2012 RexxLA International Rexx Language Symposium Proceedings

René Vincent Jansen (ed.)

THE REXX LANGUAGE ASSOCIATION
RexxLA Symposium Proceedings Series
ISSN 1534-8954

Publication Data

A publication of **RexxLA Press**

The responsible publisher of this edition is identified as *IBizz IT Services and Consultancy*, Amsteldijk 14, 1074 HR Amsterdam, a registered company governed by the laws of the Kingdom of The Netherlands.

The RexxLA Symposium Series is registered under ISSN 1534-8954
The 2012 edition is registered under ISBN 978-94-037-2962-6

2023-05-31 First printing

Introduction

History of the International Rexx Language Symposium

In 1990, Cathie Dager of SLAC[1] convened the organizing committee for the first independent Rexx[2] Symposium for Developers and Users. SLAC continued to organize this annual event until the middle of the 1990's when the RexxLA took over that responsibility. Symposia have been held annually since 1990.

About RexxLA

During the 1993 Symposium in La Jolla, California, plans for a Rexx User Group materialized. The Rexx Language Association (RexxLA), as it was called, is an independent, non-profit organization dedicated to promoting the use and understanding of the Rexx programming language. RexxLA manages several open source implementations of Rexx.

The selection procedure

Presentation proposals are solicited yearly using a CFP[3] procedure, after which the RexxLA symposium comittee reviews them and votes which presentations are selected for the symposium. The presentations are peer reviewed before being presented. Presenters are not compensated for their presentations.

Location

The 2012 symposium was held in Raleigh-Durham area, North Carolina, USA from 13 May 2012 to 16 May 2012.

[1] Stanford Linear Accelerator Center, since 2008 SLAC National Accelerator Laboratory
[2] Cowlishaw, M. F., **The REXX Language** (second edition), ISBN 0-13-780651-5, Prentice-Hall, 1990.
[3] Call For Papers.

Contents

1

Configuring Rexx Interpreter Instances from NetRexx/Java – Rony G. Flatscher

Date and Time

14 May 2012, 13:30:00 CET

Presenter

Rony G. Flatscher

Presenter Details

Rony works as a professor for Business informatics ("Wirtschaftsinformatik") at the Vienna University of Economics and Business Administration (Wirtschaftsuniversität Wien) and uses Open Object Rexx for teaching Business Administration and MIS students the object-oriented paradigm, as well as remote-controlling (automating) Windows and Windows end-user applications (e.g. MS Office, Open Office) as well as Java and Java applications (he is the author of BSF4ooRexx, the ooRexx-Java bridge, which uses Apache BSF and had Rony invited to become an ASF member). He consults and trains in all of his research fields.

Configuring Rexx Interpreter Instances from NetRexx/Java

The 2012 International Rexx Symposium

Rony G. Flatscher

Wirtschaftsuniversität Wien ■ Augasse 2-6 ■ A-1090 Wien

Agenda

- ooRexx startup options
  - Overview
- New BSF4ooRexx 4.1 support for configuring Rexx interpreter instances from Java/NetRexx
  - Overview
  - Configuration in detail
  - Nutshell examples
- Roundup

ooRexx Startup Options, 1

- Overview
  - Multiple ooRexx interpreter instances per process!
    - Each instance can be invoked with different options from C++
    - Documentation in rexxpg.pdf, section "9.1. Rexx Interpreter API"
  - For each Rexx interpreter instance there is a separate .local
    - Eg. .input (.stdin), .output (.stdout), .error (.stderr) ...
  - .environment available to all interpreter instances
- Over all
  - For each Rexx interpreter instance there may be multiple threads executing Rexx programs concurrently!

