VELaSSCo

***V**isual Analysis for **E**xtremely **La**rge-**S**cale*
***S**cientific **Co**mputing*

D4.5 – Verification Tests of the Developed Tools

Version #1.0

Deliverable Information

Grant Agreement no	619439
Web Site	http://www.velassco.eu/
Related WP & Task:	WP4, T4.1, T4.2, T4.3
Due date	November 30, 2015
Dissemination Level	PU
Nature	R
Author/s	Andreas Dietrich, Miguel Pasenau
Contributors	Frank Michel, Abel Coll, Heidi Dahl

*The research leading to these results has received funding from the European Community's Seventh Framework Programme managed by REA-Research Executive Agency http://ec.europa.eu/research/rea [FP7/2007-2013] under grant agreement n° **619439***

Approvals

	Name	Institution	Date	OK
Author	Andreas Dietrich	**FRAUNHOFER**		
Task Leader	Andreas Dietrich	**FRAUNHOFER**		
WP Leader	Frank Michel	**FRAUNHOFER**		
Coordinator	Abel Coll	**CIMNE**		

Change Log

Version	Description of Change
Version 0.1	First draft of the document
Version 0.2	Minor formal corrections
Version 0.3	Added schemas about Test Client and Test Server scenarios, Query Manager command line interaction, with output logs, and a visual test with GiD.
Version 0.4	Minor formal corrections
Version 1.0	Final version

Table of Contents

1 Introduction

In this document, we provide an overview of the methods, procedures, and utilities, which are employed to perform verification testing of the systems and tools developed in WP4.

The testing infrastructure currently consists of three major components – query unit testing, query timing, and visual testing (M21). We will describe these in more detail in the following sections.

2 Query Unit Testing

Query unit tests represent the central part of the testing environment. For each implemented query, supplementary code exists that simulates the invocation of a query by a client. A list of queries and their respective function parameters can be found in D3.1.

In order to simulate both ends of a query, we have implemented a *test client* as well as a *test server*. While the test client simulates a sequence of queries that would be issued by a user through a suitable client application, the test server simulates the responses of VELaSSCo platform. The combination of these test units can be used for different testing scenarios, which will be described in following sections.

Also the Query Manager Module has a small command interpreter to check up on to the Data Layer's Storage Module. With this interpreter the status of the Storage Module and the underlying DB Engine can be tested, specific queries can be issued by hand and both the Engine Layer and the Data Layer can be stopped.

Figure 1 shows the different test scenarios used to validate the implementation at different levels, from the Visualization client through the Access Library and the Engine Layer, down to the Data Layer.

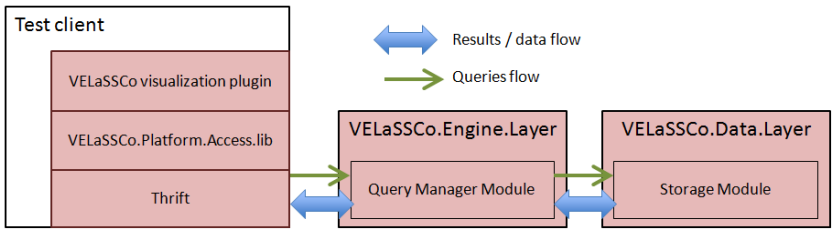


Figure 1: The *Test client* is used to verify the connection and implementation of the VQueries in both the Engine and Data Layer.

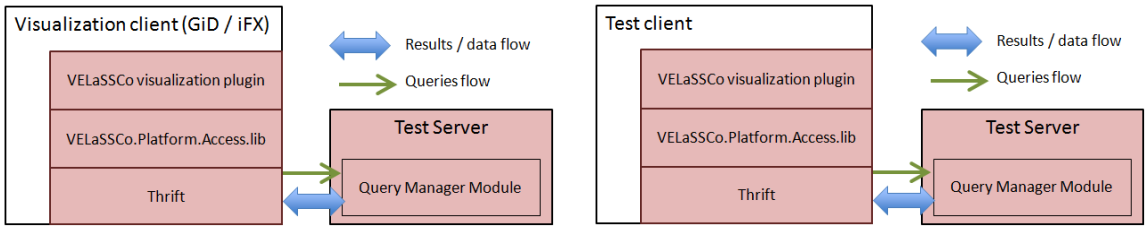


Figure 2: On the left the *Test server* is used to verify the connection from the visualization client and the access library and on the right, the Access Library itself and the connection between client and server can be verified.

