Institut für Betriebswirtschaftslehre und Wirtschaftsinformatik

Implementing Rexx Handlers in NetRexx / Java / Rexx

The 2012 International Rexx Symposium

Rony G. Flatscher

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

Agenda

- New BSF4ooRexx 4.1 support for configuring Rexx interpreter instances from Java/NetRexx
  - Setup options for command and exit handlers
- Writing Rexx (sub)command handlers
  - Nutshell example
- Writing Rexx exit handlers
  - Nutshell example
- Roundup

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**

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()
  - Use the RexxConfiguration methods for setting up exit and command handlers

New BSF4ooRexx 4.1 Support for Startup Options, RexxConfiguration, 3

- 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, 4

- 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, 5

- Configuration methods available in RexxConfiguration
  - Rexx option INITIAL_ADDRESS_ENVIRONMENT

```
void setInitialAddressEnvironment(String name) throws BSFException

String getInitialAddressEnvironment()
```

Rexx Exits and Rexx (Sub-)Command Handlers, 1

- "Callbacks" from the Rexx interpreter
- Rexx (sub-)command handler
  - Invoked by the Rexx interpreter whenever a command has to be carried out
    - A Java implemented Rexx command handler
      - Receives the ADDRESS name
      - Receives the command to carry out and
      - May return any value
        - A return value can be fetched by the Rexx program by referring to the Rexx variable "RC" (return code) upon return from the command handler

Rexx Exits and Rexx (Sub-)Command Handlers, 2

- "Callbacks" from the Rexx interpreter
- Rexx exits
  - Invoked before certain "important" actions are carried out by the Rexx interpreter, e.g.
    - Searching an external function
    - About to write a string to stdout
    - ...
  - A Rexx exit handler can choose to
    - Handle the exit itself (RexxExit.RXEXIT_HANDLED)
    - Let Rexx handle the exit (RexxExit.RXEXIT_**NOT**_HANDLED)
    - Raise an error (RexxExit.RXEXIT_RAISE_ERROR)

org.rexxla.bsf.engines.rexx.RexxHandler
A Utility Class for Java Rexx Handlers

- Java abstract class org.rexxla.bsf.engines.rexx.RexxHandler
  - Must only be used in the same thread that invokes Java Rexx exit and Rexx command handlers!
  - Defines and implements static methods
    - Makes some of the ooRexx C++ APIs available to the Java Rexx handlers
      - Fetch and set Rexx variables
      - Set thread and halt conditions
      - Raise Rexx conditions and Rexx exceptions
      - Get access to the Rexx .local, global .environment and .nil objects
    - Must supply the unaltered (opaque) argument "slot" that gets passed to Java Rexx exit and Rexx command handlers as the first argument

Writing a RexxCommandHandler

- Java interface
  org.rexxla.bsf.engines.rexx.RexxCommandHandler
- Defines a single method

  Object handleCommand(Object slot, String address, String command)

  – "slot"
    - Opaque value for using the static RexxHandler methods

  – "address"
    - Name of the addressed environment

  – "command"
    - The command to carry out

  – Return value
    - Any value, can be retrieved in Rexx via the variable named "RC" (return code)
    - Returning a Java null will be mapped to the return code number "0"

Writing a RexxCommandHandler
An Example

- Command handler name for the Rexx ADDRESS keyword statement: "TEST1"
- Commands
  - "Hi", should return the string "Hi, but who are you?"
  - "one plus two", should return the number "3"
  - "please panic a little bit", should raise a Rexx exception
  - Any other command should return the string "undefined command # xyz: [command]"
    - where "xyz" represents the current value of an appropriate counter
    - where "command" shows the undefined command

Writing a RexxCommandHandler
The Java Rexx Command Handler, 1

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

public class SampleCommandHandler implements RexxCommandHandler
{
   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 command handler
      rexxconf.addCommandHandler("TEST1", new SampleCommandHandler());
      System.err.println("edited rexxconf=["+rexxconf+"]\n");
      // Rexx code to run
      String rexxCode= "call 'testSampleCommandHandler.rex' ";
      // invoke the interpreter and run the Rexx program
      rexxEngine.apply ("SampleCommandHandler.rex", 0, 0, rexxCode, null, null);
      rexxEngine.terminate();     // terminate Rexx interpreter instance
   }

   ... continued ...
```

Writing a RexxCommandHandler
The Java Rexx Command Handler, 2

... continued ...

