# RCaller: A library for calling R from Java

by M.Hakan Satman

August 17, 2013

## Contents



**Abstract**

RCaller is an open-source, compact, and easy-to-use library for calling R from Java. It offers not only an elegant solution for the task but its simplicity is key for non-programmers or programmers who are not familier with the internal structure of R. Since R is not only a statistical software but an enormous collection of statistical functions, accessing its functions and packages is of tremendous value. In this short paper, we give a brief introduction on the most widely-used methods to call R from Java and highlight some properties of RCaller with short examples. User feedback has shown that RCaller is an important tool in many cases where performance is not a central concern.

## 1 Introduction

R [R Development Core Team(2011)] is an open source and freely distributed statistics software package for which hundreds of external packages are available. The core functionality of R is written mostly in C and wrapped by R functions which simplify parameter passing. Since R manages the exhaustive dynamic library loading tasks in a clever way, calling an external compiled function is easy as calling an R function in R. However, integration with *JVM* (Java Virtual Machine) languages is painful.

The R package *rJava* [Urbanek(2011a)] provides a useful mechanism for instantiating Java objects, accessing class elements and passing R objects to Java methods in R. This library is convenient for the R packages that rely on external functionality written in Java rather than C, C++ or Fortran.

The library *JRI*, which is now a part of the package *rJava*, uses *JNI* (Java Native Interface) to call R from Java [Urbanek(2009)]. Although *JNI* is the most common way of accessing native libraries in Java, *JRI* requires that several system and environment variables are correctly set before any run, which can be difficult for inexperienced users, especially those who are not computer scientists.

The package *Rserve* [Urbanek(2011b)] uses *TCP* sockets and acts as a *TCP* server. A client establishes a connection to Rserve, sends R commands, and receives the results. This way of calling R from the other platforms is more general because the handshaking and the protocol initializing is fully platform independent.

*Renjin* (`http://code.google.com/p/renjin`) is an other interesting project that addresses the problem. It solves the problem of calling R from Java by re-implementing the R interpreter in Java! With this definition, the project includes the tasks of writing the interpreter and implementing the internals. *Renjin* is intended to be 100% compatible with the original. However, it is under development and needs help. After all, an online demo is available which is updated simultaneously when the source code is updated.

Finally, RCaller [RCaller Development Team(2011)] is an LGPL'd library which is very easy to use. It does not do much but wraps the operations well. It requires no configuration beyond installing an R package (*Runiversal*) and locating the *Rscript* binary distributed with R. Altough it is known to be relatively inefficient compared to other options, its latest release features significant performance improvements.

# 2 Calling R Functions

Calling R code from other languages is not trivial. R includes a huge collection of math and statistics libraries with nearly 700 internal functions and hundreds of external packages. No comparable library exists in Java. Although libraries such as the Apache Commons Math [Commons Math Developers(2010)] do provide many classes for those calculations, its scope is quite limited compared to R. For example, it is not easy to find such a library that calculates quantiles and probabilities of non-central distributions. [Harner et al.(2009)Harner, Luo, and Tan] affirms that using R's functionality from Java prevents the user from writing duplicative codes in statistics softwares.

RCaller is an other open source library for performing R operations from within Java applications in a wrapped way. RCaller prepares R code using the user input. The user input is generally a Java array, a plain Java object or the R code itself. It then creates an external R process by running the *Rscript* executable. It passes the generated R code and receives the output as *XML* documents. While the process is alive, the output of the standard input and the standard error streams are handled by an event-driven mechanism. The returned *XML* document is then parsed and the returned R objects are extracted to Java arrays.

The short example given below creates two double vectors, passes them to R, and returns the residuals calculated from a linear regression estimation.

```
RCaller caller = new RCaller();
RCode code = new RCode();
double[] xvector = new double[]{1,3,5,3,2,4};
```

```
double[] yvector = new double[]{6,7,5,6,5,6};

caller.setRscriptExecutable("/usr/bin/Rscript");

code.addDoubleArray("X", xvector);
code.addDoubleArray("Y", yvector);
code.addRCode("ols <- lm ( Y ~ X )");

caller.setRCode(code);

caller.runAndReturnResult("ols");

double[] residuals =
   caller.getParser().
     getAsDoubleArray("residuals");
```

The *lm* function returns an R list with a class of *lm* whose elements are accessible with the *$* operator. The method *runAndReturnResult()* takes the name of an R list which contains the desired results. Finally, the method *getAsDoubleArray()* returns a double vector with values filled from the vector *residuals* of the list *ols*.

RCaller uses the R package *Runiversal* [Satman(2010)] to convert R lists to *XML* documents within the R process. This package includes the method *makexml()* which takes an R list as input and returns a string of *XML* document. Although some R functions return the results in other types and classes of data, those results can be returned to the *JVM* indirectly. Suppose that *obj* is an *S4* object with members *member1* and *member2*. These members are accessible with the *@* operator like *obj@member1* and *obj@member2*. These elements can be returned to Java by constructing a new list like *result¡-list(m1=obj@member1, m2=obj@member2)*.

