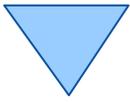




"Use Rexx and ooRexx from its Java-based Sibling NetRexx"

2010 International Rexx Symposium
Amsterdam/Almere, Netherlands
(December 2010)

© 2010 Rony G. Flatscher (Rony.Flatscher@wu.ac.at)
Wirtschaftsuniversität Wien, Austria (<http://www.wu.ac.at>)

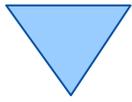


Agenda

- BSF4Rexx
 - Invoking Rexx/ooRexx programs
 - Supplying argument(s)
 - Fetching return values
- BSF4ooRexx
 - Java callbacks to ooRexx
 - Interacting with ooRexx objects
 - Intercepting ooRexx conditions
- Roundup

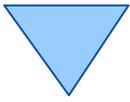
▼ Why NetRexx? Why [oo]Rexx?

- Using NetRexx
 - Much simpler than Java
 - NetRexx programmers very likely to be acquainted with Rexx
 - Possibly acquainted with ooRexx
- Take full advantage of an external Rexx/ooRexx interpreter
 - Exploit what Rexx/ooRexx supplies
 - Use existing Rexx/ooRexx programs
 - Solve problems hardly solvable in Java/NetRexx



BSF4Rexx

- Home: <http://wi.wu-wien.ac.at/rgf/rexx/bsf4rexx/current/>
 - Can be used for any RexxSAA interpreter
- Development started in 2000 (!)
- Allow Java to invoke Rexx programs
- Allow Rexx programs to interact with Java
- Special ooRexx-support makes interacting with Java easier compared to classic Rexx
 - Package "BSF.CLS"
 - Package "UNO.CLS"



BSF4ooRexx

- Home: <http://wi.wu-wien.ac.at/rgf/rexx/bsf4oorexx/current/>
 - First available in fall 2009
- Exploits new ooRexx 4.0 kernel
 - Ability for Java callbacks implemented in ooRexx
 - Implement Java interface methods in ooRexx!
 - Adds ability for Java/NetRexx to interact directly with ooRexx objects!
 - Send ooRexx messages to ooRexx objects
 - Adds ability to catch ooRexx runtime errors
 - Makes the ooRexx condition object available to Java/NetRexx



Java Invoking a Rexx Script

An Example using `exec()`

```
import org.apache.bsf.*;    // BSF support
import java.io.*;          // exception handling

public class TestSimpleExec {

    public static void main (String[] args) throws IOException
    {
        try {
            BSFManager mgr = new BSFManager ();
            BSFEngine  rexx = mgr.loadScriptingEngine("rexx");
            String      rexxCode = "SAY 'Rexx was here!'";

            rexx.exec ("rexx", 0, 0, rexxCode);

        } catch (BSFException e) { e.printStackTrace(); }
    }
}
```

Output:

Rexx was here!



NetRexx Invoking a Rexx Script

An Example using `exec()`

```
import org.apache.bsf.  
  
mgr      =BSFManager()  
Rexx     =mgr.loadScriptingEngine("rex")  
rexCode="SAY 'Rexx was here!'"  
do  
    rexxEngine.exec("rex",0,0,rexCode)  
catch e=BSFException  
    e.printStackTrace  
end
```

Output:

Rexx was here!



org.apache.bsf.BSFEngine

Methods to Execute Script Code

- `exec(srcName,lineNo,colNo,script)`
 - Executes script, no arguments, no return value
- `eval(srcName,lineNo,colNo,script)`
 - Evaluates script, no arguments, returns result value
- `apply(srcName,lineNo,colNo,script,names, args)`
 - Executes script with arguments, returns result value
- `call(object,methodName,args)`
 - Executes method in object supplying arguments, returns result value
 - Available only in BSF4ooRexx, see [RexxEngine](#) docs
 - Easier to use RexxProxy-methods instead



NetRexx Invoking a Rexx Script, 1

Executing Rexx script using apply()

```
/* 'Hello world!' with NetRexx */
parse source . . src; parse version v; say src/"v": Hello world!"

/* 'Hello world!' using a Rexx interpreter */
rexCode="parse source . . src; parse version v; say src/'v': Hello world!"
org.apache.bsf.BSFManager().apply("rex",src,0,0,rexCode,null,null)
```