```
    int counter=0;  // count # of undefined commands
    public Object handleCommand(Object slot, String address, String command)
    {
        System.err.println("address=["+address+"], command=["+command+"]");
        if (command.compareToIgnoreCase("Hi")==0) {return "Hi, who are you?";}
        else if (command.compareToIgnoreCase("one plus two")==0) {return "3";}
        else if (command.compareToIgnoreCase("please panic a little bit")==0)
        {
            RexxHandler.raiseException1(slot, 35900, this+": You asked for this exception!");
            return null;
        }
            // undefined command
        counter++;
        return "Undefined command # "+counter+": ["+command+"]";
    }
}
```

Writing a RexxCommandHandler
The NetRexx Rexx Command Handler, 1

```
class nrxSampleCommandHandler public implements org.rexxla.bsf.engines.rexx.RexxCommandHandler

properties private
   counter=0   -- count # of undefined commands

method main(s=String[]) static
   mgr       = org.apache.bsf.BSFManager()      -- create an instance of BSFManager
   rexxEngine = org.rexxla.bsf.engines.rexx.RexxEngine mgr.loadScriptingEngine("rexx")

      -- Configure the RexxEngine
   rexxconf=rexxEngine.getRexxConfiguration()
      -- add command handler
   rexxconf.addCommandHandler("TEST1", nrxSampleCommandHandler())
   say "nrxSampleCommandHandler.nrx, edited rexxonf=["rexxconf"]\n"

      -- Rexx code to run (quote filename for Unix filesystems)
   rexxCode= "call 'testSampleCommandHandler.rex' "
      -- invoke the interpreter and run the Rexx program
   rexxEngine.apply("nrxSampleCommandHandler.rex", 0, 0, rexxCode, null, null)
   rexxEngine.terminate()     -- terminate Rexx engine (Rexx interpreter instance)
```

... continued ...

Writing a RexxCommandHandler
The NetRexx Rexx Command Handler, 2

```
... continued ...

method handleCommand(slot=Object, address=String, command=String) returns Object
   say "address=["address"], command=["command"]"
   if command="Hi" then return "Hi, who are you?"
   else if command="one plus two" then return String("3")
   else if command="please panic a little bit" then
   do
      org.rexxla.bsf.engines.rexx.RexxHandler.raiseException1(slot, 35900, this": You asked for this exception!")
      return null
   end

      -- undefined command
   counter=counter+1
   return "Undefined command #" counter": ["command"]"
```

Writing a RexxCommandHandler
The Rexx Program

```
address test1 "hi"
say "rc="pp2(rc)
say

address test1  -- change address permanently
one   plus      two
say "rc="pp2(rc)
say

call testException

"nothing to do?"
say "rc="pp2(rc)
say

::requires "rgf_util2.rex" -- get public routines pp2(), ppCondition2()

::routine testException -- send the command that raises an exception
  signal on any
  address test1 "please panic a little bit"
  return
any:
  say ppCondition2(condition('Object'))
  say
  return
```

Writing a RexxCommandHandler
The Rexx Program's Output

```
E:\commandHandler>java SampleCommandHandler
edited rexxconf=[org.rexxla.bsf.engines.rexx.RexxConfiguration[initialAddressEnvironment=[null],
externalCallPath=[null],externalCallExtensions=[.rxj,.rxo,.rxjo,.jrexx],loadRequiredLibrary={},
exitHandlers={},commandHandlers={TEST1=SampleCommandHandler@4741d6}]]

address=[TEST1], command=[hi]
rc=[Hi, who are you?]