2.1 Test Client

The test client executable is currently part of the Access Library module within the code base of the project. The test client acts as a standalone replacement for iFX or GiD (or any other visualization engine). Its executable is linked to the access library and can therefore connect to the VELaSSCo platform, issue queries and receive results. Like a regular client, the test client does not need to have any information about the inner workings of the platform as it only interfaces with the access library.

A session tested with the test client typically looks like:

1. Opening Session Queries, e.g.,
    a. UserLogin()
    b. GetStatusDB()
    c. GetListOfModels()
    d. OpenModel()
2. One or more Direct Result Queries, e.g.,
    a. GetListOfMeshes()
    b. GetResultsFromVerticesID()
    c. …
3. One or more Result Analysis Queries, e.g.,
    a. GetBoundingBox()
    b. …
4. Closing Session Queries, e.g.,
    a. CloseMode()
    b. UserLogout()

On each query, the test client executable checks the query result code and writes the outcome of the query to a log file. Currently, these logs are checked manually, but it would be easily possible to automate this with the help of suitable scripts. A more detailed view of the access library is presented in D2.4 and D4.2.

2.2 Test Server

The test server executable represents the counterpart of the test client. As it is used to simulate the behavior of the VELaSSCo platform, it implements the Thrift interface of the Query Manager module as a back-end. Although the test server implements all queries, it does not return any usable result data. For example, when issuing a GetBoundingBox() query, it returns the minimum/maximum values of a hardcoded bounding box.

2.3 Query Manager Interpreter

Embedded in the Query Manager Server, an interpreter has been implemented to test the health of the DB engine, type and send some Vqueries and stop the platform.

```
# launch the Query Manager
[VELaSSCo-EL] Connecting to Data Layer at pez001:36366
[VELaSSCo-EL] listening on port36367
List of available commands :
stop: stop the dataLayer application
ping: get Status of DB
query: does a query
exit (or quit): stop the current application (enginelayer)
 ################################################

[VELaSSCo-EL] Starting VELaSSCo Server...
[VELaSSCo-EL]   using port: 36367
[VELaSSCo-EL]   before serving ...

# getting status of the DB engine
ping
#### Ping ####
### 10 servers: 7 live and 3 dead.
        7  live  servers:  pez007:60020,  pez004:60020,  pez002:60020,  pez005:60020,
pez009:60020, pez006:60020, pez008:60020
        3 dead servers: node001, node002, pez001

#### /Ping ####
List of available commands :
stop: stop the dataLayer application
ping: get Status of DB
query: does a query
exit (or quit): stop the current application (enginelayer)
 ################################################

# issuing the GetResultFromVerticesID query
ping
#### Query ####
########## getResultFromVerticesID - 3000 0.039432 0.0165764 0.00161993  0
3001 0.042038 0.0173654 0.00714324  0
(…)
3999 0.0306545 0.047857 0.00930607  0
4000 0.037661 0.0156799 0.00821694  0

#### /Query ####
List of available commands :
stop: stop the dataLayer application
```

```
ping: get Status of DB
query: does a query
exit (or quit): stop the current application (enginelayer)
 ################################################

# stopping the platform
stop
#### stop ####
ERROR: No more data to read.
```

Figure 3: Output log of the VELaSSCo Query Manager server. In blue the command line interactions.

2.4 Access Library / API Testing

Using the combination of test server and client, the functionality of the access library and its API can be tested independently from any VELaSSCo platform back-end, i.e., without having to connect to a real query manager. This facilitates uncovering bugs that might be located within the access library itself, such as parameters of wrong type, incorrect encoding/decoding of query commands/results, etc.