Output:

```
rgf_01_runRexx.nrx/NetRexx 2.05 14 Jan 2005: Hello world!
rgf_01_runRexx.nrx/REXX-ooRexx_4.1.0(MT) 6.03 2 Nov 2010: Hello world!
```



NetRexx Invoking a Rexx Script, 2 Supplying an Argument to Rexx

```
parse source . . src

rexCode="parse arg str; say 'Rexx received:' str"

vArgs=Vector()
vArgs.addElement("Hello from NetRexx")
org.apache.bsf.BSFManager().apply("rex",src,0,0,rexCode,null,vArgs)
```

Output:

Rexx received: Hello from NetRexx



NetRexx Invoking a Rexx Script, 3 Returning Edited Argument from Rexx

```
parse source . . src

rexCode="parse arg str; return reverse(str)"

vArgs=Vector()
vArgs.addElement("Hello from NetRexx")
result=org.apache.bsf.BSFManager().apply("rex",src,0,0,rexCode,null,vArgs)

say "NetRexx received:" result
```

Output:

```
NetRexx received: xxERteN morf olleH
```

NetRexx Invoking a Rexx Script, 4 Querying Process' Environment

```
parse source . . src
```

```
rexCode="return value(arg(1),, 'ENVIRONMENT')"
```

```
vArgs=Vector()
```

```
vArgs.addElement("PATH")
```

```
result=org.apache.bsf.BSFManager().apply("rex",src,0,0,rexCode,null,vArgs)
```

```
say "PATH:" result
```

Possible Output (Line-breaks by Presentation Program):

```
PATH:
```

```
E:\jdk1.6.0_18\bin;E:\rony\dev\bsf\src\bsf4oorex;D:\Programme\Java\jre6\bin\client;E:\rony\dev\bsf\src\bsf4oorex;D:\WINDOWS\system32;D:\WINDOWS;D:\WINDOWS\System32\Wbem;e:\cygwin\bin;e:\rony\tools;E:\vslick\win;E:\Programme\GNU\GnuPG\pub;D:\Programme\Gemeinsame  
Dateien\GTK\2.0\bin;D:\WINDOWS\system32;D:\WINDOWS;D:\WINDOWS\System32\Wbem;D:\Programme\TortoiseSVN\bin;D:\Programme\sK1 Project\UniConvertor-  
1.1.5\;D:\Programme\sK1 Project\UniConvertor-  
1.1.5\DLLs;D:\Programme\QuickTime\QTSystem\;D:\Programme\ooRexx;D:\Programme\SSH Communications Security\SSH Secure  
Shell;E:\jdk1.6.0_18\jre\bin\client;e:\rony\dev\bsf\src\bsf4oorex
```

NetRexx Invoking a Rexx Script, 5

Interacting with Process' Environment

```
parse source . . src
envName ="RexxLA"
value="<Rexx Language Association>"
rexCode='if arg()=1 then return value(arg(1),,"ENVIRONMENT");' -
        'call value arg(1),arg(2),"ENVIRONMENT"'

rexEngine=org.apache.bsf.BSFManager().loadScriptingEngine("rex")

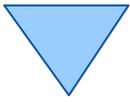
vArgs1=Vector()
vArgs1.addElement(envName)
result=rexEngine.apply(src,0,0,rexCode,null,vArgs1)
say 'Value of process environment variable "'envName"' : '['result']'

say 'Defining a process environment variable named "'envName"' ...'
vArgs2=Vector()
vArgs2.addElement(envName)
vArgs2.addElement(value)
rexEngine.apply(src,0,0,rexCode,null,vArgs2)

result=rexEngine.apply(src,0,0,rexCode,null,vArgs1)
say 'Value of process environment variable "'envName"' : '['result']'
```

Output:

```
Value of process environment variable "RexxLA": []
Defining a process environment variable named "RexxLA" ...
Value of process environment variable "RexxLA": [<Rexx Language Association>]
```



BSF4ooRexx

- Numerous examples
 - "BSF4ooRexx/samples"
 - "Nutshell programs"
 - Emphasize different possibilities BSF4ooRexx/Java allows for
 - Executable and should be easy to study/learn from them
 - "BSF4ooRexx/samples/Java" showcase
 - Invoke ooRexx from Java, supplying a Java object
 - Fetching Java object via its bean name
 - Fetching Java object as an ooRexx object
 - Sending messages from Java to ooRexx objects
 - Catching ooRexx conditions in Java

