Introduction to CUBRID Node.js driver – Part I

October 2012

The scope of this tutorial is to provide an introduction to the **CUBRID Node.js** driver – ***cubrid-node***. We will show you where to get and how to install the driver, how to setup a connection and how to develop some simple database-enabled JavaScript code.

Requirements:

* Windows Vista, Windows 7, Linux
* CUBRID 8.4.1.232 or later (<http://www.cubrid.org/>)
* Node.js (<http://nodejs.org/>)

# Node.js Overview

Node.js is a platform built on [Chrome's JavaScript runtime](http://code.google.com/p/v8/) for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. See <http://nodejs.org/> for more details.

|  |
| --- |
| **Node.js** is:   * Evented Server-side JavaScript * Good at handling lots of different kinds of I/O at the same time. * Achieves this by all making almost all network and file I/O non-blocking. |

# *cubrid-node* – a CUBRID Node.js driver

**CUBRID Node.js driver** is an open-source project with the goal of implementing a 100% native node.js driver for the CUBRID database engine ([www.cubrid.org](http://www.cubrid.org)). It features a complete callbacks model and also an events model.

The official name of the driver is **node-cubrid** and the home is located on **github**: <https://github.com/CUBRID/node-cubrid>.

Here is a summary of the main functions implemented in the node-cubrid driver:

|  |  |
| --- | --- |
| Function name | Scope |
| CUBRIDConnection(brokerServer, brokerPort, user, password, database, cacheTimeout) | Initializes the driver module |
| connect(callback) | Connect to database |
| getEngineVersion(callback) | Get database version |
| query(sql, callback) | Executes a query which returns records |
| fetch(queryHandle, callback) | Fetch more records from a query |
| closeQuery(queryHandle, callback) | Close a query |
| queryWithParams(sql, arrParamsValues, callback) | Executes a query which returns records, using parameters in the query SQL statement |
| batchExecuteNoQuery(sqlsArray, callback) | Execute queries in batch mode |
| batchExecute(sqlsArray, callback) | Execute a single SQL statement |
| batchExecuteWithParams(sqlsArray, arrParamsValues , callback) | Execute a single SQL statement, with paramaters |
| beginTransaction(callback) | Begin a transaction |
| setAutoCommitMode(autoCommitMode, callback) | Set the auto-commit session mode |
| rollback(callback) | Rollback a transaction |
| commit(callback) | Commit a transaction |
| getSchema(schemaType, callback) | Gets database schema information |
| close(callback) | Disconnect from the database |

Beside the functions listed above, there are other utility-related functions provided by the driver; for example:

* *src\utils\Helpers.js*. Provides various utilities, from code formatting to input parameters validation and logging.
* *src\utils\ActionQueue.js*. Provides support for executing functions in sequence and for a while-do loop.

These functions are intended mainly for private usage, in the driver code; however, feel free to use them if you need such functionality.

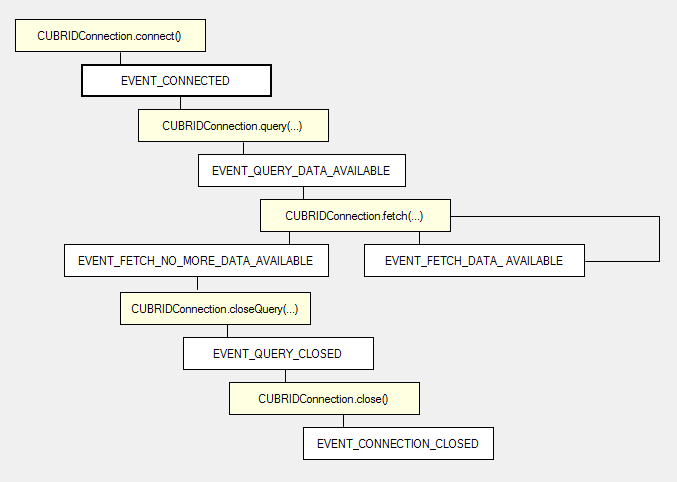
|  |
| --- |
| Remember: the driver is under continuous development and improvement, which means new functions are added, some functions are refactored, more and more examples and tests are available… Please check the [github](https://github.com/CUBRID/node-cubrid) project home page to stay up to date! |