An example output log for the test client and server executables can be shown in Figure 4 and Figure 5 respectively. It can be seen that every request on the client side corresponds to a response on the server side. In this example, the client sends a wrong query command with “CloseMode” as query name instead of “CloseModel”, which produces an error as result code.

```
# launch test client to connect to test server
Connecting to 'localhost:9990' ...

# user login
[VELaSSCo] UserLogin_Result:
[VELaSSCo]     result    : 0
[VELaSSCo]     sessionID : -845483053887632862

# get database status
[VELaSSCo] StatusDB_Result:
[VELaSSCo]         result : 0
[VELaSSCo]         status : Test_Server OK
status = Test_Server OK

# query model list
[VELaSSCo] Query_Result:
[VELaSSCo]     result : 0
in VELaSSCo_models:
   status = Ok
   model_list = NumberOfModels: 2
Name: DEM_examples/FluidizedBed_small
FullPath:
/localfs/home/velassco/common/simulation_files/DEM_examples/FluidizedBed_small
Name: fine_mesh-ascii_
FullPath: ../../../../../VELASSCO-Data/Telescope_128subdomains_ascii

# open model
[VELaSSCo] Query_Result:
```

```
[VELaSSCo]     result : 0
OpenModel:
   status = Ok
   model_modelID = 00112233445566778899aabbccddeeff

# get results from vertices
[VELaSSCo] Query_Result:
[VELaSSCo]     result : 0
Vertex: 0  ID: 6  Values: [ -0.105564 -0.287896 -0.377642 ]
Vertex: 1  ID: 7  Values: [ 0.259839 -0.366375 -0.377652 ]

# get bounding box
doing valGetBoundingBox
[VELaSSCo] Query_Result:
[VELaSSCo]     result : 0
GetBoundingBox: 00112233445566778899aabbccddeeff ( ascii)
         bbox = ( -0.5, -0.5, -0.5) - (0.5, 0.5, 0.5).

# close model
[VELaSSCo] Query_Result:
[VELaSSCo]     result : 4
VELaSSCo ERROR:
  Invalid/unknown query. ← Error
```

Figure 4: Output log of the VELaSSCo Access Library test client executable. The test client issues a number of queries to the test server in order to test the API. Note that the close command caused an error in this test.

```
# launch test server
[VELaSSCo] before StartServer
[VELaSSCo] Starting VELaSSCo Server...
[VELaSSCo]   using port: 9990
[VELaSSCo]   before serving ...

# user login
[VELaSSCo]
[VELaSSCo] ----- UserLogin() -----
[VELaSSCo]
[VELaSSCo] Input:
[VELaSSCo]   url      : localhost:9990
[VELaSSCo]   name     : andreas
[VELaSSCo]   password : 1234
[VELaSSCo]
[VELaSSCo] Output:
[VELaSSCo]   result    : 0
[VELaSSCo]   sessionID : -845483053887632862

# get database status
[VELaSSCo]
[VELaSSCo] ----- GetStatusDB() -----
[VELaSSCo]
[VELaSSCo] Input:
[VELaSSCo]   sessionID : -845483053887632862
[VELaSSCo]
[VELaSSCo] Output:
[VELaSSCo]   result : 0
[VELaSSCo]   status : Test_Server OK

# query model list
[VELaSSCo]
```

```
[VELaSSCo] ----- Query() -----
[VELaSSCo]
[VELaSSCo] Input:
[VELaSSCo]   sessionID : -845483053887632862
[VELaSSCo]   query     :
{
  "name"           : "GetListOfModels",
  "groupQualifier" : "",
  "namePattern"    : "*"
}
[VELaSSCo]
[VELaSSCo] Output:
[VELaSSCo]   result : 0
[VELaSSCo]   data   :
NumberOfModels: 2
Name: DEM_examples/FluidizedBed_small
FullPath:
/localfs/home/velassco/common/simulation_files/DEM_examples/FluidizedBed_small
Name: fine_mesh-ascii_
FullPath: ../../../../../VELASSCO-Data/Telescope_128subdomains_ascii

# open model
[VELaSSCo]
[VELaSSCo] ----- Query() -----
[VELaSSCo]
[VELaSSCo] Input:
[VELaSSCo]   sessionID : -845483053887632862
[VELaSSCo]   query     :
{
  "name"            : "OpenModel",
  "uniqueName"      :
"T_VELaSSCo_Models:/localfs/home/velassco/common/simulation_files/D2C/Data/Fluidized_
Bed_Small:FluidizedBed",
  "requestedAccess" : ""
}
[VELaSSCo]
[VELaSSCo] Output:
[VELaSSCo]   result : 0
[VELaSSCo]   data   :
00112233445566778899aabbccddeeff

# get results from vertices
[VELaSSCo]
[VELaSSCo] ----- Query() -----
[VELaSSCo]
[VELaSSCo] Input:
[VELaSSCo]   sessionID : -845483053887632862
[VELaSSCo]   query     :
{
  "name"       : "GetResultFromVerticesID",
  "modelID"    : "d94ca29be534ca1ed578e90123b7",
  "resultID"   : "MASS",
  "analysisID" : "DEM",
  "vertexIDs"  : [1,2,3,4,5,6,7],
  "timeStep"   : 10000
}
[VELaSSCo]
[VELaSSCo] Output:
[VELaSSCo]   result : 0
[VELaSSCo]   data   :
0000000000000000: 32 20 33 0a 06 00 00 00 00 00 00 00 07 00 00 00  2 3.............
0000000000000010: 00 00 00 00 b1 8b a2 07 3e 06 bb bf c1 8c 29 58  ........>.....)X
```

```
0000000000000020: e3 6c d2 bf 63 25 e6 59 49 2b d8 bf 09 6c ce c1  .l..c%.YI+...l..
0000000000000030: 33 a1 d0 3f a6 9b c4 20 b0 72 d7 bf 2a 37 51 4b  3..?... .r..*7QK
0000000000000040: 73 2b d8 bf                                      s+..

# get bounding box
[VELaSSCo]
[VELaSSCo] ----- Query() -----
[VELaSSCo]
[VELaSSCo] Input:
[VELaSSCo]   sessionID : -845483053887632862
[VELaSSCo]   query     :
{
  "name"         : "GetBoundingBox",
  "modelID"      : "00112233445566778899aabbccddeeff",
  "numVertexIDs" : "0",
  "lstVertexIDs" : [],
  "analysisID"   : "",
  "stepOptions"  : "ALL",
  "numSteps"     : "0",
  "lstSteps"     : []
}
[VELaSSCo]
[VELaSSCo] Output:
[VELaSSCo]   result : 0
[VELaSSCo]   data   :
0000000000000000: 00 00 00 00 00 00 e0 bf 00 00 00 00 00 00 e0 bf  ................
0000000000000010: 00 00 00 00 00 00 e0 bf 00 00 00 00 00 00 e0 3f  ...............?
0000000000000020: 00 00 00 00 00 00 e0 3f 00 00 00 00 00 00 e0 3f  .......?.......?

# close model
[VELaSSCo]
[VELaSSCo] ----- Query() -----
[VELaSSCo]
[VELaSSCo] Input:
[VELaSSCo]   sessionID : -845483053887632862
[VELaSSCo]   query     :
{
  "name"       : "CloseMode", ← Error: “CloseMode” instead of “ClosedModel”
  "modelID"    : "d94ca29be534ca1ed578e90123b7"
}
[VELaSSCo]
[VELaSSCo] Output:
[VELaSSCo]   result : 4
[VELaSSCo]   Invalid Query
```

