Grant Agreement No.: 604656

Project acronym: NanoSim

Project title: A Multiscale Simulation-Based Design Platform for Cost-Effective CO_2 Capture Processes using Nano-Structured Materials (NanoSim)

Funding scheme: Collaborative Project

Thematic Priority: NMP

THEME: [NMP.2013.1.4-1] Development of an integrated multi-scale modelling environment for nanomaterials and systems by design

Starting date of project: 1st of January , 2014

Duration: 48 months

WP N°	Del. N°	Title	Contributors	Version	Lead beneficiary	Nature	Dissemin. level	Delivery date from Annex I	Actual delivery date dd/mm/yyyy
1	6	A MongoDB connectivity implementation in the Porto Scripting Environment	Thomas Hagelien	0.1	SINTEF	Report	PU	31/06/2014	31/06/2014

1 Executive Summary

MongoDB is an open-source database, designed for handling very large amounts of data. The key role of MongoDB in the NanoSim project is to act as our common central database, storing everything from simulation results to meta-data.

In NanoSim WP1 we've employed the MongoDB C-Driver and made a wrapper to Qt/C++. Based on this we've also created a wrapper to our scripting engine in the Porto Scripting Shell, and deployed the implementation as a plugin, using the Porto Plugin Interface.

2 MongoDB Driver

2.1 Qt/C++ Wrapper

The C++ wrapper technique is quite naïve and straight forward. Each *mongoc* structure type is represented with its own C++ class.

Example:

C API:	**Qt/C++ API:**
mongoc_client_t	class Client : public QObject
functions	**Methods**
mongoc_client_get_collection() mongoc_client_get_database() mongoc_client_get_gridfs() …	collection() database() gridFS()

The current implementation of the wrappers is available at GitHub:

https://github.com/NanoSim/Porto/tree/master/portostorage/src/mongo/driver

2.2 Script wrappers

In order to make the MongoDB driver available from script, we had to create an additional wrapper-layer and build a library that can be loaded runtime for the Porto Scripting Shell Environment (plugin).

```
#include <QObject>
#include "isoftplugin.h"

class QScriptEngine;
class MongoPlugin : public QObject
                  , public ISoftPlugin
{
   Q_OBJECT
   Q_PLUGIN_METADATA(IID "org.sintef.soft/ISoftPlugin/0.1")
   Q_INTERFACES(ISoftPlugin)

public:
   virtual ~MongoPlugin();
   virtual void registerPlugin(QScriptEngine *engine) override;

}; // class MongoPlugin
```

Since ECMAScript is a prototype-oriented language, the process of exposing a C++ class to ECMAScript is register the instance of an object, and register its interfaces (methods) as prototypes of that object. In our case this is trivial and the plugin library is a very thin wrapper layer. The current version (0.1) implementation is available on GitHub:

https://github.com/NanoSim/Porto/tree/master/portotools/src/plugins/mongo

2.3 Usage

Using the MongoDB classes are now straight forward.

```
// Inserts a JSON-document into the MongoDB database

var client = new MongoClient();
var collection = client.collection('dbname', 'collectionName');
collection.insert({somedoc:"value", foo: "bar"});
```