# 3 Handling Plots

Although the graphics drivers and the internals are implemented in C, most of the graphics functions and packages are written in the R language and this makes the R unique with its graphics library. RCaller handles a plot with the function *startPlot()* and receives a *java.io.File* reference to the generated plot. The function *getPlot()* returns an instance of the *javax.swing.ImageIcon* class which contains the generated image in a fully isolated way. A Java example is shown below:

```
RCaller caller = new RCaller();
RCode code = new RCode();
File plotFile = null;
ImageIcon plotImage = null;

caller.
setRscriptExecutable("/usr/bin/Rscript");

code.R_require("lattice");

try{
 plotFile = code.startPlot();
 code.addRCode("
```

```
      xyplot(rnorm(100)~1:100, type='l')
      ");
}catch (IOException err){
 System.out.println("Can not create plot");
}

caller.setRCode(code);
caller.runOnly();

plotImage = code.getPlot(plotFile);
code.showPlot(plotFile);
```

The method *runOnly()* is quite different from the method *RunAndReturnResult()*. Because the user only wants a plot to be generated, there is nothing returned by R in the example above. Note that more than one plots can be generated in a single run.

Handling R plots with a *java.io.File* reference is also convenient in web projects. Generated content can be easly sent to clients using output streams opened from the file reference. However, RCaller uses the temp directory and does not delete the generated files automatically. This may be a cause of a *too many files* OS level error which can not be caught by a Java program. However, cleaning the generated output using a scheduled task solves this problem.

# 4 Live Connection

Each time the method *runAndReturnResult()* is called, an *Rscript* instance is created to perform the operations. This is the main source of the inefficiency of RCaller. A better approach in the cases that R commands are repeatedly called is to use the method *runAndReturnResultOnline()*. This method creates an *R* instance and keeps it running in the background. This approach avoids the time required to create an external process, initialize the interpreter, and load packages in subsequent calls.

The example given below returns the determinants of a given matrix and its inverse in sequence, that is, it uses a single external instance to perform more than one operation.

```
double[][] matrix =
    new double[][]{{5,4,5},{6,1,0},{9,-1,2}};
caller.setRExecutable("/usr/bin/R");
caller.setRCode(code);

code.clear();
code.addDoubleMatrix("x", matrix);
code.addRCode("result<-list(d=det(x))");
caller.runAndReturnResultOnline("result");

System.out.println(
"Determinant is " +
  caller.getParser().
   getAsDoubleArray("d")[0]
   );

code.addRCode("result<-list(t=det(solve(x)))");
caller.runAndReturnResultOnline("result");
```

```
System.out.println(
"Determinant of inverse is " +
  caller.getParser().
   getAsDoubleArray("t")[0]
   );
```

This use of RCaller is fast and convenient for repeated commands. Since R is not thread-safe, its functions can not be called by more than one threads. Therefore, each single thread must create its own *R* process to perform calculations simultaneously in Java.

# 5 Monitoring the Output

RCaller receives the desired content as *XML* documents. The content is a list of the variables of interest which are manually created by the user or returned automatically by a function. Apart from the generated content, R produces some output to the standard output (*stdout*) and the standard error (*stderr*) devices. RCaller offers two options to handle these outputs. The first one is to save them in a text file. The other is to redirect all of the content to the standard output device. The example given below shows a conditional redirection of the outputs generated by R.

```
if(console){
 caller.redirectROutputToConsole();
}else{
 caller.redirectROutputToFile(
     "output.txt" /* filename */,
     true  /* append? */);
}
```

# 6 Conclusion

In addition to being a statistical software, R is an extendable library with its internal functions and external packages. Since the R interpreter was written mostly in C, linking to custom C/C++ programs is relatively simple. Unfortunately, calling R functions from Java is not straightforward. The prominent methods use *JNI* and *TCP* sockets to solve this problem. In addition, *renjin* offers a different perspective to this issue. It is a re-implementation of R in Java which is intended to be 100% compatible with the original. However, it is under development and needs help. Finally, RCaller is an alternative way of calling R from Java. It is packaged in a single jar and it does not require setup beyond the one-time installation of the R package *Runiversal*. It supports loading external packages, calling functions, handling plots and debugging the output generated by R. It is not the most efficient method compared to the alternatives, but users report that performance improvements in the latest revision and its simplicity of use make it an important tool in many applications.

# References

[Commons Math Developers(2010)] Commons Math Developers. Apache Commons Math, Release 2.1. Available from `http://commons.`

apache.org/math/download_math.cgi, Apr. 2010. URL http://commons.apache.org/math.

[Harner et al.(2009)Harner, Luo, and Tan] E. Harner, D. Luo, and J. Tan. JavaStat: A Java/R-based statistical computing environment. *Computational Statistics*, 24(2):295–302, May 2009.

[R Development Core Team(2011)] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL http://www.R-project.org/. ISBN 3-900051-07-0.

[RCaller Development Team(2011)] RCaller Development Team. RCaller: A library for calling R from Java, 2011. URL http://code.google.com/p/rcaller.

[Satman(2010)] M. H. Satman. *Runiversal: A Package for converting R objects to Java variables and XML.*, 2010. URL http://CRAN.R-project.org/package=Runiversal. R package version 1.0.1.

[Urbanek(2009)] S. Urbanek. How to talk to strangers: ways to leverage connectivity between R, Java and Objective C. *Computational Statistics*, 24(2):303–311, May 2009.

[Urbanek(2011a)] S. Urbanek. *rJava: Low-level R to Java interface*, 2011a. URL http://CRAN.R-project.org/package=rJava. R package version 0.9-2.

[Urbanek(2011b)] S. Urbanek. *Rserve: Binary R server*, 2011b. URL http://CRAN.R-project.org/package=Rserve. R package version 0.6-5.