Figure 5: Output log of the VELaSSCo Access Library server client executable. The test server answers the query requests from the test client using hardcoded results. Note the error caused by an invalid query command.

2.5 Query Manager (Platform) Testing

The test client also can be used to test the VELaSSCo back-end. Instead of connecting to the test server, the test client connects to the platform query manager and issues the same sequence of commands as shown in Section 2.1. The output log of the query manager is similar to the one produced by the test server application.

3 Query Timing

In order to measure the overall performance of the VELaSSCo system as perceived by the user, the access library supports tracing of API functions. To enable this feature an environment variable with name VAL_API_TRACE_FILE has to be defined. The value of this variable should provide the name of the trace file (See Figure 6).

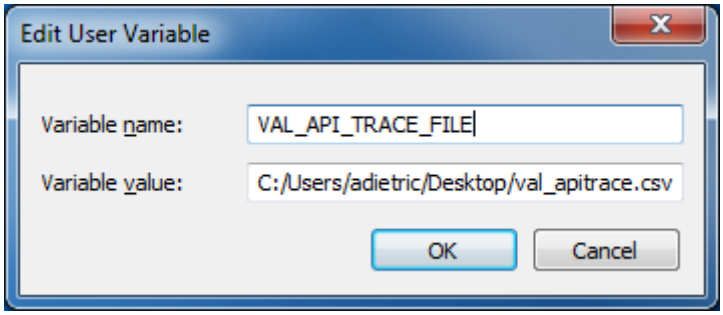


Figure 6: Defining the location of the access library API trace file using the VAL_API_TRACE_FILE environment variable.