address=[TEST1], command=[ONE PLUS TWO]
rc=[3]

address=[TEST1], command=[please panic a little bit]
        [ADDITIONAL] =[an Array (1 items) id#_266374451]
                [SampleCommandHandler@4741d6: You asked for this exception!]
        [CODE]       =[35.900]
        [CONDITION]  =[SYNTAX]
        [DESCRIPTION]=[]
        [ERRORTEXT]  =[Invalid expression]
        [INSTRUCTION]=[SIGNAL]
        [MESSAGE]    =[SampleCommandHandler@4741d6: You asked for this exception!]
        [PACKAGE]    =[a Package id#_266374542]
        [POSITION]   =[20]
        [PROGRAM]    =[E:\commandHandler\testSampleCommandHandler.rex]
        [PROPAGATED] =[1]
        [RC]         =[35]
        [TRACEBACK]  =[a List (0 items) id#_266374526]

address=[TEST1], command=[nothing to do?]
rc=[Undefined command # 1: [nothing to do?]]
```

Writing a RexxCommandHandler
Teaser: The Rexx ☺ Rexx Command Handler, 1

```
-- prepare another Rexx interpreter instance besides the current one for this Rexx program
rexxEngine=.bsf~new("org.apache.bsf.BSFManager")~loadScriptingEngine("rexx")
rexxconf=rexxEngine~getRexxConfiguration
proxy=BsfCreateRexxProxy(.sampleCommandHandler~new, ,"org.rexxla.bsf.engines.rexx.RexxCommandHandler")
rexxconf~addCommandHandler("TEST1", proxy)
say "rexSampleCommandHandler.rxj, edited rexxconf="pp(rexxconf~toString)
say

-- invoke the interpreter and run the Rexx program
rexxCode= "call 'testSampleCommandHandler.rex' "
rexxEngine~apply("from_rexSampleCommandHandler.rxj", 0, 0, rexxCode, .nil, .nil)
rexxEngine~terminate       -- terminate Rexx engine (Rexx interpreter instance)


::requires BSF.CLS   -- get Java support

... continued ...
```

Writing a RexxCommandHandler
Teaser: The Rexx ☺ Rexx Command Handler, 2

```
... continued ...

::class SampleCommandHandler  -- Rexx class implementing the "handleCommand" interface

::method init        -- needed to define the "counter" attribute and set it to "0"
  expose counter
  counter=0

::method handleCommand  -- Rexx command handler implemented in Rexx!
  expose counter
  use arg slot, address, command

  say "address=["address"], command=["command"]"
  if command~caselessEquals("Hi") then return "Hi, who are you?"
  else if command~caselessEquals("one plus two") then return 3
  else if command~caselessEquals("please panic a little bit") then
  do
    raise syntax 35.900 array (self": You asked for this exception!")
  end

     -- undefined command
  counter=counter+1
  return "Undefined command #" counter": ["command"]"
```

Writing a RexxExitHandler

- Java interface
  org.rexxla.bsf.engines.rexx.RexxExitHandler
- Defines a single method

  int handleExit(Object slot, int exitNumber, int subFunction, Object[] parmBlock)

  – "slot"
    - Opaque value for using the static RexxHandler methods

  – "exitNumber"
    - The exit number, cf. the Rexx documentation in "rexxpg.pdf", "9.12.2. Context Exit Definitions"

  – "subFunction"
    - The exit's subfunction number, cf. the Rexx documentation in "rexxpg.pdf"

  – "parmBlock"
    - A Java array representing the C "parmBlock" structure, depends on the exit and subfunction

  – Return value
    - 0 (RXEXIT_HANDLED), 1 (RXEXIT_NOT_HANDLED), -1 (RXEXIT_RAISE_ERROR)

Writing a RexxExitHandler
An Example

- Exit number 14 (RXVALUE)
  - Invoked every time the VALUE()-BIF gets invoked with an unknown selector value
- Handles only invocations, if selector="RGF"
- "parmBlock" array (cf. JavaDocs for full documentation)
  - First entry: selector (a string)
  - Second entry: variable name (a string)
  - Third entry: value

Writing a RexxExitHandler
The Java Rexx Exit Handler, 1

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

public class SampleExitHandler implements RexxExitHandler
{
   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

      // Rexx code to run
      String rexxCode= "call 'testSampleExitHandler.rex' " ;

      // Configure the RexxEngine
      RexxConfiguration rexxconf=rexxEngine.getRexxConfiguration();
      System.err.println("default rexxconf=["+rexxconf+"]\n");

         // add system exits
      rexxconf.addExitHandler(RexxExitHandler.RXVALUE, new SampleExitHandler() );
      System.err.println("edited  rexxconf=["+rexxconf+"]\n");

      // invoke the interpreter and run the Rexx program
      rexxEngine.apply ("SampleExitHandler.rex", 0, 0, rexxCode, null, null);
      rexxEngine.terminate();     // terminate Rexx engine instance
   }


   ... continued ...
```

