New Features in BSF4ooRexx
(Camouflaging Java as ooRexx)

http://sourceforge.net/projects/bsf4oorexx/files/

The 2015 International Rexx Symposium

Rony G. Flatscher

Wirtschaftsuniversität Wien ■ Welthandelsplatz 1 ■ A-1020 Wien

Agenda

- Purpose of BSF4ooRexx
- Changes to BSF4ooRexx since Aruba-Symposium
- Roundup and outlook

Purpose of BSF4ooRexx, 1

- Java
  - Available on practically all computers
  - Java's runtime environment (JRE)
    - A huge class library, available for all operating systems
      - Java classes (and programs) run everywhere, even GUIs !
      - Practically every relevant software problem/challenge solved
    - Constantly updated to the latest technologies
    - Used for
      - Java applets running in web browsers
      - Used for stand-alone, fully fledged professional applications

Purpose of BSF4ooRexx, 2

- BSF4ooRexx
  - Make all of Java (classes and the runtime) available to ooRexx
  - Instead of Assembler or C(++) external function libraries for special needs, *all of Java is made available as a huge external function (and class) library!*
    - One size fits all ;-)
      - No need anymore to create separate external function libraries for specific functionality one is seeking!
  - *Make it easy* for ooRexx programmers to take advantage
    - *Camouflage all of Java as ooRexx !*

Purpose of BSF4ooRexx, 3

- BSF4ooRexx
  - ooRexx code can be even used instead of Java code!
    - Interface methods
      - Often used for callback functionality, e.g. in Java GUI classes
    - Abstract methods
  - Java/NetRexx code can control invocation of Rexx scripts
    - Allows ooRexx to be used as a macro language for Java apps!
      - Can create and use arbitrary many ooRexx interpreter instances
    - Can interact with individual ooRexx objects
      - Send oo-Rexx messages with or without arguments
      - Fetch return values from ooRexx scripts

Purpose of BSF4ooRexx, 4
An Example

```
dim=.bsf~new("java.awt.Dimension", 100, 200)
say dim~toString

::requires BSF.CLS       -- get Java support
```

```
Output:

java.awt.Dimension[width=100,height=200]
```

Changes and New Features, 1

- Installation
  - Improve Windows installer
    - Windows XP “runas” does not work in latest versions of Windows
    - Use elevation introduced with Vista
  - Improve MacOSX installer
    - Supply ooRexx 4.2.0 with BSF4ooRexx
    - Circumvent a bug in AOO 4.0.x (PATH not set)
    - Do not preload awt-related classes as AOO 4.0.x and 4.1.x cannot dispatch internal ooRexx macros under certain cirumstances

Changes and New Features, 2

- New „ooRexxTry.rxj”
  - Enhances existing ooRexxTry.rxj
    - Undockable windows
    - More configuration features
  - Distinguishes between regular and trace output
  - Bachelor paper by Gerald Leitner, finalized in February 2015

Changes and New Features, 3

- "Auto-attach" to appropriate Java environment
  - Previously: ooRexx multithreading
    - Save thread ID that can interact with Java
    - Each ooRexx thread needs to use BSFAttachToTID(tid)
      - ooRexx programmer must somehow communicate the tid to use in new ooRexx threads
    - Each ooRexx thread should then use BSFDetach()
    - Cumbersome, hence also error-prone
  - Now:
    - BSF4ooRexx will *automatically attach and detach*
    - Implementation will be simplified, possibly speed improved

Changes and New Features, 4
Multithreading Example 1a

```
jo=.test~new
say "main: tid="||BSFGetTID() "worker, tid="tid
jo~worker
sleepTime=1.5
say "main: tid="||BSFGetTID() "about to sleep" sleepTime" secs"
call syssleep sleepTime
say "main: tid="||BSFGetTID() "finish. <--"

::requires BSF.CLS   -- use all of Java as a huge external ooRexx library!

::class test
::method worker      -- use ooRexx multithreading
  tid=BSFGetTID()    -- save current (main) TID
  jo=.BSF~new("java.awt.Dimension", 123, 456)
  reply              -- return to caller

  call BSFAttachToTID tid  -- connect to main TID
  do i=1 to 3
     call sysSleep random(0,100)/100
     say "worker: tid=" || BSFGetTID()", jo~toString:" jo~toString
  end
  call BSFDetach    -- detach from TID
```