When VAL_API_TRACE_FILE is defined, the access library will automatically create the trace file once it gets initialized. Every time an application invokes one of the access library functions through its API, the execution time of the function is measured and written to the trace file. Since the output is formatted as a comma-separated values (CSV) table, the trace file can be directly loaded into a spreadsheet application. An example trace resulting from the query sequence used in Figure 4 and Figure 5 is depicted in Figure 7.

```
FunctionName;                ExecutionTime[sec]
valUserLogin;                0.0088918
valGetStatusDB;              0.000815947
valGetListOfModels;          0.00163069
valOpenModel;                0.00190418
valGetResultFromVerticesID;  0.00267605
valGetBoundingBox;           0.00245599
valCloseModel;               0.00134874
valErrorMessage;             3.32054e-006
```

Figure 7: An example access library API trace. Each invocation of an API function is tracked and its execution time measured.

It has to be noted that the timings represent the complete execution time of an API function, i.e., they include not only the actual time it takes for a query to be executed by the storage back-end, but also any time needed to transfer and encode/decode data.

4 Visual Testing

Testing the operational functionality of VELaSSCo requires testing the behaviour of the VELaSSCo visualization plug-ins that connect the platform to the visualization engines (see Figure 8).

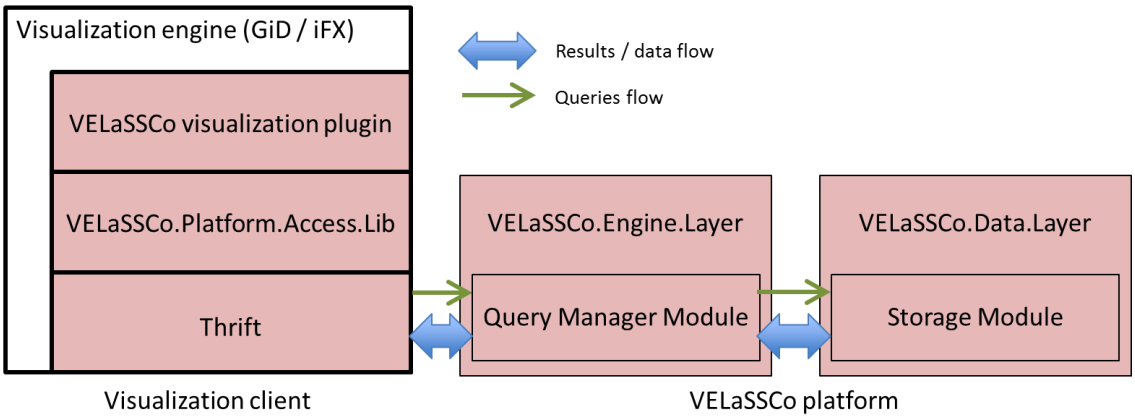


Figure 8: Query workflow triggered by the visualization engine.

This is usually done by visual inspection, i.e., by operating the visualization engine (see D4.2), and manually verifying the output of the VELaSSCo plug-in as shown with iFX in Figure 9.

