Teiid - Scalable Information Integration

1

Teiid Admin Shell User Guide

6.0.0 GA

Introduction Teiid Admin Shell

1.1. Introduction to MMAdmin

MMAdmin is a script based programing environment that enables user to access, monitor and control Teiid Server and MMQuery environments. This tool is built using programming language called BeanShell (*http://beanshell.org* [http://beanshell.org/]). MMAdmin can be used in ad-hoc scripting, or to run pre-defined scripts. It is not a graphical tool; it's a command line driven environment.

1. It is a fully functional programming environment with resource flow control and exception management. Under the covers it is a fully functional Java programming environment.
2. It is an Administrative tool. The user can connect to a running Teiid Server and invoke any of the Admin API methods, such as "addVDB" or "stopConnector", to control Teiid System just like using Console. Since, this is script driven, these tasks can be automated and re-run at a later time.
3. It is a data access tool. The user can connect to a VDB, issue any SQL commands, and view the results of the query.
4. It is a migration tool. This can be used to develop scripts like moving the Virtual Databases (VDB), Connector Bindings, and Configuration from one development environment to another. This will enable users to test and automate their migration scripts before production deployments.
5. It is a testing tool. The JUnit (*http://junit.org* [http://junit.org/]) test framework is built into it. User can write regression tests for checking system health, or data integrity that can be used to validate a system functionality automatically instead of manual verification by QA personnel.

1.1.1. Where can you find MMAdmin?

Where can you find MMAdmin

MMAdmin is bundled with the Teiid under “tools” package. Download and unzip this file to any directory using WinZip or tar command. Once you have unzipped the file, in root directory you will find “mmadmin” executable script to invoke the tool.

Windows: Double click or execute "mmadmin.cmd"

*nix: Execute the "mmadmin" script

to invoke MMAdmin. This will initialize the shell and present you with a prompt:

```
mmadmin $
```

Note: JDBC client API jar files are required on the same machine in order to successfully invoke MMAdmin. Usually these jar files are found in “Server” or “Console” installation directories under “client” directory.

Getting Started with MMAdmin

To learn the basics of BeanShell (*http://beanshell.org* [http://beanshell.org/]) take a look at their documents and tutorials on their website A copy of the BeanShell document is also can be found in the MMAdmin installation directory. Learn how basic scripts can be written and get familiar with some commonly used commands.

Note: BeanShell documentation provided is of version 1.3, which is taken as is from their website, for latest changes refer to BeanShell website. The BeanShell version 2.0b4 is currently being used in MMAdmin.

Basic knowledge of the Java programming language is required in order to effectively design and develop scripts using the MMAdmin. To learn Java language find learning resources at *http://java.sun.com* [http://java.sun.com/].

You can learn about the Teiid administrative API either using “help()” command or find JavaDoc for in the installation directory.

MMAdmin is a specialized version of BeanShell with lots of pre-built libraries and scripts designed to be used with the Teiid system. MMAdmin works in two different modes: interactive or script run mode.

In interactive mode, user can invoke the tool and connect to a live Teiid system and issue any ad-hoc commands to control the system or issue a SQL query against connected virtual database and view the results. All the commands executed during interactive mode are automatically captured into a log file, more on this later in the document.

In the script run mode, user can execute/play back previously developed scripts. This mode especially useful to automate any testing or to perform any repeated configurations/migrations changes to a Teiid system

2.1. Essential rules to follow in using MMAdmin

To use MMAdmin successfully, there are some syntactical rules you should keep in mind.

1. All the commands end with semi-colon [;]. Commands without any input parameters end with open and close parentheses '()' and semi-colon at the end. Example:

```
help();
```

2. All commands are case sensitive. So, enter exactly as command is written.

3. If a command requires input parameter(s), they should be declared inside "(" and ")" and if they are string based parameters then they should wrapped with double quotes. A command can have more than one parameter. Example:

```
cd("/home/johndoe");
```

4. Any other Java program can be invoked from script, if the required Java class libraries are already in class path. Import the required classes and instantiate the class to execute. Example:

```
import java.sql.*;
clazz = new MyClass();
clazz.doSomething();
```

5. Take a look at a variety of commands available in Bean Shell to be used along with your scripts. Some of the common ones you can find at *http://beanshell.org/manual/bshmanual.html#Useful_BeanShell_Commands* [http://beanshell.org/manual/bshmanual.html#Useful_BeanShell_Commands]

6. You can write and publish your own scripts. See How to write your own scripts?

To execute the commands in interactive mode you enter them first and press enter to execute, then enter next command, so on.

To exit the tool in the interactive mode, first disconnect if you are connected to the Teiid system by executing “disconnect();” then execute "exit();". In the script mode, when execution of the script finishes the tool will exit automatically, however you still have to disconnect from Teiid system in the script.

