databases, and reuse such programs without modification with systems that are deployed using either Intersystems Cache or GT.M.

A couple of differences may result however from fundamental differences between Cache and GT.M, for example, the fact that GT.M relies on the users permissions of the underlying operating system, and therefore does not need to implement mechanism for login and connecting to the database itself.

In general, however, it should be possible to conciliate the large majority of the two APIs, even though this may require a certain number of development iterations.

References

- [1] CMake Multi-Platform Configuration. <http://www.cmake.org>, December 2012. 5
- [2] MongoDB Drivers. <http://www.mongodb.org/display/DOCS/Drivers>, December 2012. 1
- [3] V8 Javascript Engine. <https://developers.google.com/v8/intro>, December 2012. 3