The New BSF4ooRexx 6.00. In The New BSF4ooRexx 6.00, Hrsg. René Jansen, Chip Davis

Flatscher, Rony G.

Published: 01/03/2018

Document Version
Publisher's PDF, also known as Version of record

The New BSF4ooRexx 6.00

Rony G. Flatscher (Rony.Flatscher@wu.ac.at), WU Vienna
"The 2018 International Rexx Symposium", Aruba, Dutch West Indies
March 25th – 29th, 2018

Abstract. BSF4ooRexx is a Java bridge for the dynamic scripting language ooRexx. Version 6.00 increases the required level of Java from Java 1.4 to Java 1.6/6.0 and gets a totally rewritten reflection engine in two versions that enables it to also work efficiently with the module based versions of Java, introduced with Java 9 in the fall 2017. This article describes the most important changes to BSF4ooRexx since the 2017 International Rexx Symposium planned to be eventually released as version 6.00.

1 Introduction

BSF4ooRexx, the "Bean Scripting Framework for ooRexx", is a Java bridge for ooRexx, allowing ooRexx to interact with Java classes and Java objects, as if they were ooRexx classes and ooRexx objects by requiring the supplied ooRexx package `BSF.CLS`. As a result ooRexx programmers can simply send ooRexx messages to Java objects that will cause the appropriate Java methods to be invoked reflectively by BSF4ooRexx.

Since the first proof-of-concept implementation of a Rexx-to-Java bridge in 2000 many features got added and taking advantage of the powerful native APIs introduced with ooRexx 4.0 even wrapping ooRexx objects in Java objects became possible. The resulting BSF4ooRexx package has enabled ooRexx programmers to implement Java interface classes with ooRexx classes.[1]

Over the years BSF4ooRexx has become a complete bi-directional bridge that allows ooRexx to fully exploit Java and at the same time allows Java programmers to fully interact and exploit ooRexx, making it easy to add ooRexx as a scripting language to Java applications.

[1] The `BSF4ooRexx` external function `BsfCreateRexxProxy()` allows Rexx objects to be wrapped as Java objects. Java method invocations on the wrapped Java object will cause ooRexx messages to be sent to the wrapped ooRexx object transparently.

2 New Properties and Features in BSF4ooRexx 6.00

BSF4ooRexx version number "6.00"[2] indicates that it requires at least Java version 1.6, a.k.a. "Java 6", as the minimum Java version, whereas BSF4ooRexx 4.x required minimally Java version 1.4.[3]

2.1 New Java Version Information

The Java versioning scheme started out with Java 1.0 and each subsequent version was defined by adding 0.1 to it, yielding 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7 and 1.8 over the course of twenty years. The versions 1.5, 1.6, 1.7 and 1.8 were later renamed to 5, 6, 7 and 8 respectively. Starting with the Java version released in the fall of 2017 a new versioning scheme took effect by naming it 9 and in the future version numbers will be increased by adding 1 to the previous one.[4]

The ooRexx package BSF.CLS now defines new entries in the Rexx directory that Rexx programmers can fetch with the environment symbol .bsf4rexx:

- entry named "JAVA.VERSION": this will return the current Java version as reported by Java[5], e.g. the string "1.8.0_162".
- entry named "JAVA.MAJOR.VERSION": this will return the string "6" for Java 1.6, "7" for Java 1.7, "8" for Java 1.8, "9" for Java 9, "10" for Java 10 and so forth.
- entry named "JAVA.MINOR.VERSION": this will return the string "0_162" for Java version "1.8.0_162".

---

[2] The external Rexx function BSFVersion() will report in the first returned word the current version number as Jvv.YYYYMMDD: "J" indicates the minimum Java version, the "vv" number represents the version number followed by a dot and the sorted date of that particular release. This encoding makes it easy for Rexx programmers to extract the minimum Java version, the current BSF4ooRexx bridge version and the date of that particular release. Whenever the minimum required Java version "J" changes the version number "vv" will be reset to "00".

If requiring the package BSF.CLS then the environment symbol .bsf4rexx (a Rexx directory) stores the version number with the index name "VERSION". Hence .bsf4rexx~version will return the current BSF4ooRexx version number in the form Jvv.YYYYMMDD.