Note: If SSL is turned on the Teiid server, you would need to supply the correct certificates for connection. Edit the command file used to execute the MMAdmin and make sure correct trust store is defined in the path.

2.2. Basic commands to get started

The list below contains some common commands used in MMAdmin. The best way to learn scripting in the MMAdmin is to read the scripts in "samples" directory in the MMAdmin kit's installation directory, and experiment your own scripts using a developer instance of Teiid System.

```
```

```
print("xxx");   // print something to console

help();        // shows all the available admin commands;

connect();      // connect using connection.properties file

connect(<URL>);   // connect to Teiid using the supplied URL

connectAsAdmin(<url>); // connect as admin; no need have the vdb name. SQL commands will
 not work under this connection

execute(<SQL>); // run any SQL command. Note in interactive mode you can directly specify SQL
 on command line

currentConnectionName();  // returns the current connection name

useConnection(<connection name>); // switches to using the given connection settings

disconnect();   // disconnects the current connection in the context

exit();   // exit the shell
```

2.3. "help" command

This below command lists all the available administrative API commands in the MMAdmin. Please note that none of the BeanShell commands or custom commands will be shown in this list. Documentation is only source for reviewing those commands presently.

```
mmadmin $ help();
```

To get a specific definition about a command and it's required input parameters , use the below form of help. The example shown will show detailed JavaDoc description about "addVDB" method.

```
mmadmin $ help("addVDB");
/**
 * Import a {@link VDB} file.
 * <br>A VDB file with internal definitions. This is the default VDB export configuration
 * begining with MetaMatrix version 4.3.</br>
```

```
 *
 * @param name VDB Name
 * @param vdbFile byte array of the VDB Archive
 * @param option Code of the AdminOptions to use when executing this method. There are
 choices aboutwhat to do when a connector binding with the given identifier already exists in the
 system.
 */
VDB addVDB ( String name , String vdbFile , int option
```

If not sure about exact command, and to narrow the list available commands, help can be used in the following form, note the “*” at the end

```
mmadmin $   help(“get*”);
```

This will list all the commands available that begin with “get”, for example “getconnectorBindings, getVDBs” etc.

For every administrative API call, there is corresponding command in MMAdmin. For a reference to full administrative API, please look at “documents” sections in the installation directory.

Working with Scripts

3.1. How to write your own scripts

Open up a text editor, and enter all the commands in a editor and save the file ".bsh" extension and place it in the distribution where MMAdmin tool can access it. Take a look at the "samples" directory for example scripts. For example:

```
// foo.bsh
importCommands("commands");
load("server");

connect();
execute("select * from pm1.g1");
printResults();
disconnect();
exit();
```

and execute this script file using instructions in Executing script file in non-interactive mode section.

All script files that need to be executed in the non-interactive mode must have the following two lines at the beginning of the script file before execution. These two commands will load all the necessary libraries into the shell before execution.

```
importCommands("commands");
load("server");
```

3.2. Executing script file in interactive mode

To execute a script file "foo.bsh" in a directory "some/directory" in interactive mode, execute as following

```
mmadmin $ source ("some/directory/foo.bsh");
```

"foo.bsh" is read into current context of the shell as if you typed in the whole document. If your script only contained method calls, you can explicitly invoke the call to execute.

3.3. Executing script file in non-interactive mode

To execute a script file "foo.bsh" in a directory "some/directory" in non-interactive mode, execute as following command at the command prompt

```
mmadmin.sh some/directory/foo.bsh
```

Note that, in the script mode it is NOT possible to pass in the command line parameters as

```
mmadmin.sh some/directory/foo.bsh One Two
```

The parameters can be passed in as Java system properties. For example

```
mmadmin.sh some/directory/foo.bsh input1=One input2=Two
```

Inside the script file, you can access these properties using Java system property semantics

```
value = System.getProperty(“input1”); // will return "One"
```

3.4. Log File and Recorded Script file

During the interactive mode, all the commands executed by the user are recorded in "adminscript.txt" file. This file can be found in the root installation directory. This file later can serve as reference to commands that are executed or will aid in converting commands executed in the interactive mode into a standalone script, that can be run at a later time.

User can also capture the commands entered during a interactive session to their own script file by using “startRecording” and “stopRecording” commands. For example,

```
mmadmin $ startRecording (“directory/filename.bsh”);
mmadmin $ <commands..>
mmadmin $ stopRecording()
```

in this case all the commands executed after the “startRecording” and before the “stopRecording” are captured in the “directory/filename.bsh” file. This gives the user an option to capture only certain portions of the interactive session tat they are interested in and ignore the rest of it.

Also note that all the output during the interactive mode is sent to "mmadmin.log" file. In the script mode, no such files are created. In script mode, user can capture the standard out and redirect to a file if they need the log file.