# Driver Event model

Cubrid-node implements, in addition to the standard callbacks model functionality, a rich event model:

|  |  |
| --- | --- |
| Event name | Notes |
| EVENT\_ERROR | Raised when an error is encountered |
| EVENT\_CONNECTED | Raised when a connection was established |
| EVENT\_ENGINE\_VERSION\_AVAILABLE | Raised when the database version is returned to the client |
| EVENT\_BATCH\_COMMANDS\_COMPLETED | Raised when the commands batch execution is completed |
| EVENT\_QUERY\_DATA\_AVAILABLE | Raised when the data from the query is available to the client |
| EVENT\_SCHEMA\_DATA\_AVAILABLE | Raised when the database schema is available to the client |
| EVENT\_FETCH\_DATA\_ AVAILABLE | Raised when more query data is available to the client through successive fetch command(s) |
| EVENT\_FETCH\_NO\_MORE\_DATA\_AVAILABLE | Raised when no more data is available from the query |
| EVENT\_BEGIN\_TRANSACTION | Raised when a transaction was started |
| EVENT\_SET\_AUTOCOMMIT\_MODE\_COMPLETED | Raised when auto-commit mode was set |
| EVENT\_COMMIT\_COMPLETED | Raised when commit was completed |
| EVENT\_ROLLBACK\_COMPLETED | Raised when rollback was completed |
| EVENT\_QUERY\_CLOSED | Raised when the query was closed |
| EVENT\_CONNECTION\_CLOSED | Raised when the connection to the database was closed |

The logical chain between events, for a query, is described in the below diagram:



For more details about the events parameters, please consult the driver documentation, or look at the code examples and the tests included in the driver released code.

And, why not, look directly at the [driver code](https://github.com/CUBRID/node-cubrid/tree/master/src)…! ☺

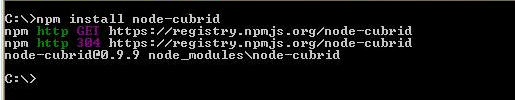
# Driver installation

Besides the obvious way of downloading the driver code, there is a much easier way to install the driver – by using the **npm** driver support.

[npm](https://npmjs.org/) is a [NodeJS](http://nodejs.org/) package manager. As its name implies, you can use it to install node programs/modules which are registered in the npm catalog, as it does the cubrid node.js driver.

So all you have to do to install node-cubrid is to execute the following command:

**>npm install node-cubrid**



**npm** automatically takes care of any dependencies and of downloading the latest available published driver code. And uninstalling the driver is all about executing a simple command:

**>npm uninstall node-cubrid**

# Connecting to CUBRID

Once everything is in place, let’s see how we write a simple program which does only one thing – it connects to a CUBRID database. For simplicity, let’s assume we want to connect to a local CUBRID installation, in particular to the well-known *demodb* database.

And this is how you do it, using the node-cubrid driver:

*var CUBRIDConnection = require('../src/CUBRIDConnection');*

*var CUBRIDClient = new CUBRIDConnection('localhost', 33000, 'public', '', 'demodb');*

*CUBRIDClient.connect(function (err) {*

*if (err) {*

*errorHandler(err);*

*} else {*

*CUBRIDClient.close(function (err) {*

*if (err) {*

*errorHandler(err);*

*}*

*})*

*}*

*});*

|  |
| --- |
| Please note the usage of the *src/CUBRIDConnection* JavaScript file, which contains the definition of the driver main “object”.  Also, please note how the database connect information are being initialized within the **new** call. |

If you prefer using **events**, this is how you can achieve the same results as above:

*var CUBRIDConnection = require('../src/CUBRIDConnection');*

*var CUBRIDClient = new CUBRIDConnection('localhost', 33000, 'public', '', 'demodb');*