[3] Java 1.4 got introduced in 2002, Java 1.6 in December 2006, more than a decade ago [1]. As organisations and companies tend to run their application in stable environments it may be the case that there are still Java 1.6/6.0 deployments in the field for which the ooRexx-Java bridge should keep working.

[4] In 2017 the OpenJDK community decided to create a new Java version from Java 9 on every six months, cf. [2].

[5] The current Java version can be retrieved in Java with the following Java statement: java.lang.System.getProperty("java.version").

2.2 Masking Name Change from "MACOSX" to "DARWIN"

The ooRexx package BSF.CLS defines four operating system name releated entries in the ooRexx directory that Rexx programmers can fetch with the environment symbol .bsf4rexx:

- entry named "OPSYS": uppercased name of the operating system as the PARSE UPPER SOURCE keyword statement returns , e.g. "LINUX", "MACOSX", "WINDOWSNT",
- entry named "OPSYS1": the first (uppercase) letter of the operating system name, e.g. "L", "M", "W",
- entry named "OPSYS2": the first two (uppercase) letters of the operating system name, e.g. "LI", "MA", "WI"
- entry named "OPSYS3": the first three (uppercase) letter of the operating system name. e.g. "LIN", "MAC", "WIN".

These entries are meant to help BSF4ooRexx programmers, who need to determine the operating system under which their Rexx programs get executed.[6]

ooRexx used to report the Apple operating system as "MACOSX" but in ooRexx 5.0 beta that name got changed to the one returned by the operating system itself, i.e. "DARWIN". To allow BSF4ooRexx programs to continue to run unchanged this name change will be masked by keeping the "MACOSX", "M", "MA" and "MAC" values for the above four entries, although the ooRexx 5.0 PARSE SOURCE keyword statement returns "DARWIN".[7]

2.3 Adjusting Java 9 Support for MacOS

Apple's Java 1.6/6.0 runtime environment was installed with proprietary classes that support the integration with the MacOS desktop environment and concluded the Apple Java support with that version. Oracle's Java versions 1.7/7 and 1.8/8 would rely on Apple's Java 1.6/6.0 to be present for desktop integration.

Starting with Java 9 access to Apple specific[8] classes is not allowed anymore. The

[6] For example Windows and Unix file systems usually differ in their path and file separator characters and how a fully qualified path has to be assembled, such that it becomes important to learn the operating system the Rexx scripts is currently using in order to correctly build paths.

[7] The full operating system information can be retrieved using the ooRexx SysUtility function SysVersion().

[8] Examples of Apple specific support in their Java runtime environment are for instance the Java class "com.apple.eawt.Application" or the property "apple.laf.useScreenMenuBar".

BSF4ooRexx support for MacOS was adjusted in the BSF.CLS and the UNO.CLS packages accordingly and makes sure that starting with Java 9 on Apple the new Java alternatives for operating system independent desktop integration gets used like the new and portable Java 9 class "java.awt.Taskbar".

2.4 Slot Argument Now an Instance of Type Slot.Argument

Whenever the Java side causes a message to be sent to a Rexx object, the BSF4ooRexx infrastructure will append a slot argument of type Directory, which contains entries that may be helpful for the Rexx programmer. In order to allow Rexx programs to determine that an argument is a slot argument, the new type Slot.Argument[9], currently[10] a subclass of Directory, will be used for storing the Java context information of the message. This way Rexx programmers can easily and unambiguosly determine whether an argument is such a slot argument, e.g., with "if someArg~isA(.Slot.Argument) then …".

2.5 New Implementations of the Reflection Invocation Strategies

BSF4ooRexx uses Java reflection in order to find Java constructors, fields and methods, it needs to invoke on behalf of Rexx programs. Since its inception the Java language defines reflection classes in the Java package named java.lang.reflect. With the introduction of Java 1.7/7 the new Java package java.lang.invoke got introduced into the Java language allowing an alternative means of reflection in Java which has since been extended.