ooRexx Startup Options, 2

- Options available in C++
  - APPLICATION_DATA
    - void * pointer to Rexx interpreter instance application data
  - EXTERNAL_CALL_PATH
    - String containing additional paths to search for Rexx programs
  - EXTERNAL_CALL_EXTENSIONS
    - String containing additional file extensions which Rexx programs may have, such that the interpreter should look for them as well
  - LOAD_REQUIRED_LIBRARY
    - Array of strings denoting libraries (DLLs, .so) to load

ooRexx Startup Options, 3

- Options available in C++ (continued)
  - REGISTER_LIBRARY
    - Allows to register external Rexx functions/methods implemented in C++ in the program starting the Rexx interpreter instance
  - DIRECT_EXITS
    - ooRexx 4.x version of Rexx exits (array of exit handlers)
  - DIRECT_ENVIRONMENTS
    - ooRexx 4.x version of Rexx environments (array of [sub]command handlers)
  - INITIAL_ADDRESS_ENVIRONMENT
    - String denoting the name of the default environment

ooRexx Startup Options, 4

- Options available in C++ (continued)
  - REGISTERED_EXITS
    - Legacy (SAA) Rexx exits (array of exit handlers)
  - REGISTERED_ENVIRONMENTS
    - Legacy (SAA) Rexx environments (array of [sub]command handlers)

New BSF4ooRexx 4.1 Support for Startup Options, Overview

- The Java RexxEngine represents an ooRexx interpreter instance and is managed by a BSFManager
- Initialization of the Rexx interpreter instance gets now deferred as long as possible
  - Allows configuring the Rexx interpreter instance
    - A default configuration matching previous BSF4ooRexx options
    - Configuration done via RexxConfiguration object of RexxEngine
  - Rexx interpreter instance gets created upon the first request from Java to execute Rexx code
    - No (re-)configuration possible anymore

New BSF4ooRexx 4.1 Support for Startup Options, RexxConfiguration, 1

- Options available in Java/NetRexx
  - EXTERNAL_CALL_PATH
  - EXTERNAL_CALL_EXTENSIONS
  - LOAD_REQUIRED_LIBRARY
  - DIRECT_EXITS
  - DIRECT_ENVIRONMENTS
  - INITIAL_ADDRESS_ENVIRONMENT
- Options *not* available to Java/NetRexx (not applicable)
  - APPLICATION_DATA, REGISTER_LIBRARY, REGISTERED_EXITS, REGISTERED_ENVIRONMENTS

New BSF4ooRexx 4.1 Support for Startup Options, RexxConfiguration, 2

- To get access to the RexxConfiguration
  - Create a RexxEngine instance
  - Use the public method getRexxConfiguration()
- Configuration methods available in RexxConfiguration
  - Rexx option EXTERNAL_CALL_PATH
    - The files and paths need to be separated by the operating system's conventions, use eg.

      `System.getProperty("file.separator")`
      - "/" on Unix, "\" on Windows

      `System.getProperty("path.separator")`
      - ":" on Unix, ";" on Windows

```
void setExternalCallPath(String path) throws BSFException
String getExternalCallPath()
```


New BSF4ooRexx 4.1 Support for Startup Options, RexxConfiguration, 3

  - Configuration methods available in RexxConfiguration
    - Rexx option EXTERNAL_CALL_EXTENSION
      - Additional file extensions, separated by a comma (,)

```
void setExternalCallExtension(String path) throws BSFException
String getExternalCallExtension()
```

    - Rexx option LOAD_REQUIRED_LIBRARY
      - One or more native libraries that are required for the Rexx programs

```
void addRequiredLibrary(String libraryName) throws BSFException
String[] getRequiredLibraries()
```

New BSF4ooRexx 4.1 Support for Startup Options, RexxConfiguration, 4

- Configuration methods available in RexxConfiguration
  - Rexx option DIRECT_EXITS
    - Exit numbers ("function") defined in interface RexxExitHandler
    - Defined exit handlers can be replaced at runtime!
      - Defined exits can be temporarily "nullified" at runtime!

```
void addExitHandler(int function, RexxExitHandler exitHandler)
      throws BSFException
RexxExitHandler setExitHandler(int function, RexxExitHandler
      exitHandler) throws BSFException
RexxExitHandler getExitHandler(int function)
BitSet getDefinedExits()
Object [] getExitHandlers()
      Object[] contains an int and a RexxExitHandler array object
```


New BSF4ooRexx 4.1 Support for Startup Options, RexxConfiguration, 5

- Configuration methods available in RexxConfiguration
  - Rexx option DIRECT_ENVIRONMENTS
    - Defined command handlers can be replaced at runtime!

```
void addCommandHandler(String name, RexxCommandHandler
      commandHandler) throws BSFException
RexxCommandHandler getCommandHandler(String name)
RexxCommandHandler setCommandHandler(String name,
      RexxCommandHandler commandHandler) throws BSFException
Object[] getCommandHandlers()
      Object[] contains a String and a RexxCommandHandler array object
```