*CUBRIDClient.connect();*

*CUBRIDClient.on(CUBRIDClient.EVENT\_ERROR, function (err) {*

*throw err.message;*

*});*

*CUBRIDClient.on(CUBRIDClient.EVENT\_CONNECTED, function () {*

*console.log('Connection opened.');*

*CUBRIDClient.query('select \* from game');*

*});*

*CUBRIDClient.on(CUBRIDClient.EVENT\_QUERY\_DATA\_AVAILABLE, function (result, queryHandle) {*

*CUBRIDClient.closeQuery(queryHandle, function () {*

*});*

*});*

*CUBRIDClient.on(CUBRIDClient.EVENT\_QUERY\_CLOSED, function (queryHandle) {*

*CUBRIDClient.close();*

*});*

*CUBRIDClient.on(CUBRIDClient.EVENT\_CONNECTION\_CLOSED, function () {*

*console.log('Connection closed');*

*});*

As you can see, it is very easy to work with the driver functions model – they follow the standard database logic model you know from other database drivers. The only particular thing, which is specific to node.js, is the callbacks/events model. If you need or want to read more about it, we recommend starting with the following online resources:

* <http://www.theprojectspot.com/tutorial-post/nodejs-for-beginners-callbacks/4>
* <http://wekeroad.com/2012/02/25/nodejs-callback-conventions-and-your-app>
* <http://nodemanual.org/latest/nodejs_dev_guide/working_with_callbacks.html>
* <http://nodejs.org/api/events.html>
* <http://howtonode.org/demystifying-events-in-node>

# Executing a non-query SQL command

In the next example, we will see how we can use the batchExecute function.

We will:

* Create a table
* Insert some data in the table
* Drop the table

For simplicity, the error-handling code was removed. Remember, it is a best practice in your live applications to always test for errors!

*var CUBRIDConnection = require('../src/CUBRIDConnection');*

*var CUBRIDClient = new CUBRIDConnection('localhost', 33000, 'public', '', 'demodb');*

*CUBRIDClient.connect(function () {*

*CUBRIDClient.batchExecuteNoQuery('drop table if exists node\_test', function () {*

*CUBRIDClient.batchExecuteNoQuery('create table node\_test(id int)', function () {*

*CUBRIDClient.batchExecuteNoQuery('insert into node\_test values(1)', function () {*

*CUBRIDClient.batchExecuteNoQuery('drop table node\_test', function () {*

*CUBRIDClient.close();*

*})*

*})*

*})*

*})*

*});*

# Executing a query

In order to execute a SQL query which returns data to the client, we need to use the standard pattern:

1. Send query
2. Receive:
   1. Total records count
   2. Records information: column names , column data types
   3. The first batch of records (the number of records return depends on the socket internal buffer size)
3. Send command to fetch more data
4. Receive fetched data
5. Repeat steps 3-4 while more data is available
6. Close the query request

Let’s see now some query examples.

## Simple query

*var CUBRIDConnection = require('../src/CUBRIDConnection'),*

*Helpers = require('../src/utils/Helpers'),*

*Result2Array = require('../src/resultset/Result2Array');*

*var CUBRIDClient = new CUBRIDConnection('localhost', 33000, 'public', '', 'demodb');*

*CUBRIDClient.connect(function () {*

*CUBRIDClient.query('select \* from code', function (err, result, queryHandle) {*