▼ Parse Java Object by BeanName, 1

- BSF4ooRexx invokes ooRexx program
 - Java objects as arguments are supplied by their BeanName (a string)
 - unique index value into the BSFRegistry
 - ooRexx program needs to use the public routine `bsf.wrap()` to create an ooRexx proxy object for the Java object in the BSFRegistry
 - `bsf.wrap()` is defined in the package "`BSF.CLS`"
 - BeanName needs to have "`<O>`" prepended to indicate that the string is a reference to a Java object

Parse Java Object by BeanName, 2

"nrxRunRexx_01.nrx"

```
parse source . . src
```

```
rexCode= "parse arg beanName           ;" -  
         "say 'beanName           :' beanName           ;" -  
         "javaObj=bsf.wrap('<0>' || beanName)           ;" - /* Insider-Knowhow */  
         "say 'javaObj~class       :' javaObj~class       ;" -  
         "say 'javaObj~toString:' javaObj~toString       ;" -  
         ">::requires BSF.CLS           ;"
```

```
vArgs=Vector()  
vArgs.addElement( System.getProperties )
```

```
org.apache.bsf.BSFManager().apply("rex",src,0,0,rexCode,null,vArgs)
```

Possible Output (Line-breaks by Presentation Program):

```
beanName       : java.util.Properties@9971ad  
javaObj~class  : The BSF_REFERENCE class  
javaObj~toString: {java.runtime.name=Java(TM) SE Runtime Environment,  
sun.boot.library.path=E:\jdk1.6.0_18\jre\bin, java.vm.version=16.0-b13,  
java.vm.vendor=Sun Microsystems Inc., java.vendor.url=http://java.sun.com/,  
path.separator=;, java.vm.name=Java HotSpot(TM) Client VM,  
file.encoding.pkg=sun.io, sun.java.launcher=SUN_STANDARD, user.country=AT,  
sun.os.patch.level=Service Pack 3, ...
```

Use Java Object as an ooRexx Object, 1

"nrxRunRexx_02.nrx"

- BSF4ooRexx invokes ooRexx program
 - Java objects as arguments are supplied by their BeanName (a string)
 - unique index value into the BSFRegistry
 - If BSF4ooRexx detects that the public routine `bsf.wrap()` is available in the interpreter instance, then it will create an ooRexx proxy object for each Java object argument, then
 - **USE ARG** will fetch directly the ooRexx proxy object
 - **PARSE ARG** would parse the BeanName

Use Java Object as an ooRexx Object, 2

"nrxRunRexx_02.nrx"

```
parse source . . src
```

```
rexEngine=org.apache.bsf.BSFManager().loadScriptingEngine("rex")  
rexEngine.apply(src,0,0,"::requires BSF.CLS",null,null)
```

```
rexCode= "use arg javaObj          ;" -  
         "say 'javaObj~class      :' javaObj~class      ;" -  
         "say 'javaObj~toString:' javaObj~toString    ;" -  
         "::requires BSF.CLS      ;" -
```

```
vArgs=Vector()  
vArgs.addElement( System.getProperties )
```

```
rexEngine.apply(src,0,0,rexCode,null,vArgs)
```

Possible Output (Line-breaks by Presentation Program):

```
bjavaObj~class      : The BSF_REFERENCE class  
javaObj~toString: {java.runtime.name=Java(TM) SE Runtime Environment,  
sun.boot.library.path=E:\jdk1.6.0_18\jre\bin, java.vm.version=16.0-b13,  
java.vm.vendor=Sun Microsystems Inc., java.vendor.url=http://java.sun.com/  
path.separator=;, java.vm.name=Java HotSpot(TM) Client VM,  
file.encoding.pkg=sun.io, sun.java.launcher=SUN_STANDARD, user.country=AT, ...
```

▼ Sending Messages to ooRexx Objects, 1

"nrxRunRexx_03.nrx"

- ooRexx objects can be made available to Java/NetRexx
 - [org.rexxla.bsf.engines.rexx.RexxProxy](#) class
 - Supplies methods to send messages to the ooRexx objects via the Java [RexxProxy](#) objects
 - Modelled after ooRexx 4 kernel APIs e.g.
 - `sendMessage(msgName)`
 - `sendMessage(msgName,arg1)`
 - `sendMessage(msgName,arg1,arg2) ...`
 - Hence easy for Java/NetRexx programmers to send ooRexx objects messages

