NetRexx Scripting for Java Applications
http://www2.hursley.ibm.com/netrexxx/

Mike Cowlishaw
IBM Fellow
mfc@uk.ibm.com
Overview

- Introduction to NetRexx
- Example -- minor classes
- Using the compiler-interpreter
- Example -- scripting
- Questions?
What is NetRexx?

- A complete *alternative* to the Java language, for writing classes for the JVM
- Based on the simple syntax of Rexx, with Rexx decimal arithmetic
- Fully exploits the Java object model, exceptions, and binary arithmetic
- Automates type selection & declaration
- Removes many historical quirks
NetRexx Java implementation

- Current implementation first *translates* NetRexx to accessible Java source or *interprets* directly (or both).
- Runs on any Java platform
- Any class written in Java can be used
  - GUI, TCP/IP, I/O, Database, *etc.*
- Anything you could write in Java can be written in NetRexx
NetRexx programs

toast.nrx

/* This wishes you good health. */
say 'Cheers!'
Control constructs

if answer='yes' then say 'OK!'
else say 'shucks'

loop i=0 for mystring.length
    say i':' mystring[i]
end i

also do..end for simple grouping
select label choice  
  when name='Kewl' then say 'Cool?'
  when back.color=Color.red then say 'Hot'
  otherwise say '<sigh>'
end choice

select case i+1  
  when 1, 2, 3 then say 'some'
  otherwise say 'many'
end
Select and automatic switch{ }

- For example ...

```plaintext
select case i+1
    when 1 then say 'one'
    when 2 then say 'two'
    when 3 then say 'three'
    otherwise say 'uh?'
end
```
Select and automatic switch{

Can generate...

```
switch(i+1)
{
    case 1: RexxIO.Say("one"); break;
    case 2: RexxIO.Say("two"); break;
    case 3: RexxIO.Say("three"); break;
    default: RexxIO.Say("uh?");
}
```
Strings - the base type

- Strings in NetRexx are of type *Rexx*
  - by default, data and numbers are strings
  - standard methods from Object Rexx
  - conversions

- Automatic inter-conversion with Java String class, char and char[ ] arrays, and numeric primitives (optional)
Arithmetic

- Preferred arithmetic is from ANSI X3.274
- Decimal, just one type of number
  - follows human rules (2 * 1.20 is 2.40)
  - gives exact results when expected (e.g., for 0.1, 0.3, 0.9/10)
- no overflow at binary boundaries
- arbitrary precision

numeric digits 300
say 1/7
Standard Decimal Arithmetic

See:

http://www2.hursley.ibm.com/decimal
Binary classes and methods

- The **binary** keyword instructs the compiler to use native (binary) arithmetic types and operations (boolean, byte, int, long, float, etc.)

- Achieves the full speed of the Java Virtual Machine

- No performance penalty for using NetRexx instead of Java
Explicit typing

- Casting/conversions use the \textit{blank} (concatenation) \textit{operator}
  
  \begin{verbatim}
  number=int 7*y  -- number is an int
  number2=int     -- variable declaration
  \end{verbatim}

- Consistently extends to method arguments

  \begin{verbatim}
  method size(x=int, y=int, depth=int 3)
  \end{verbatim}
Other features from Rexx

- Case-insensitivity
- Parse
- Trace (methods, all, results)

2 ** number=1/7
   >v> number "0.142857143"
3 ** parse number before '.' after
   >v> before "0"
   >v> after "142857143"
4 ** say after'. 'before
   >>> "142857143.0"
Exceptions

- Semantics from Java
- Generalized and simplified syntax

```
say 'Please enter a number:'
number=ask       -- read a line
do
    say 'reciprocal is:' 1/number
catch Exception
    say 'Sorry, could not divide-'""'number'"" into 1'
end
```
NetRexx JavaBean support

- JavaBean (indirect) properties

```netrexx
properties indirect
  filling=Color.red

generates (or checks):

method getFilling returns java.awt.Color
  return filling
method setFilling($1=java.awt.Color)
  filling=$1
```
NetRexx Inner Class support

- Minor and Dependent classes

```plaintext
class Foo
    x=Bar()  
    y=Foo.Bar null
    z='Hello'
    x.Counter

class Foo.Bar dependent extends AnOther
    method Counter
        say parent.z
```
Buttons.nrx

(Softcopy available at the NetRexx web site.)