*var arr = Result2Array.RowsArray(result);*

*for (var j = 0; j < arr.length; j++) {*

*Helpers.logInfo(arr[j].toString());*

*}*

*CUBRIDClient.closeQuery(queryHandle, function () {*

*CUBRIDClient.close(function () {*

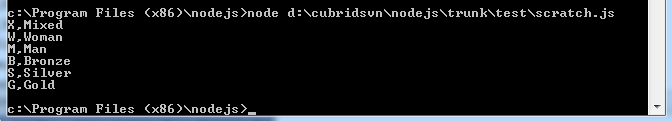
*})*

*})*

*})*

*});*

The result of the above code is:



|  |
| --- |
| Please note the usage of the *src/ Result2Array* JavaScript file, which contains a set of functions which converts the JSON data received by the driver into array data.  The following functions are available:   * **RowsArray**. Return the query records as a two-dimensional array[rows, columns] * **ColumnNamesArray**. Returns a single-dimensional array with the columns names returned by the query. * **ColumnTypesArray**. Returns a single-dimensional array with the columns data types returned by the query. * **TotalRowsCount**. Returns the total number of records returned by the query. Not the same as the current rows count in the data array – **TotalRowsCount** includes also the result of all possible successive fetch commands. |

## Fetching more results

We will now implement all the steps described at the beginning at this section, including consecutive fetch commands. And this time we will use driver events in our code:

*var CUBRIDConnection = require('../src/CUBRIDConnection'),*

*Result2Array = require('../src/resultset/Result2Array');*

*var CUBRIDClient = new CUBRIDConnection('localhost', 33000, 'public', '', 'demodb');*

*CUBRIDClient.connect(null);*

*CUBRIDClient.on(CUBRIDClient.EVENT\_ERROR, function (err) {*

*throw err.message;*

*});*

*CUBRIDClient.on(CUBRIDClient.EVENT\_CONNECTED, function () {*

*console.log('Connection opened.');*

*CUBRIDClient.query('select \* from participant, null);*

*});*

*CUBRIDClient.on(CUBRIDClient.EVENT\_QUERY\_DATA\_AVAILABLE, function (result, queryHandle) {*

*console.log('Total query result rows count: ' + Result2Array.TotalRowsCount(result));*

*console.log('First "batch" of data returned rows count: ' + Result2Array.RowsArray(result).length);*

*CUBRIDClient.fetch(queryHandle, null);*

*});*

*CUBRIDClient.on(CUBRIDClient.EVENT\_FETCH\_DATA\_AVAILABLE, function (result, queryHandle) {*

*console.log('Next fetch of data returned rows count: ' + Result2Array.RowsArray(result).length);*

*CUBRIDClient.fetch(queryHandle, null);*

*});*

*CUBRIDClient.on(CUBRIDClient.EVENT\_FETCH\_NO\_MORE\_DATA\_AVAILABLE, function (queryHandle) {*

*CUBRIDClient.closeQuery(queryHandle, null);*

*});*

*CUBRIDClient.on(CUBRIDClient.EVENT\_QUERY\_CLOSED, function () {*

*CUBRIDClient.close(null);*

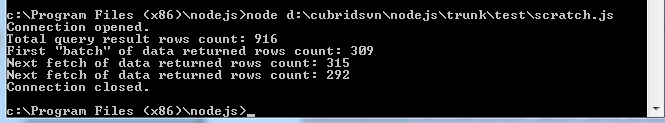
*});*

*CUBRIDClient.on(CUBRIDClient.EVENT\_CONNECTION\_CLOSED, function () {*

*console.log('Connection closed.');*

*});*

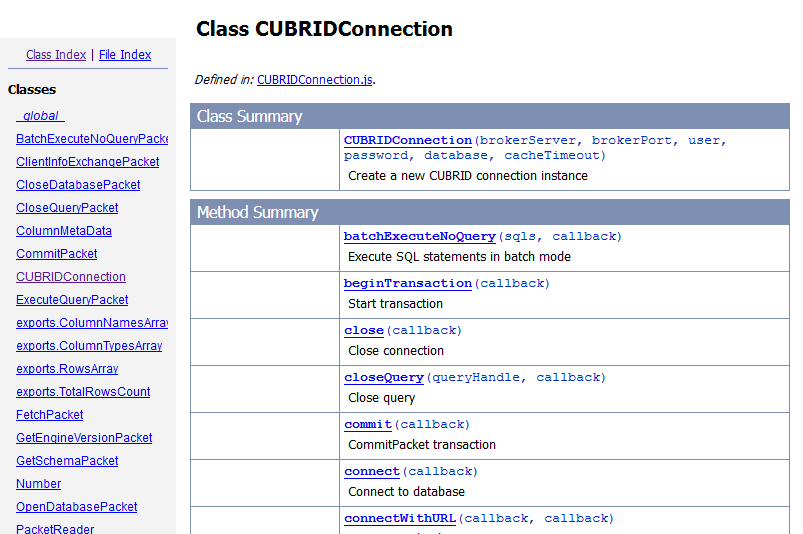
And the result is, as expected:



# Driver documentation

The driver source code is documented using the standard [jsdoc-toolkit](http://code.google.com/p/jsdoc-toolkit/) compatible documentation tags ([jsdoc-toolkit](http://code.google.com/p/jsdoc-toolkit/) is documentation generator for JavaScript code).

The generated documentation can be found in the code repository, under the *src\documentation* folder. Just open the index.html file in a browser and you will immediately have access t the driver documentation:



What we recommend first of all to any user who wants to get familiar with the driver is to take a look at the code examples included with the driver code.

There are:

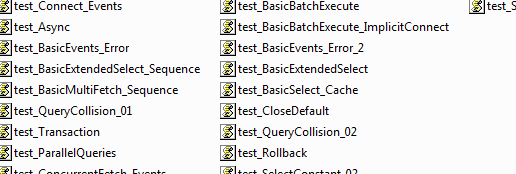
* Dozens of test cases showing the usage of all the driver functions
* End-to-End scenarios demos
* 5 web applications demos

Going through these many examples will put you on fast track to use the driver in your applications!

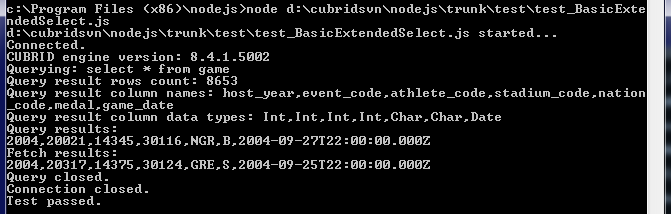
You can find all these in the following driver code folders:

**src\test**

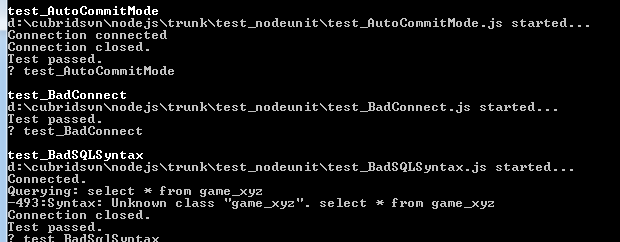
**src\demo**



Running the test cases can be done one-by-one – see below image:



On the other hand, if you install [node-unit](https://github.com/caolan/nodeunit), you can execute all of them at once (there is a *\test\_nodeunit* folder where all the test cases are enabled to be executed with **node-unit**:



# Links & Resources

|  |  |
| --- | --- |
| Driver home page | <https://github.com/CUBRID/node-cubrid>  <https://github.com/organizations/CUBRID> |
| Online driver documentation | <https://github.com/CUBRID/node-cubrid/tree/master/documentation> |
| Online Wiki | <http://www.cubrid.org/wiki_apis/entry/cubrid-node-js-driver> |
| Node.js resources | <http://howtonode.org/>  <http://nodejs.org/community/>  <http://en.wikipedia.org/wiki/Node.js> |
| CUBRID API Blog | <http://www.cubrid.org/blog/cubrid-appstools/> |

And this concludes the first **CUBRID Node.js** tutorial.

Please let us know your feedback – we highly appreciate your suggestions and comments - and remember to periodically check the CUBRID web site ([www.cubrid.org](http://www.cubrid.org)) for more CUBRID tutorials and resources.

**Thank you!**

The CUBRID APIs Team