Connection Management

4.1. Default Connection Properties

The file "connection.properties" in the installation directory of the MMAdmin defines the default connection properties with which user can connect to Teiid system. The following properties can be defined using this file

```
server.host = <server host name or ip address>
server.port = <server port number>
user.name = <user name>
user.password = <password>
vdb.name = <vdb name>
```

When properties file is found, a call to "connect()" without any input parameters, will connect to the Teiid system using the properties defined in properties file. However, user can always pass in parameters in the connect method to connect to a same or different server than one mentioned in the “connection.properties”. Look all the all the different connect methods using the “help()” method.

Note that it is not secure to leave the passwords in clear text, as defined above. Please take necessary measures to secure the properties file, or do not use this feature and always pass in password interactively or some other secure way.

Note: At any given time user can be actively connected to more than one system or have more than one connection to same system. To manage the connections correctly each connection is created given a unique connection name. To learn more about this look at Handling multiple Connections section.

4.2. Handling Multiple Connections

Using MMAdmin, user can actively manage more than one connection to a single or multiple Teiid systems. For example, two separate connections can be maintained, one to the development server and one to the integration server at the same time. This is possible because MMAdmin supports a feature called named connections.

Every time a connection is made, it assigns a unique name to that connection and manages the context of the connection under that name. Unless, user makes another connection all the commands executed after the connection is made will use that connection. If another connect command is executed then a new connection is made with a unique name and execution will be

switched to use the new connection that has been created. The previous connection will be held as it is in its current state, and will not be closed.

In the interactive mode the name of the connection can found in the command prompt. Also, you could use the following command in the interactive and script mode to find out the current connection's name

```
name = currentConnectionName();
```

Knowing the names of the connection that user is working with is important, as this would enable user to switch the active connection that they have currently working with another connection they have previously made. To switch the active connection, use the following command and supply the name of the connection to be used

```
useConnection(name);
```

If user supplies the same name as the active connection as they are currently participating in, then this operation will simply return with out any modifications. There is no limitation as to how many simultaneous connections that user can maintain, this is up to the script developer.

The following shows an example of using and switching between two connections.

```
// creates a connection
connect();

//capture the connection name
conn1 = currentConnectionName();

// run a SQL command using "conn1"
select * from table;

// creates a second connection
connect();

conn2 = currentConnectionName();
```

```
// runs a SQL command on the connection "conn2"
select * from table;

// switch the connection to "conn1"
useConnection(conn1);

// run the SQL command using connection "conn1"
select * from table;

// close the connection in the "conn1"
disconnect();

// switch to "conn2"
useConnection(conn2);
disconnect();

exit();
```

Writing Tests using MMAdmin

JUnit [http://junit.org] testing framework is integrated into the MMAdmin, so you can write regression tests and run tests in the JUnit style. To write a test,

1. start a method name which begins with "testxxxx()"
2. use method body to write actual test. Method body can contain any MMAdmin based script using Admin API and JDBC commands
3. finally use “assertions” to validate results.

For example:

```
testCountRows(){
 connect();
 execute("select * from table");

 // make sure you have 100 rows
 assertRowCount(100);

 disconnect();
}

testResults(){
 connect();
 execute("select * from table");

 // defines the all the values you are expecting
 String[] expected ={
  {"col1[string]", "col2[int]"},
  {"VALUE1", "100"}
  ..
 };

 // make sure your results match to expected
 assertResultSet(expected);
 disconnect();
}

// now actually run the above defined tests
runTests();
```

In the above example, “testCountRows”, made connection to a Teiid system and issued a query and made sure that certain table has required number of rows. Where as “testResults” asserted that results match to a certain set of pre-built results.

Note the “runTests()” call at the end of the script, as this is the call which triggers the execution of the tests defined in the script. There are various types asserts you can use in your scripts, for the all the available assert types please refer to *Assertion Library*

Appendix A. Frequently Asked Questions

A.1. Are there any pre-built scripts available?

Yes, there are a few utility scripts are available in the "samples" directory with ".bsh" extension.

A.2. I have written a very useful script to do XYZ, I would like this to be part of the distribution?

Yes, we would love to hear from users. Please submit the script to Technical Support, and if this script found to be useful and generic enough for all the customers, we will include the script in the scripts library in the following releases.

A.3. What is different between "connect(<url>)" method and "connectAsAdmin()"?

Connect connects to the Teiid system using a VDB name, so you should be able to run any SQL command which is based on that VDB along with Admin method calls. If you use "connectAsAdmin" then you can run all the admin calls but you can not run any SQL commands, and you do not need a VDB to connect to the Teiid system.

A.4. How can I turn on the debug?