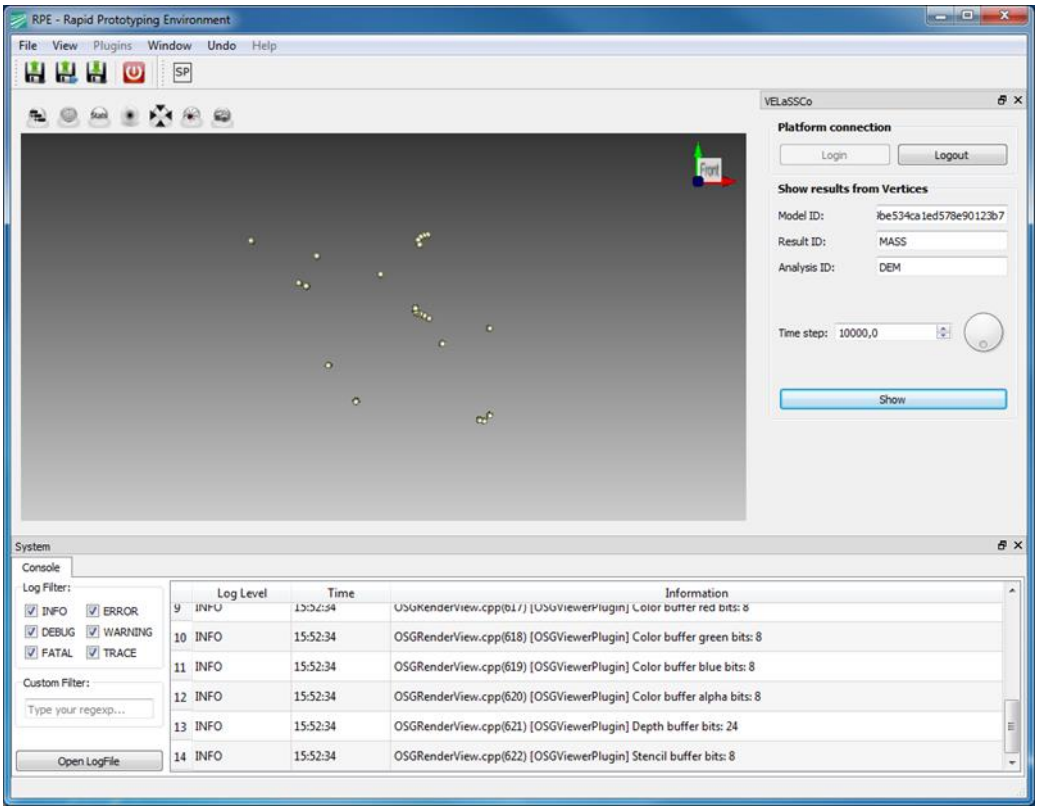


Figure 9: DEM particles resulting from a "GetResultsFromVerticesID" query shown in iFX.

Using the visualization engine will trigger a sequence of queries, which is similar to the one shown in Section 2.1. Typically, a user logs in, issues a number of visualization queries, and logs out. The correct reaction of the platform in response to the user input can be verified by checking the log files.

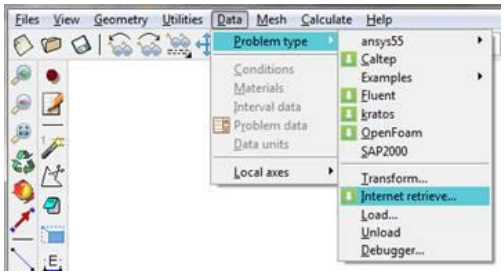
In order to make it easier to perform repeated tests of the iFX/GiD plug-in as the system grows, we employ a scripting tool named SikuliX [1]. SikuliX automates operating graphics user's interfaces. It can autonomously perform a sequence of actions such as selecting menu entries, pushing buttons, pulling sliders, etc. One of the advantages of SikuliX is its capability to recognize UI elements by itself. It is therefore not necessary to exactly record mouse positions. More importantly, when the user interface is undergoing (moderate) changes, the original scripts still continue to work.

The following steps describes an example of the Visual Testing of the VQuery GetBoundingBox with the visualization client GiD.

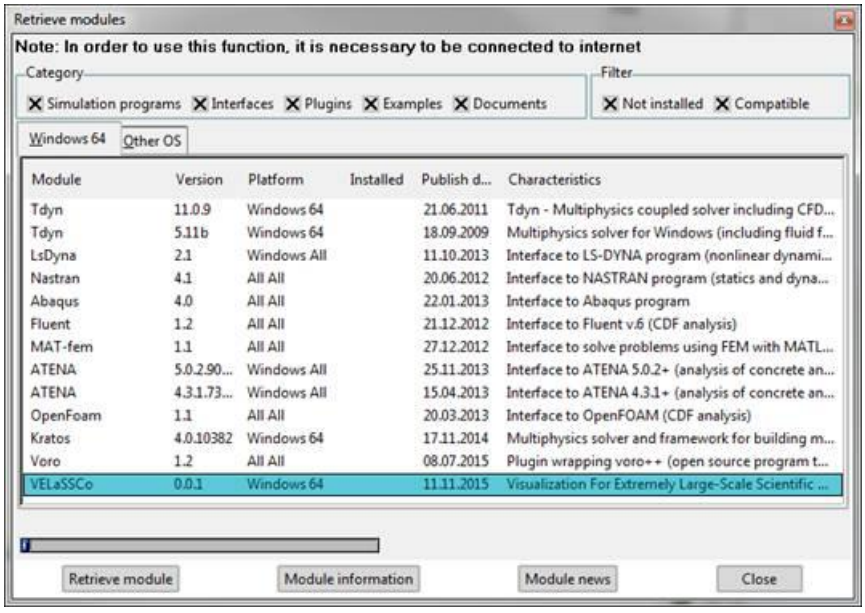
1. The visualization client GiD has to be downloaded from