As the baseline of BSF4ooRexx 6.0 is Java 1.6/6.0 it is still necessary to use the Java package java.lang.reflect, which can be used on Java 1.7/7[11], 1.8/8 and higher as well. Applying the reflection via the Java package java.lang.invoke is only possible starting with Java 1.7/7. BSF4ooRexx by default uses the java.lang.invoke based reflection engine if the current Java version is 1.8/8 or higher.

Java 9 changed the rules for accessibility with the introduction of the module system, such that there may be reflective accessibility invocation problems that

[9] This follows a suggestion by Jon Wolfers at the International Rexx Symposium 2017.

[10] Once ooRexx 5.0 is released it is envisioned that the new ooRexx class StringTable gets subclassed instead of Directory.

[11] Although Java 1.7/7 introduced the new java.lang.invoke package, its implementation in the Java 1.7/7 runtime is not correct in a few Java classes (e.g. at runtime the hasNext() method will cause a java.lang.IllegalAccessException in: "class java.util.AbstractList$Itr.public boolean java.util.AbstractList$Itr.hasNext()") such that it is important for BSF4ooRexx to apply the java.lang.reflect package instead for Java 1.7/7.

cannot be easily overcome anymore.[12] Taking advantage of the classes in java.lang.invoke makes it possible to efficiently reflect and invoke at runtime on Java 9 too.

The BSF4ooRexx Java package org.rexxla.bsf.engines.rexx now includes two different classes for reflective invocations on behalf of Rexx programs. One employs the java.lang.reflect package (RexxReflectJava6) and one employs the java.lang.invoke package (RexxReflectJava7). By default RexxReflectJava7 will be used starting with Java 1.8/8.[13]

2.6 Allowing DO...OVER Loops on all Kinds of Java Collections

The collection classes that the Java runtime environments supply may have different kind of means to iterate over the collected objects. With BSF4ooRexx 6.00 any object from a Java class implementing one of the following Java interface classes can be used in ooRexx do...over loops:

- java.lang.Enum[14]
- java.lang.Iterable
- java.util.Collection
- java.util.Enumeration
- java.util.Iterator
- java.util.Map

There are two ooRexx proxy methods that get added to such proxy objects: makeArray and supplier, which both allow iterating over collection objects on Rexx.

The ooRexx do...over loop mechanism will send the makeArray message to the collection object and iterate over the elements in the returned Rexx array. This way it becomes possible to iterate over any Java collection with the ooRexx semantics.

[12] Starting with Java 9 it may be the case that classes from modules that are not exported to classes that employ reflection upon them, cannot be interacted with. E.g. it would not be possible to overrule missing accessibility rights by invoking the setAccessible(true) method on a non-exported, reflected object!

[13] For a transitional period from Java 9 up, the method setAccessible(true) will be tolerated, but appropriate runtime warnings will get issued. Using BSF4ooRexx new reflection infrastructure will cause these warnings to be issued, however, once the Java runtime will create runtime exceptions the implemented BSF4ooRexx reflective algorithm will react accordingly, making sure that accessible superclasses will get used to successfully invoke methods reflectively.

[14] The Java class java.lang.Enum gets used as the base class for any enum type by the Java compiler. An Enum type serves among other things as a collection of its enum values.

The new sample program "samples/1-070_demoDoOver.rxj" in the BSF4ooRexx installation package demonstrates this new feature.

2.7 New Source Options in BsfCreateRexxProxy()

The external BSF4ooRexx function BsfCreateRexxProxy(rexxObject,…) allows a Rexx object to be boxed (wrapped) as a Java object. Methods invoked by Java programs in such a proxy object will cause a Rexx message by the name of the invoked Java method to be sent synchroneously to its contained (boxed) ooRexx object.