Changes and New Features, 4
Multithreading Example 1b

Output:

```
E:\201504-RexxLA\>attach02.rex
main: tid=5608 worker, tid=TID
main: tid=5608 about to sleep 1.5 secs
worker: tid=4596, jo~toString: java.awt.Dimension[width=123,height=456]
worker: tid=4596, jo~toString: java.awt.Dimension[width=123,height=456]
main: tid=5608 finish. <--
worker: tid=4596, jo~toString: java.awt.Dimension[width=123,height=456]
```

Changes and New Features, 4
Multithreading Example 1c

```
jo=.test~new
say "main: tid="||BSFGetTID() "worker, tid="tid
jo~worker
sleepTime=1.5
say "main: tid="||BSFGetTID() "about to sleep" sleepTime" secs"
call syssleep sleepTime
say "main: tid="||BSFGetTID() "finish. <--"

::requires BSF.CLS   -- use all of Java as a huge external ooRexx library!

::class test
::method worker      -- use ooRexx multithreading

  jo=.BSF~new("java.awt.Dimension", 123, 456)
  reply               -- return to caller


  do i=1 to 3
     call sysSleep random(0,100)/100
     say "worker: tid=" || BSFGetTID()", jo~toString:" jo~toString
  end

```

Changes and New Features, 5

- Method resolution
  - Change resolving Java methods
    - Allows default *interface methods in Java 8* to become accessible
      - Java interface classes have not methods by definition
      - Starting with Java 8 a default interface method can be defined
    - Important for lambda support
      - Java package java.util.function.*
      - ooRexx can be used to implement any of the Java lambda functions!

Changes and New Features, 6
Lambda Example 1a (demo-j8-lambda.rex)

```
#!/usr/bin/rexx
wordstring="Just a bunch of words to test for killer items containing a k"
   -- turn Rexx-string into a Java-string: this allows us to use Java's String methods
refWordString=.bsf~new("java.lang.String", wordstring)
   -- convert the string into a Java List (a Collection):
alist=bsf.loadClass("java.util.Arrays")~asList(refWordString~split(" "))

-- create a RexxProxy of our Worker class which implements the two functional interface
rexxWorker=BsfCreateRexxProxy(.worker~new, ,"java.util.function.Predicate", -
                                            "java.util.function.Consumer"     )

say "This predicate filter just selects words containing the letter 'k':"
sa=alist~stream~filter(rexxWorker)~toArray   -- "filter" employs the Predicate interface
loop w over sa    -- print the results for verification:
   say '['w']'
end
say "-"~copies(30)

alist~stream~foreach(rexxWorker)             -- "forEach" employs the Consumer interface

::requires BSF.CLS    -- get Java-support

::class Worker

   -- implements the interface java.util.function.Predicate
::method test      -- will return .true (1) for strings containing 'k', .false (0) else
  use arg s
  return s~caselessPos('k')>0

   -- implements the interface java.util.function.Consumer
::method accept    -- just shows each word
  use arg s
  say "["s"]"
```

Changes and New Features, 6
Lambda Example 1b

Output:

```
E:\201504-RexxLA\>rexx demo-j8-lambda.rex
This predicate filter just selects words containing the letter 'k':
[killer]
[k]
------------------------------
[Just]
[a]
[bunch]
[of]
[words]
[to]
[test]
[for]
[killer]
[items]
[containing]
[a]
[k]
```

Roundup and Outlook

- Roundup
  - Improved installation for Windows and MacOSX
  - New “ooRexxTry.rxj”
  - New “auto-attach” feature
  - Improved Java method resolution
- Outlook
  - Start work on JSR-223 interface for BSF4ooRexx
  - 64-bit Windows
    - Make sure that 32-bit Java support gets installed correctly