/* Buttons.nrx -- a window with two buttons */
class Buttons adapter extends Frame-
            implements WindowListener, ComponentListener

properties shared
    shadow=Image              -- offscreen image
properties constant
    mywidth=200               -- our shape
    myheight=300              -- ..
    glass=Toolkit.getDefaultToolkit().getScreenSize
/* Main method; called when started */
method main(s=String[]) static
    frame=Buttons("My Buttons" Rexx(s))  -- make a frame
    -- now size and place it mid-screen
    frame.setBounds((glass.width-mywidth)%2,-
                     (glass.height-myheight)%2, mywidth, myheight)
    frame.show  -- and make it visible
*/

/* The constructor for Buttons */
method Buttons(s=String)
    super(s)  -- title to superclass
    setLayout(FlowLayout())  -- set layout scheme
    add(Buttons.Left())  -- add one button ..
    add(Buttons.Right())  -- .. and the other
    addWindowListener(this)  -- listen to Window event
    addComponentListener(this)  -- and component events
/* newimage -- make a new offscreen image */
method newimage
    shadow=createImage(getSize.width, getSize.height)

/*/ componentResized -- called when graphics resized */
method componentResized(e=ComponentEvent)
    newimage         -- make new sized image

/*/ update & paint -- called when window is updated */
method update(g=Graphics)         -- avoid flicker
    paint(g)
method paint(g=Graphics)
    if shadow=null then newimage   -- ensure an image
        g.drawImage(shadow, 0, 0, 0, this)-- copy to screen

/*/ windowClosing -- called when window is closed */
-- We need to handle this to end the program
method windowClosing(e=WindowEvent)
    exit
/* A dependent class for a button */
class Buttons.Left dependent extends Button-
   implements ActionListener

method Left  -- construct the button
   super("Green")  -- we choose the label
   addActionListener(this)  -- listen for actions

method actionPerformed(a=ActionEvent)  -- pressed
   g=parent.shadow.getGraphics  -- get the image
   g.setColor(Color.green)  -- choose a colour
   -- now colour the image
   g.fillRect(0, 0, parent.getSize.width,-
             parent.getSize.height)
   parent.repaint  -- and request redraw
/ A dependent class for a button */
class Buttons.Right dependent extends Button-
    implements ActionListener

method Right
    super("Red")
    addActionListener(this)

method actionPerformed(a=ActionEvent) -- pressed
    g=parent.shadow.getGraphics  -- get the image
    g.setColor(Color.red)        -- choose a colour
    -- now colour the image
    g.fillRect(0, 0, parent.getSize.width,-
               parent.getSize.height)
    parent.repaint       -- and request redraw
Using NetRexxC

- Typical wrapper scripts (nrc.rex, nrc.bat) included in package
- Many options (most also specifiable in program)
- Demonstration ...
The procedure for interpreting a NetRexx script from Java or NetRexx is extremely simple:

- make an interpreter (once only)
- ask the interpreter to parse the script's source file
- get the resulting Class object (stub)
- create real instances, invoke method(s), etc., using the usual Java reflection API
The NetRexxA API

- NetRexxA() -- builds an interpreter object
- parse(files=String[], flags=String[]) returns boolean
- getClassObject(package=String, name=String) returns Class
  (add dimension for an array class)
options binary
import COM.ibm.netrexx.process.NetRexx

interpreter=NetRexxA() -- make interpreter

files=['hello.nrx'] -- a file to interpret
flags=['nocrossref', 'verbose0'] -- flags
interpreter.parse(files, flags) -- parse

helloClass=interpreter.getClassObject(null,['hello']) -- find the hello Class
Using the API  [2]

-- find the 'main' method
classes=[interpreter.getClassObject('java.lang',
    'String', 1)]
mainMethod=helloClass.getMethod('main', classes)

-- now invoke it, with a null instance (it's
-- static) and an empty String array (values)

values=[Object String[0]]
loop for 10    -- let's call it ten times...
    mainMethod.invoke(null, values)
end
Summary

- A blend of Rexx and Java
  - scripting *and* application development
  - a truly general-purpose language
- Both decimal and binary arithmetic
- High productivity and simplicity
  - Java source for a typical class has 35% more tokens than NetRexx
- Designed for *users*, not compilers.
Questions?

... Please fill in your evaluation form!
http://www2.hursley.ibm.com/netrexx/

NetRexx

Rexx + Java

Strong typing doesn’t need extra typing