Sending Messages to ooRexx Objects, 2

"nrxRunRexx_03.nrx"

```
import org.rexxla.bsf.engines.rexx.  
parse source . . src  
  
rexxEngine=org.apache.bsf.BSFManager().loadScriptingEngine("rexx")  
  
rexxCode="d=.directory~new ;" -  
        "d~ooRexx='Open Object Rexx' ;" -  
        "d~BSF4ooRexx='Bean Scripting Framework for Open Object Rexx' ;" -  
        "return d ;" -  
        "::requires BSF.CLS ;"  
  
rp=RexxProxy rexxEngine.apply(src,0,0, rexxCode,null,null);  
  
say 'rp~ooRexx           = [' rp.sendMessage0("ooRexx") ']'  
say 'rp~entry("ooRexx") = [' rp.sendMessage1("entry", "ooRexx") ']'  
say 'rp["00REXX"]       = [' rp.sendMessage1("[ ]", "00REXX") ']\n'  
  
say 'rp~BSF4ooRexx      = [' rp.sendMessage0("Bsf4ooRexx") ']'  
say 'rp~entry("BSF4ooRexx") = [' rp.sendMessage1("entry", "BSF4ooRexx") ']'  
say 'rp["BSF400REXX"]   = [' rp.sendMessage1("[ ]", "BSF400REXX") ']'
```

Output:

```
rp~ooRexx           = [Open Object Rexx]  
rp~entry("ooRexx")  = [Open Object Rexx]  
rp["00REXX"]       = [Open Object Rexx]
```

```
rp~BSF4ooRexx      = [Bean Scripting Framework for Open Object Rexx]  
rp~entry("BSF4ooRexx") = [Bean Scripting Framework for Open Object Rexx]  
rp["BSF400REXX"]   = [Bean Scripting Framework for Open Object Rexx]
```

▼ Catching ooRexx Conditions, 1

"nrxRunRexx_04.nrx"

- ooRexx conditions (exceptions) can be caught
- Can be raised as a result of
 - Running a Rexx program that raises a condition
 - Sending a message to an ooRexx object
- Specific Java exception class
 - [org.rexxla.bsf.engines.rexx.RexxException](#) class
 - Method [getRexxConditionObject\(\)](#) returns the [RexxProxy](#) condition object
 - Java/NetRexx can send ooRexx messages to it

Catching ooRexx Conditions, 2

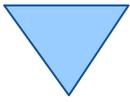
```
import org.rexxla.bsf.engines.rexx.  
parse source . . src  
  
rexxEngine=org.apache.bsf.BSFManager().loadScriptingEngine("rexx")  
  
rexxCode= "a=1                                     \n" -  
          "b=0                                     \n" -  
          "say a/b  -- cause a syntax error at runtime \n" -  
          ">::requires BSF.CLS                      "  
  
do  
    result=rexxEngine.apply(src,0,0,rexxCode,null,null)  
catch re=RexxException  
    System.out.println("Rexx program threw an exception:\n")  
    rp=re.getRexxConditionObject()  
  
    say '      rp.sendMessage("condition")           : [' rp.sendMessage("condition") ']'  
    say '      rp.sendMessage("code")                : [' rp.sendMessage("code") ']'  
    say '      rp.sendMessage("message")             : [' rp.sendMessage("message") ']'  
    say '      rp.sendMessage("program")             : [' rp.sendMessage("program") ']\n'  
  
    traceBack=RexxProxy rp.sendMessage("traceback")  
    say '      traceBack.sendMessage("firstItem"): \n\t[' traceBack.sendMessage("firstItem") ']'  
end
```

Output:

Rexx program threw an exception:

```
rp.sendMessage("condition")           : [SYNTAX]  
rp.sendMessage("code")                : [42.3]  
rp.sendMessage("message")             : [Arithmetic overflow; divisor must not be zero]  
rp.sendMessage("program")             : [nrxRunRexx_04.nrx]
```

```
traceBack.sendMessage("firstItem"):  
[      3 *-* say a/b  -- cause a syntax error at runtime ]
```



Roundup and Outlook

- Using BSF4Rexx/BSF4ooRexx is easy
- Allows
 - Executing Rexx and ooRexx programs
 - Submitting arguments to Rexx
 - Fetching return values from Rexx
 - Sending messages to ooRexx objects
 - Catching Rexx conditions
 - Allows referring to the ooRexx condition object
- Powerful !
- Questions ?