http://www.gidhome.com/download/developer-versions

2. Once installed, to retrieve the VELaSSCo plug-in for GiD, go to Data-->Internet retrieve

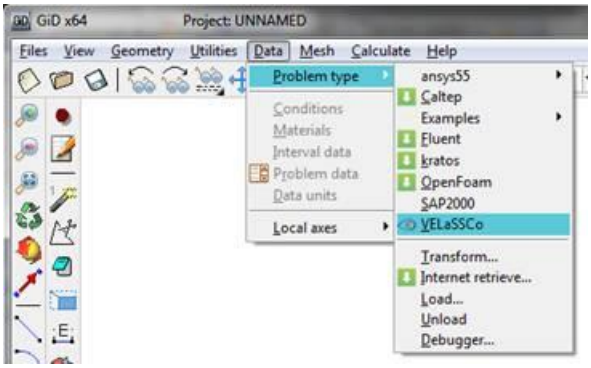


3. and select the "VELaSSCo" problem-type and press the "Retrieve module" button.



(eventually a window will pop up to ask for administrative permissions: they are needed to install the module into GiD's program folder)

4. Now you'll see a new entry in the Data top menu:



5. If you select this entry, GiD will change to post-process mode and new icon bar will appear:

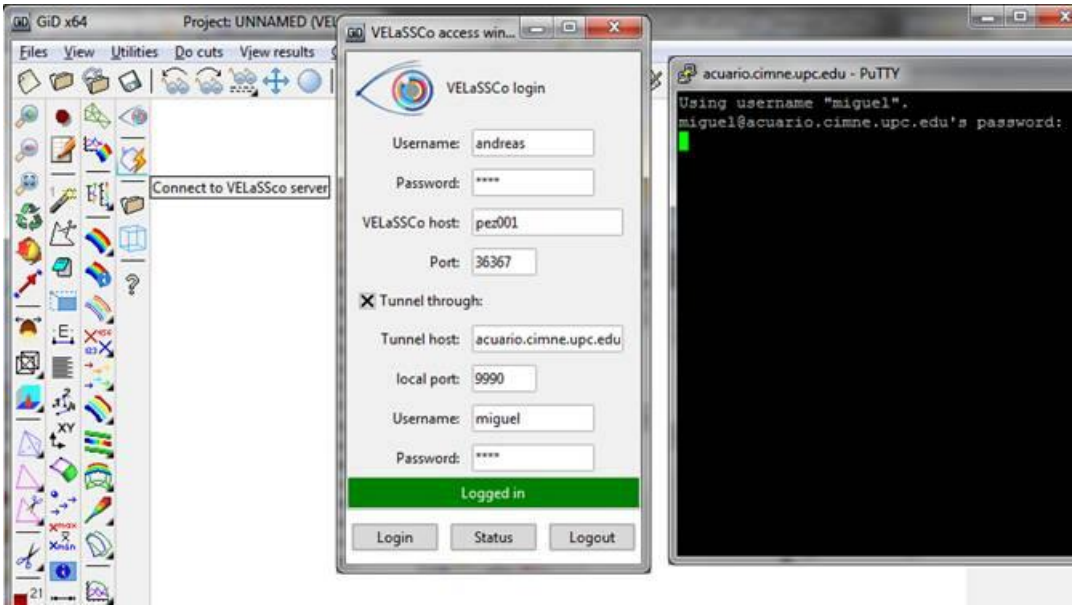
(if it appears as a standalone window, you can embed the bar with Utilities-->Tools-->Toolbars ... window)