New BSF4ooRexx 4.1 Support for Startup Options, RexxConfiguration, 6

- Configuration methods available in RexxConfiguration
  - Rexx option INITIAL_ADDRESS_ENVIRONMENT

```
void setInitialAddressEnvironment(String name) throws BSFException
String getInitialAddressEnvironment()
```


Nutshell Example "External Call Extension"

- Java program "SampleFileExtension.java"
  - Add file extension ".xyz" as another Rexx program extension
- Rexx programs
  - "testSampleFileExtension.rex"
  - "rexx_pgm.xyz"

Nutshell Example "External Call Extension " SampleFileExtension.java

```
import org.apache.bsf.*;
import org.rexxla.bsf.engines.rexx.*;

public class SampleFileExtension
{
   public static void main (String args[]) throws BSFException
   {
      BSFManager mgr      =new BSFManager();     // create an instance of BSFManager
      RexxEngine rexxEngine=(RexxEngine) mgr.loadScriptingEngine("rexx");  // load the Rexx engine

         // Configure the RexxEngine
      RexxConfiguration rexxconf=rexxEngine.getRexxConfiguration();
         // add ".xyz" to default call extensions
      rexxconf.setExternalCallExtensions(rexxconf.getExternalCallExtensions()+",.xyz");
      System.err.println("SampleFileExtension.java, Rexx configuration:\n\n"+rexxconf+"\n");

         // Rexx code to run (quote filename for Unix filesystems)
      String rexxCode= "call 'testSampleFileExtension.rex' ";
         // invoke the interpreter and run the Rexx program
      rexxEngine.apply ("SampleFileExtension.rex", 0, 0, rexxCode, null, null);
      rexxEngine.terminate();     // terminate Rexx engine (Rexx interpreter instance)
   }
}
```


Nutshell Example "External Call Extension " nrxSampleFileExtension.nrx

```
mgr          = org.apache.bsf.BSFManager()      -- create an instance of BSFManager
rexxEngine = org.rexxla.bsf.engines.rexx.RexxEngine mgr.loadScriptingEngine("rexx")  -- load the Rexx engine

   -- Rexx code to run (quote filename for Unix filesystems)
rexxCode= "call 'testSampleFileExtension.rex' "
   -- Configure the RexxEngine
rexxconf=rexxEngine.getRexxConfiguration()
   -- add ".xyz" to default call extensions
rexxconf.setExternalCallExtensions(rexxconf.getExternalCallExtensions()",.xyz")
System.err.println("nrxSampleFileExtension.nrx, Rexx configuration:\n\n"rexxconf"\n")

   -- invoke the interpreter and run the Rexx program
rexxEngine.apply("SampleFileExtension.rex", 0, 0, rexxCode, null, null)
rexxEngine.terminate()     -- terminate Rexx engine (Rexx interpreter instance)
```

Nutshell Example "External Call Extension" testSampleFileExtension.rex, rexx_pgm.xyz

```
/* testSampleFileExtension.rex */
parse source . . f
say "---> This is from" pp(filespec("name",f)) "<---"
say

say "Testing Rexx programs with arbitrary extension '.xyz' ..."
call "rexx_pgm.xyz"   -- quoted, such that it can be found on Unix too

say "--- now not supplying the extension '.xyz'"
call "rexx_pgm"       -- quoted, such that it can be found on Unix too
say "---> The end. <---"

::routine pp
  return "["arg(1)"]"
```

```
/* rexx_pgm.xyz */
parse source . . f
say
say "      ==> This is from" pp(filespec("name",f)) "<=="
say
exit

::routine pp
  return "["arg(1)"]"
```