However, if the first argument rexxObject is of type string, method, or – new in 6.00 – of type routine, array then it is by default taken as the Rexx code that should get executed whenever a message gets received from Java.[15] The semantics depend on the type of the first argument:

- string, method, or an array (new in version 6.00): represents the code of an unknown method that runs upon each received message; the code needs to fetch the supplied arguments in the unknown method sequence: messageName, messagArgs (an array of arguments, if any),
- routine: the code will be invoked using call, supplying the arguments in the unknown method sequence: messageName, messagArgs (an array of arguments, if any),

If it is intended to box (wrap) the first Rexx argument of type string, array, method, or routine as is, then in this particular case that the first argument's type is string, method, array, or routine one needs to supply the string argument "REXXOBJECT" as the third argument to the BsfCreateRexxProxy() external Rexx function call. Supplying such a boxed (wrapped) Rexx object to Java will allow Java programmers to interact directly with the Rexx object using Rexx messages.[16]

2.8 Improving Error Information for FXML Invoked Rexx Code

The BSF4ooRexx RexxScriptEngine implementation [7] allows ooRexx to be used as a normal Java scripting language using Java's scripting framework as implemented in the javax.script package and introduced with version 1.6/6.0 in 2006.

[15] In this scenario the Rexx program "BSF_OnTheFly.cls" gets employed in order to create the appropriate Rexx object.

[16] Cf. the Javadocs documenting the class org.rexxla.bsf.engines.rexx.RexxProxy and its methods starting with the name sendMessage.

One area where the Java scripting framework gets deployed is JavaFX. It makes it possible to define Java scripting programming languages in FXML user interface definition files that should be employed for invoking programs and event handler code [8]. Unfortunately, as of Java 9 JavaFX does not supply the file name which is the source of the code that the receiving script engine gets for execution.[17] In the case of a runtime error in the executed script, the script engine is therefore not able to supply the name of the script file that caused a runtime error, making it very difficult and cumbersome to debug such script programs.

The RexxScriptEngine implementation got enhanced to supply at least the name of the hosting FXML file for which the script engine executes the submitted script code if there is no "javax.script.filename" entry in the engine scope Bindings in the ScriptContext.[18] This way Rexx programmers gain at least the ability to identify the hosting file that invoked the Rexx script code.

2.9 "Shebang Line" in all BSF4ooRexx Rexx Programs

A "shebang line" [3] in a script starts with the character sequence #! (the number sign followed by an exclamation mark) followed by a program that is supposed to execute the script file containing that shebang. If such a script file gets the execution attribute set, it can be executed by just supplying the script file name. The operating system will extract the program from the shebang line that is to be used to execute the script.

In order to take advantage of this Unix mechanism for the BSF4ooRexx Rexx programs, all Rexx scripts were changed to define the shebang line as:[19]

```
#!/usr/bin/env rexx
```

This will cause, with the help of the Unix env program, the Rexx interpreter rexx to be located in one of the directories that the PATH environment variable defines which then gets used for executing the script.

---

[17] The class javafx.fxml.FXMLLoader does not supply the entry "javax.script.filename" in the engine scope Bindings in the ScriptContext as of Java 9.

[18] RexxScriptEngine uses the location URL attribute from the ScriptContext for this purpose.

[19] The shebang line must be the first line and the line end character must be the single character LF (linefeed: "0A"x), *not* the Windows end-of-line sequence CR-LF (carriage return-linefeed: "0d0a"x)!

2.10 New External Rexx Function "bsfTestPing()"

To aid in the development and the profiling of the BSF4ooRexx framework a new external Rexx function "bsfTestPing([numberCallsToJava])" got created:

- with no argument given, this function immediately returns to the Rexx program,
- if a number is given as a single argument it specifies the number of times a Java test method, which immediately returns to C++, should be called via JNI[20].

This way it becomes possible to measure the overhead of calling an external Rexx function from Rexx, but also to measure the overhead of calling a Java method from native code.

2.11 New BSF()-Subfunction "testPing"

The external Rexx function BSF() was extended with a new subfunction named "testPing". It is meant for measuring round trip times from Java to native C/C++ code and from Java to Rexx and always returns .nil. Its syntax is:

- BSF("testPing"[,repetitions=1[,rexxObject,messageName]])

repetitions determines the number of times a native C++ function, that immediately returns, gets called from Java. It defaults to 1.

If the third argument is given, then it denotes a rexxObject to which the message named messageName gets sent repetitions times from Java (via native C/C++ code to the ooRexx interpreter and back).