6. Remember to start the QueryManager and the Storage module at pez001

7. Press the 'connect' icon to connect to the platform:

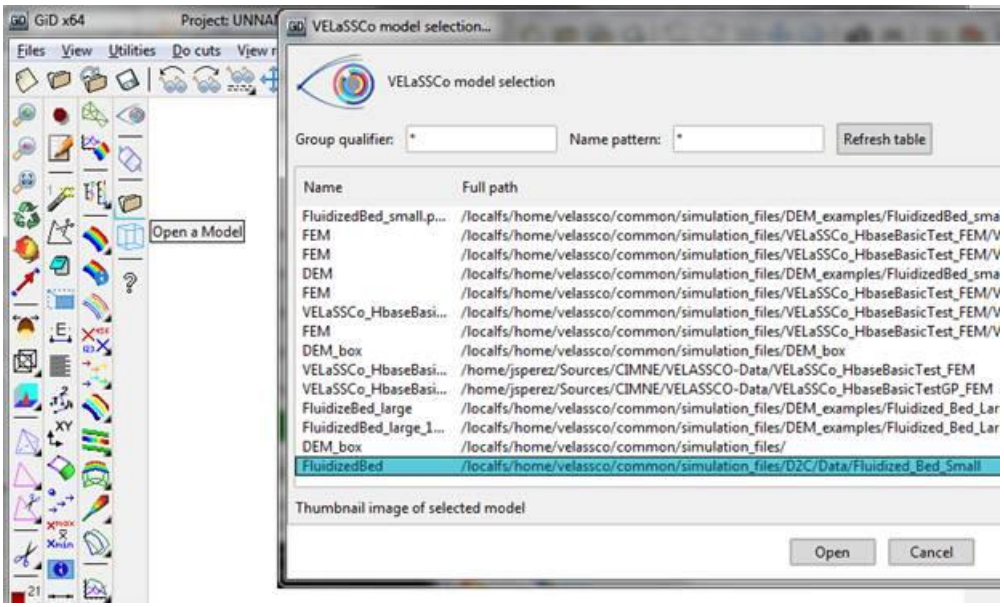
(when using the tunnel option, you'll have 5 seconds to enter your password to create the tunnel. If you don't have time, press 'login' again)

DON'T CLOSE THE CONNECTION WINDOW. Closing the connection window will automatically do a CloseModel + UserLogout + kill the ssh tunnel

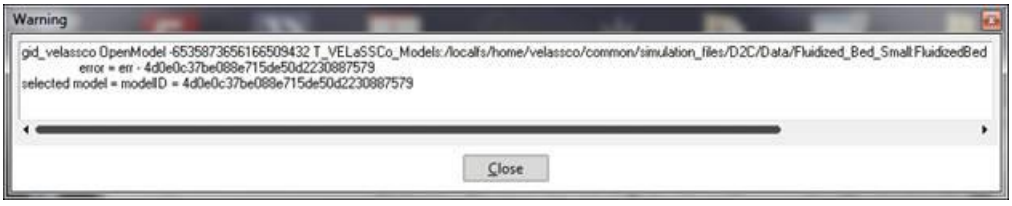


With the 'Status' button you can check if the Data server is alive

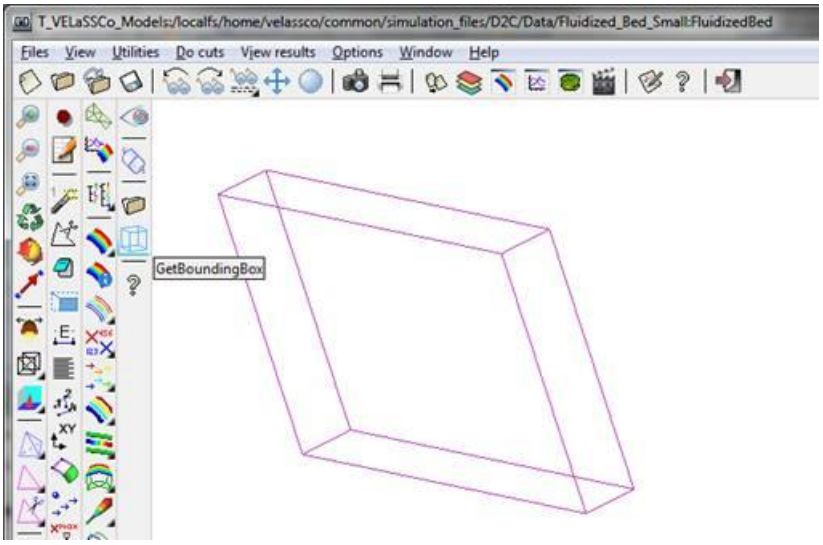
8. Now select a model:



A window with the modelId information will appear:



9. Now you can get the Bounding Box and visualize it:



The message window will show the bounding box:

5 References

[1] SikuliX, www.sikulix.com.