At command prompt type

```
mmadmin $ debug= true;   // to turn it ON
mmadmin $ debug= false; // to turn it OFF
```

A.5. Is IDE support available for writing the scripts?

Full IDE support is not available, however eclipse can be used as the editor for developing the scripts. Just set the “.bsh” extension association with Java editor, so all the color syntax is available for your script.

A.6. Is debugging support available?

No. Not for the scripts. However, if the scripts are calling into other Java code, those calls can be intercepted and can be run through debugger.

A.7. How to turn OFF interactive mode? How can I capture results to a file?

Once you start the shell in the interactive mode, at the prompt enter the following

```
bsh.interactive=false;
```

this will turn off the interactive features like spitting the results to console. This is good way to capture the results of the executed query. Use *printResults(File)* method to capture the previously executed SQL command's results to the file handle supplied

Appendix B. Assert Commands Available

The following are the list of assert commands available in the MMAdmin to be used with the scripts.

B.1. Assertions for SQL results

```
/**
 * Assert that the result set has given number of rows.
 * @param count – Number of rows expected
 */
assertRowCount(int count);


/**
 * Assert that the result set contents match those of supplied rows
 * @param expectedRows – expected rows
 */
assertResultsSetEquals(String[] expectedRows);

/**
 * Assert that the result set contents match those of supplied rows in
 * a file
 * @param expectedRows – expected rows
 */
assertResultsSetEquals(File expected)
```

B.2. Assertions for checking any values

```
/**
 * assert that the given expression is true or false
 */
void assertTrue(boolean condition);
void assertTrue(String message, boolean condition);
void assertFalse(boolean condition);
void assertFalse(String message, boolean condition);
```

```
/**
 * Assert that two supplied values are equal
 */
void assertEquals(boolean expected, boolean actual);
void assertEquals(byte expected, byte actual);
void assertEquals(char expected, char actual);
void assertEquals(double expected,  double actual,double delta) ;
void assertEquals(float expected, float actual,float delta);
void assertEquals(int expected, int actual);
void assertEquals(Object expected, Object actual);
void assertEquals(long expected, long actual);
void assertEquals(short expected, short actual) ;
void assertEquals(String message,  boolean expected,  boolean actual);
void assertEquals(String message, byte expected,byte actual);
void assertEquals(String message, char expected,char actual);
void assertEquals(String message, double expected,double actual,double delta);
void assertEquals(String message, float expected,float actual,float delta);
void assertEquals(String message, int expected, int actual);
void assertEquals(String message, Object expected,Object actual);
void assertEquals(String expected, String actual);
void assertEquals(String message,String expected,String actual);
void assertEquals(String message,long expected,long actual)
void assertEquals(String message,short expected,short actual);

/**
 * Assert that the given object is either Null or Not Null
 */
void assertNotNull(Object object);
void assertNotNull(String message, Object object);
void assertNull(Object object) ;
void assertNull(String message, Object object);

/**
 * Assertion to check if two supplied objects same objects or not
 * the difference here is these use == vs equals in the equals methods.
 */
void assertNotSame(Object expected, Object actual) ;
void assertNotSame(String message,Object expected,Object actual);
void assertSame(Object expected, Object actual);
void assertSame(String message,Object expected,Object actual);
```

```
/**
 * Assertion to fail the program explicitly
 */
void fail();
void fail(String message);
```

Appendix C. JDBC Commands

The following are the list of commands available in the JDBC module, these will aid in executing SQL commands in the MMAdmin shell. These commands can be used in your program once after the initial connection is made

```
/**
 * Executes a SQL Command
 */
execute(String command);

/**
 * Executes a SQL Command using the prepared statement. Note that command can contain ?
 * for input parameter place holders and the object array can contain the corresponding values. If
 * command starts with “exec” then it will treat the command as the stored procedure call.
 */
execute(String command, Object[] parms);

/**
 * Executes a supplied commands in the string array using Batch mode
 * @param commands - array of commands
 */
execute(String[] commands);

/**
 * Prints the previously executed ResultSet contents to system out; usally in the interactive more
 the
 * resultset is automatically read, so this applies only in the script mode.
 */
printResults();

/**
 * Prints the previously executed ResultSet contents to a file.usally in the interactive more the
 * resultset is automatically read, so this applies only in the script mode.
 * @param File
 */
printResults(File file);

/**
 * Prints the previously executed ResultSet to system out, in the form required by the assert call.
 Usally in
 * the interactive more the resultset is automatically read, so this applies only in the script mode.
```

```
 * @param comparemode - true - print in the format that can be used in the regression tests
 *                    - false - print to console
 */
printResults(boolean true);

/**
 * Walks the ResultSet, but does not print any results. Good for performance testing. Usally in
 * the interactive more the resultset is automatically read, so this applies only in the script mode.
 */
walkResults()
```