This way it becomes possible to measure the overhead of calling native code from Java and the overhead of sending messages to Rexx objects from Java.

2.12 Reworking Package[21] BSF.CLS

The ooRexx package BSF.CLS camouflages Java as ooRexx and is used to take full advantage of the Java bridge. BSF.CLS formerly contained numerous expensive Rexx interpret keyword statements. With the advent of the routine class in ooRexx an alternative to dynamically create and invoke Rexx routines has become

[20] JNI is the acronym for Java Native Interface which allows direct interactions between C/C++ and Java.

[21] In ooRexx 5.0 the term "package" denotes a file that contains Rexx code and directives. The Rexx code before the first directive in this context is named *"prolog"*.

available which compared to the interpret keyword statements can be invoked faster.

All interpret keyword statements got replaced by using routine objects instead and in the process caching them. In addition the external (BSF4ooRexx) Rexx functions get cached as ooRexx routine objects as well!

As a result of this work the invocation of BSF-related routines on ooRexx 4.2 became all in all about twenty times faster compared to the previous version of BSF.CLS!

2.13 New "bsf.compile()" Routine in BSF.CLS

ooRexx 5.0 new resource directive allows any text to become part of an ooRexx package (program) including programs in any other programming language.[22] Therefore it may be helpful at times, if Rexx programmers become able to define Java programs that get compiled dynamically at runtime. The new BSF4ooRexx sample "samples/9-030_compileJavaClassAndUseIt.rxj" demonstrates this ability.

The syntax of the new bsf.compile[23] routine is:

- bsf.compile(fullJavaClassName, javaSourceCode)

This routine returns an ooRexx proxy class for the created Java class object which understands the new message for creating instances.

fullJavaClassName is the fully qualified name of the created Java class that may include package names. javaSourceCode is either a string or an array of strings containing the Java code to compile.

2.14 Supporting GUI-Thread Interaction from Non-GUI-Thread

Java's awt/swing and JavaFX GUI (acronym for: "graphical user interface") classes allow direct interaction with GUI objects only from its GUI thread, otherwise the GUI may hang or exhibit erroneous behaviour. If Rexx programs wish to interact with GUI objects from a different thread then it becomes necessary to use Java's awt/swing or JavaFX means for making sure that Rexx interaction with Java GUI

[22] ooRexx will allow access to any resource direcitve value via the environment symbol .resources, a Rexx directory whose indices are the names of the resources defined on the resource directive.

[23] The current implementation uses the opensource Janino compiler, cf. [4].

objects occur on the GUI thread[24].

It is quite easy to employ the necessary Java methods from Rexx to make sure that Rexx code will be executed on the GUI thread such that it is safe for it to interact with the GUI objects for those in the know. In real world scenarios it has turned out to be quite difficult to employ the Java infrastructure for proper GUI-interaction for Rexx programmers, resulting in error-prone GUI programs making it very time-consuming to debug such scenarios.[25]

To allow an easy to be applied solution for ooRexx programmers two ooRexx classes got created that allow updating any JavaFX GUI from a non-GUI thread from Rexx in a way that looks natural and therefore easy to ooRexx programmers. The solution takes advantage of the ooRexx message based architecture and defines two public classes in `BSF.CLS`:

- `FXGuiThread`, used to later dispatch messages on the GUI thread, supplying the following class methods for this purpose:
  - `isGuiThread`: returns `.true`, if current thread is the JavaFX GUI thread (safe to interact directly with GUI objects), `.false` else,
  - `runLater(target,messageName[,"I"|"A",arg…])`[26]: creates and queues a `GUIMessage` object that sends `messageName` with the supplied arguments to the `target` object on the JavaFX GUI thread later that gets returned to the caller,
  - `runLaterLatest(target,messageName[,I"|"A",arg…])`: will remove any queued `GUIMessage` objects directed at `target` with the same `messageName` before creating and queuing a new `GUIMessage` object as `runLater()` above.
- `GUIMessage`: this class is modelled after the ooRexx `Message` class, such that the reader can consult the ooRexx reference (`rexxref.pdf`) for the documentation of the following methods and attributes:
  - `arguments`

[24] For awt/swing GUIs one can use the static utility method `invokeLater(java.lang.Runnable)` in the class `javax.swing.SwingUtilities` For JavaFX GUIs one can use the static utility method `runLater(java.lang.Runnable)` in the class `javafx.application.Platform`.

