JAMAL 2.3 Manual

Maksim Khadkevich

February 25, 2013

Contents

0.1 Introduction 3
0.2 Installation and usage 3
0.3 FAQ 4

0.1 Introduction

JAMAL (JAva MAtlab Linking) makes it possible to call Matlab functions from java, passing and returning java primitives and their arrays. JAMAL is based on the remote method invocation (RMI) technology and allows for calling Matlab function on the fly, without saving results to a temporary file.

0.2 Installation and usage

1. Either use precompiled *lib/jamal-2.3.jar* or compile source files and put them into JAR archive. Ant script build.xml is provided to facilitate the compilation process.

   !!!Important!!! If you decided to compile JAMAL, do not forget to invoke rmic compiler to generate MatlabCallerImpl_Stub.class, otherwise you will not be able to connect to MatlabServer.

2. Add jamal-2.3.jar file to the Matlab classpath: type in the matlab prompt

   ```
   edit classpath.txt
   ```

   and add the necessary line (for example: "c:/soft/jamal/jamal-2.3.jar").

3. Add in the classpath of your java program *jamal-2.3.jar*.

4. If you are planning to pass huge data arrays to Matlab, you should increase Java heap size (otherwise OutOfMemory exception would be thrown).

   Create a text file named java.opts in the $MATLABROOT/bin/$ARCH directory. $MATLABROOT is the MATLAB root directory and $ARCH is your system architecture, which you find by typing at the MATLAB Command Prompt:

   ```
   matlabroot computer('arch')
   ```

5. There are two possible ways to run MatlabServer: GUI mode and command-line mode.

   To launch MatlabServer in GUI mode type in the Matlab prompt

   ```
   »com.jamal.server.MatlabServer
   ```

   This starts server-side part of JAMAL. The following message should be normally displayed:

```
Jamal::MatlabServer is ready
```

MatlabServer can also be launched from a java program by invoking the following MatlabClient constructor: *MatlabClient(String host, String matlabExecutablePath)*. *matlabExecutablePath* should be the full path to the Matlab executable (e.g. "c:/MATLAB/R2009b/bin/matlab.exe").

6. Some examples of calling Matlab functions are given in the class *TestMatlabClient*. *MatlabClient.executeMatlabFunction(String matlabFunctionName, Object[ ] inputArgs, int numberOfOutputArgs)* passes matlab finction name, input arguments and number of output arguments. One has to know exactly how many output arguments are there in the matlab function.

7. In order to stop running MatlabServer one has to call *MatlabClient.shutDownServer()* method. In the current implementation of JAMAL it is **not** possible to do it from the inside of Matlab (for example, using CTRL+C).

   Another option is to call the main method of class *com.jamal.client.Jamal*. Calling

   ```
   java -jar jamal-2.3.jar -h <host>
   ```

   from command line sends a signal to stop MatlabServer running on host <host>.

   ```
   java -jar jamal-2.3.jar
   ```

   without program parameters shows GUI interface with the capability of shutting down running MatlabServer.

   More detailed instructions with screenshots can be found here.

0.3 FAQ

1. **Passing multidimensional java arrays.**

   Due to some differences between Java and Matlab in the representation of multidimensional arrays, one needs to transform input Java array, that is represented in Matlab as cells into a Matlab array. The example of computing the sum of a 2-D array is given in the class *com.jamal.TestMatlabClient*. Running this example requires *test2dArray.m* and *javaCellArgs2Matlab.m* files to be added to the Matlab path.

   Many thanks to Harry Rostovtsev for discovering this problem in the context of JAMAL and for providing a solution for the 2-Dimensional case.

More about linking Java and Matlab, conversion of primitive types and their arrays you can read Matlab tech docs.