Writing a RexxExitHandler
The Java Rexx Exit Handler, 2

... continued ...

```
    // implementation of a RXVALUE exit handler, Java arrays are 0-based
  public int handleExit(Object slot, int exitNumber, int subFunction, Object[] parmBlock)
  {
    System.err.println("(Java side) -> selector=["+parmBlock[0]+"], varName=["+parmBlock[1]+"]"+
                       ", value=["+parmBlock[2]+"]");

    String selector=(String) parmBlock[0];
    if (selector.compareToIgnoreCase("RGF")==0) // o.k., addressed to us, handle it
    {
      if (parmBlock[2]==null) // if value is null, give it some value
      {
        parmBlock[2]="value for variable name ["+parmBlock[1]+"] by a Java Rexx exit handler";
      }
      return RexxExitHandler.RXEXIT_HANDLED;
    }
    return RexxExitHandler.RXEXIT_NOT_HANDLED;
  }
}
```

Writing a RexxExitHandler
The NetRexx Rexx Exit Handler, 1

```
import org.rexxla.bsf.engines.rexx.RexxExitHandler

class nrxSampleExitHandler public implements RexxExitHandler

method main(s=String[]) static
   mgr       = org.apache.bsf.BSFManager()      -- create an instance of BSFManager
   rexxEngine = org.rexxla.bsf.engines.rexx.RexxEngine mgr.loadScriptingEngine("rexx")

      -- Configure the RexxEngine
   rexxconf=rexxEngine.getRexxConfiguration()
      -- add exit handler
   rexxconf.addExitHandler(RexxExitHandler.RXVALUE, nrxSampleExitHandler() );
   say "nrxSampleExitHandler.nrx, edited rexxonf=["rexxconf"]\n"

      -- Rexx code to run (quote filename for Unix filesystems)
   rexxCode= "call 'testSampleExitHandler.rex' "
      -- invoke the interpreter and run the Rexx program
   rexxEngine.apply("nrxSampleExitHandler.rex", 0, 0, rexxCode, null, null)
   rexxEngine.terminate()     -- terminate Rexx engine (Rexx interpreter instance)

... continued ...
```

Writing a RexxExitHandler
The NetRexx Rexx Exit Handler, 2

... continued ...

```
    // implementation of a RXVALUE exit handler, NetRexx (Java) arrays are 0-based
method handleExit(slot=Object, exitNumber=int, subFunction=int, parmBlock=Object[]) returns int
   if parmBlock[2]=null then
      say "(NetRexx side) -> selector=["parmBlock[0]"], varName=["parmBlock[1]"]"
   else
      say "(NetRexx side) -> selector=["parmBlock[0]"], varName=["parmBlock[1]"], value=["parmBlock[2]"]"

   selector=String parmBlock[0]
   if selector="RGF" then      -- o.k., addressed to us, handle it
   do
      if (parmBlock[2]==null) then    -- if value is null, give it some value
      do
         parmBlock[2]="value for variable name ["parmBlock[1]"] by a NetRexx Rexx exit handler"
      end
      return RexxExitHandler.RXEXIT_HANDLED
   end
   return RexxExitHandler.RXEXIT_NOT_HANDLED
```

Writing a RexxExitHandler
The Rexx Program

```
say "(Rexx side) value('abc', , 'RGF')              ="pp(value('abc', , 'RGF'))
say

say "(Rexx side) value('def', , 'rGf')              ="pp(value('def', , 'rGf'))
say

say "(Rexx side) value('ghi', 'na,sowas!', 'RGF')="pp(value('ghi', 'na,sowas!', 'RGF'))

::requires "BSF.CLS"    -- get access to public routine pp()
```