[25] These observations were gained with students at WU [5], who created ooRexx applications with `JavaFX` together with the need to update the `JavaFX` GUI off a Rexx thread.

[26] The arguments are exactly the same as the ones needed to create an ooRexx `message` object.

- completed
- hasArguments
- hasError
- hasResult
- errorCondition
- messageName
- result
- target

Although the target object usually will be some JavaFX GUI object, this is not necessary. Alternatively one could instead send any ooRexx message to any ooRexx object later, making sure that it will be dispatched on the GUI thread where it is safe to directly interact with the GUI objects.

3 Roundup and Outlook

This article introduced new features to the BSF4ooRexx external function package that bridges ooRexx and Java. To indicate the minimum Java version that gets supported the major version number got changed to 6 to indicate that Java version 1.6/6.0 is required at least.

The most significant change has to do with the reflection part of the package which got totally rewritten and for which two versions exist: one for Java 1.6/6 and 1.7/7 employing java.lang.reflect, and one for Java 1.8/8 and up employing the java.lang.invoke by default.

It is planned that the ability for ooRexx programmers to easily update JavaFX GUIs from a non-GUI thread (using the classes FXGuiThread and GUIMessage defined in the package BSF.CLS) gets ported to awt/swing GUIs adding perhaps a class AwtGUIThread with the same class methods and behavior as implemented in the FXGuitThread class.

It is planned that once ooRexx 5.0 gets released, BSF4ooRexx will get released as well supporting ooRexx 4.1 and later. That released version of BSF4ooRexx then should be immediately reworked to exploit some of the new features introduced with ooRexx 5.0 like the package local directory, but also the new and faster StringTable class instead of the Directory class, if backward compatibility can be maintained.

4 References

[1] "Java version history", Wikipedia (as of 2018-03-01): https://en.wikipedia.org/wiki/Java_version_history

[2] "Open JDK Homepage" (as of 2018-03-01): https://openjdk.java.net/

[3] "Shebang", Wikipedia (as of 2018-03-01): https://en.wikipedia.org/wiki/Shebang_(Unix)

[4] "Janino Homepage" (as of 2018-03-01): http://janino-compiler.github.io/janino/

[5] "WU – Wirtschaftsuniversität Wien/Vienna University of Economics and Business" (as of 2018-03-01): https://www.wu.ac.at/

[6] Javadocs for the Java package javax.script (as of 2018-03-01): https://docs.oracle.com/javase/8/docs/api/javax/script/package-summary.html

[7] Flatscher R.G.: "'RexxScript' – Rexx Scripts Hosted and Evaluated by Java (Package javax.script)", in: Proceedings of the "The 2017 International Rexx Symposium", Amsterdam, The Netherlands, April 9th - 12th 2017. URL (as of 2018-03-01): http://www.rexxla.org/events/2017/presentations/201704-RexxScript-Article.pdf

[8] Flatscher R.G.: "JavaFX for ooRexx – Creating Powerful Portable GUIs for ooRexx", in: Proceedings of the "The 2017 International Rexx Symposium", Amsterdam, The Netherlands, April 9th - 12th 2017. URL (as of 2018-03-01): http://www.rexxla.org/events/2017/presentations/201704-RexxScript-Article.pdf

[9] Sourceforge homepage BSF4ooRexx (acronym for "bean scripting framework – BSF – for ooRexx"), an ooRexx and Java bridge (as of 2018-03-01): https://sourceforge.net/projects/bsf4oorex

[10] Sourceforge homepage ooRexx ("open object-oriented Rexx"), a dynamically typed scripting language (as of 2018-03-01): http://www.rexxla.org/events/2017/presentations/201711-ooRexx-JavaFX-Article.pdf

[11] Download page for ooRexx 5.0 beta (as of 2018-03-01): https://sourceforge.net/projects/oorexx/files/oorexx/5.0.0beta/