Nutshell Example "External Call Extension" Running the Program

```
E:\extendFileExtension>java SampleFileExtension
SampleFileExtension.java, Rexx configuration:

org.rexxla.bsf.engines.rexx.RexxConfiguration[initialAddressEnvironment=[null],externalCallPath=[null],
externalCallExtensions=[.rxj,.rxo,.rxjo,.jrexx,.xyz],loadRequiredLibrary={},
exitHandlers={},commandHandlers={}]

---> This is from [testSampleFileExtension.rex] <---

Testing Rexx programs with arbitrary extension '.xyz' ...

     ==> This is from [rexx_pgm.xyz] <==

--- now not supplying the extension '.xyz'

     ==> This is from [rexx_pgm.xyz] <==

---> The end. <---
```

Nutshell Example "External Call Path"

- Java program "SamplePathExtension.java"
  - Add file extension ".xyz" as another Rexx program extension
  - Add path "./anotherpath" for the Rexx interpreter to look up
- Rexx programs
  - "testSamplePathExtension.rex"
  - "anotherpath/rexx_pgm.xyz"

Nutshell Example "External Call Path" SamplePathExtension.java

```
import org.apache.bsf.*;
import org.rexxla.bsf.engines.rexx.*;

public class SamplePathExtension
{
   public static void main (String args[]) throws BSFException
   {
      BSFManager mgr      =new BSFManager();     // create an instance of BSFManager
      RexxEngine rexxEngine=(RexxEngine) mgr.loadScriptingEngine("rexx");  // load the Rexx engine

         // Configure the RexxEngine
      RexxConfiguration rexxconf=rexxEngine.getRexxConfiguration();
         // add ".xyz" to default call extensions
      rexxconf.setExternalCallExtensions(rexxconf.getExternalCallExtensions()+",.xyz");
         // add ".\anotherpath" (Windows), "./anotherpath" (Unix)
      rexxconf.setExternalCallPath("."+System.getProperty("file.separator")+"anotherpath");
      System.err.println("SamplePathExtension.java, Rexx configuration:\n\n"+rexxconf+"\n");

         // Rexx code to run (quote filename for Unix filesystems)
      String rexxCode= "call 'testSamplePathExtension.rex' ";
         // invoke the interpreter and run the Rexx program
      rexxEngine.apply ("SamplePathExtension.rex", 0, 0, rexxCode, null, null);
      rexxEngine.terminate();     // terminate Rexx engine (Rexx interpreter instance)
   }
}
```


Nutshell Example "External Call Extension " nrxSamplePathExtension.nrx

```
mgr       = org.apache.bsf.BSFManager()      -- create an instance of BSFManager
rexxEngine = org.rexxla.bsf.engines.rexx.RexxEngine mgr.loadScriptingEngine("rexx")  -- load the Rexx engine

   -- Rexx code to run (quote filename for Unix filesystems)
rexxCode= "call 'testSamplePathExtension.rex' "
   -- Configure the RexxEngine
rexxconf=rexxEngine.getRexxConfiguration()
   -- add ".xyz" to default call extensions
rexxconf.setExternalCallExtensions(rexxconf.getExternalCallExtensions()",.xyz")
   -- add ".\anotherpath" (Windows), "./anotherpath" (Unix)
rexxconf.setExternalCallPath("."System.getProperty("file.separator")"anotherpath")

System.err.println("nrxSamplePathExtension.nrx, Rexx configuration:\n\n"rexxconf"\n")

   -- invoke the interpreter and run the Rexx program
rexxEngine.apply("SamplePathExtension.rex", 0, 0, rexxCode, null, null)
rexxEngine.terminate()     -- terminate Rexx engine (Rexx interpreter instance)
```


Nutshell Example "External Call Path" testSamplePathExtension.rex, rexx_pgm.xyz

```
/* testSamplePathExtension.rex */
parse source . . f
say "---> This is from" pp(filespec("name",f)) "<---"
say

say "Testing Rexx programs with arbitrary extension '.xyz' ..."
call "rexx_pgm.xyz"   -- quoted, such that it can be found on Unix too

say "--- now not supplying the extension '.xyz'"
call "rexx_pgm"       -- quoted, such that it can be found on Unix too
say "---> The end. <---"

::routine pp
  return "["arg(1)"]"
```

```
/* ./anotherpath/rexx_pgm.xyz */
parse source . . f
say
say "      ==> This is from" pp(filespec("name",f)) "<=="
say
exit

::routine pp
  return "["arg(1)"]"
```