Writing a RexxExitHandler
The Rexx Program's Output

```
E:\exitHandler>java SampleExitHandler
default rexxconf=[org.rexxla.bsf.engines.rexx.RexxConfiguration[initialAddressEnvironment=[null],
externalCallPath=[null],externalCallExtensions=[.rxj,.rxo,.rxjo,.jrexx],loadRequiredLibrary={},
exitHandlers={},commandHandlers={}]]

edited  rexxconf=[org.rexxla.bsf.engines.rexx.RexxConfiguration[initialAddressEnvironment=[null],
externalCallPath=[null],externalCallExtensions=[.rxj,.rxo,.rxjo,.jrexx],loadRequiredLibrary={},
exitHandlers={RXVALUE/14/SampleExitHandler@4741d6},commandHandlers={}]]

(Java side) -> selector=[RGF], varName=[abc], value=[null]
(Rexx side) value('abc',,'RGF')           =[value for variable name [abc] by a Java Rexx exit handler]

(Java side) -> selector=[rGf], varName=[def], value=[null]
(Rexx side) value('def',,'rGf')           =[value for variable name [def] by a Java Rexx exit handler]

(Java side) -> selector=[RGF], varName=[ghi], value=[na,sowas!]
(Rexx side) value('ghi','na,sowas!','RGF')=[na,sowas!]
```

Writing a RexxCommandHandler
Teaser: The Rexx ☺ Rexx Exit Handler, 1

```
 -- prepare another Rexx interpreter instance besides the current one for this Rexx program
clzName="org.rexxla.bsf.engines.rexx.RexxExitHandler"
clz=bsf.loadClass(clzName)
.local~RXEXIT_HANDLED    =clz~RXEXIT_HANDLED
.local~RXEXIT_NOT_HANDLED=clz~RXEXIT_NOT_HANDLED

rexxEngine=.bsf~new("org.apache.bsf.BSFManager")~loadScriptingEngine("rexx")
rexxconf=rexxEngine~getRexxConfiguration
proxy=BsfCreateRexxProxy(.sampleExitHandler~new, ,clzName)
rexxconf~addExitHandler(clz~RXVALUE, proxy)
say "rexSampleExitHandler.rxj, edited rexxconf="pp(rexxconf~toString)
say

-- invoke the interpreter and run the Rexx program
rexxCode= "call 'testSampleExitHandler.rex' "
rexxEngine~apply("from_rexSampleExitHandler.rxj", 0, 0, rexxCode, .nil, .nil)
rexxEngine~terminate       -- terminate Rexx engine (Rexx interpreter instance)


::requires BSF.CLS   -- get Java support

... continued ...
```

Writing a RexxCommandHandler
Teaser: The Rexx ☺ Rexx Exit Handler, 2

```
... continued ...

::class SampleExitHandler  -- Rexx class implementing the "handleExit" interface

    // implementation of a RXVALUE exit handler, Java array camouflaged as a Rexx array, hence 1-based !
::method handleExit
  use arg slot, exitNumber, subFunction, parmBlock

  say "(RexxExitHandler side) -> selector=["parmBlock[1]"], varName=["parmBlock[2]"], value=["parmBlock[3]"]"

  if parmBlock[1]~caselessEquals("RGF") then    -- o.k., addressed to us, handle it
  do
     if (parmBlock[3]=.nil) then  -- if value is null, give it some value
        parmBlock[3]="value for variable name ["parmBlock[2]"] by a Java Rexx exit handler"

     return .RXEXIT_HANDLED
  end

  return .RXEXIT_NOT_HANDLED
```

Roundup

- New BSF4ooRexx 4.1
  - Adds the ability for Java/NetRexx programs to
    - Define (sub-)command and exit handlers for a Rexx interpreter instance
    - Configuration from Java/NetRexx is very easy using the RexxConfiguration class
    - Implementing Java or NetRexx or Rexx (!) (sub-)command handlers is very easy
    - Implementing Java or NetRexx or Rexx (!) exit handlers is very easy
  - Allows any Java/NetRexx application to easily adopt Rexx as its scripting language and exploit every facet of it for its purposes!