RexxLA

NetRexx internals

by Marc Remes

remesm@gmail.com

Agenda

- What's new in NetRexx 5.01
- In depth
  - The translator
    - Components
    - Building it
    - Source files
    - Package
  - Extending the language
    - The new INTERPRET instruction

What's new

- New INTERPRET instruction
- Default JDK 1.8 class generation (class file format 52)
- New -targetvm option allows higher class files version generation
- -crossref option lists classes, methods and properties
- More complete JSR223 javax.scripting framework (using jsr223 engine reader and writer)
- More complete classic Rexx stream support (charin, charout, chars, linein, lineout, lines, stream)

What's new

- Fixes

NetRexx-21 superfluous warning on crossref option ("overrides program default")
NetRexx-57 Spurious E output removed in SPECS stage with FS specified
NetRexx-59 Java 9 and later disallow _ (underscore) as variable name, Java compiler fails
(_ allowed again but Java sees $_)
NetRexx-60 Javafx jar download location changed in readme. EnzoLCD example fixed.
NetRexx-62 emoji escape code is incorrect
NetRexx-63 parsing special words and variables
NetRexx-66 pipc does not honour file name extension and possibly removes existing .nrx files
NetRexx-67 exit code is not propagated when interpreting
NetRexx-68 Pipelines JVM 23 removed ThreadGroup.stop() fixed
NetRexx-72 Pipelines diskr '<' wrongly constructs absolute path filename
NetRexx-73 Emacs 30.1 does not colour fonts; fixed in netrexx-mode.el and still good for earlier Emacs

What's new

- New examples
  - xref2uml.nrx, generate PlantUML inputfiles from crossref files
  - rexxtry.nrx, every rexx needs its rexxtry
  - rexxcps.nrx, corrected calculating clauses per second
  - jsrbindings.nrx, retrieve bindings key values

The translator

- What it is
- NetRexxC.jar
  - core NetRexx classes
  - the translator, compiler and interpreter
  - njpipes, NetRexx implementation of CMS Pipelines,
  - nrws, NetRexx workspace
  - NetRexx JSR223 Scripting Engine
- NetRexxF.jar
  - all the above
  - modified eclipse java compiler (based on Java 15, emits default 1.8 class version files)
  - when no JDK is available
- NetRexxR.jar
  - core NetRexx classes only
  - when only NetRexx classes are used, plain Java, no NetRexx instructions
- Compiles itself, written in NetRexx

The translator

- What it is not
  - Does not 'run Java bytecode'
- Either it translates NetRexx source code into Java source code, and compiles it
- Or it interprets NetRexx source code with its own scheduler (RxInterpreter.nrx), and is able to handle Java bytecode objects when interfacing with native Java, eg.
  - call a Java method
  - be called by a Java method
  - access Java fields on Java classes or class instances
  - provide access to properties on NetRexx classes for native Java

The repository

- Git repo
  - distributed version control system
- Hosted on https://sourceforge.net/
  - URL https://sourceforge.net/projects/netrexx/
  - Clone locally with 'git clone https://git.code.sf.net/p/netrexx/code netrexx-code'
- Duplicated to https://github.com/
  - Hosts 'issue' tickets

The repository code structure

- ./src holds all source code
  - ./src/netrexx/lang
    - Core NetRexx classes
  - ./src/org/netrexx/process
    - The translator, interpreter and compiler
  - ./src/org/netrexx/njpipes
    - The NetRexx CMS Pipelines implementation
  - ./src/org/netrexx/jsr223
    - The NetRexx javax.script engine implementation
  - ./src/org/netrexx/diag
    - Diagnostics code testing validity of the built code
- /test
  - Additional test artifacts

The repository code structure

- ./ant contains some jar files supporting the build
  - ant-launcher.jar
    - java -jar ant/ant-launcher.jar when ant command absent
  - ant.jar
    - ant task used to build the translator
    - source available in ./tools/ant-task
  - ecj-I20201218-1800-NRX-18.jar
    - Modified eclipse Java compiler :
      - java source not needed
      - minimum and default class version 52 (Java 1.8)
    - Used to build the translator
    - Included in NetRexxF.jar and used with -ecj
  - jansi-1.17.jar and jline-3.13.1-SNAPSHOT.jar
    - Used in NetRexx Workspace nrws, a playground to explore NetRexx and Pipelines
  - json-20240303.jar
    - The reference implementation for the Javascript Object Notation data format

The repository code structure

- ./bin contains shorthand CLI commands to manage NetRexx
  - NetRexxC.sh, bat and .cmd
    - Wrapper to invoke java org.netrexx.process.NetRexxC
  - nrc, nrc.bat and nrc.cmd
    - Wrapper to invoke NetRexxC.sh, .bat, .cmd
  - nr
    - Invoke nrc -verbose0 -noconsole $1 -arg $2 $3 ..
    - Used as shebang
      - #!/usr/bin/env nr
    - Interprets executable .nrx files in linux/macOS
  - nrws, nrws.bat
    - Start the NetRexx Workspace
  - pipe, pipe.bat
    - Start the NetRexx Pipelines processor
  - pipc, pipc.bat
    - Start the NetRexx Pipelines compiler
  - hello.nrx
    - Sample

The repository code structure

- ./documentation contains the different publications sources
  - In LaTeX typesetting
  - Added functionality per RexxLA TextTools build.rexx routine
  - ./nrl, the NetRexx Language Definition
  - ./pg, the Programming guide
  - ./njpipes, the Pipelines Guide and Reference
  - Others
    - Supporting common include files
  - Documentation is built from the ./*/tex/book directory by 'rexx TextTools/build.rexx'

The repository code structure

- ./examples contains a significant amount of example source code to explore
  - ./rosettacode
  - ./uml contains UML diagrams of NetRexx code
  - ./ibm-historic
  - ./javaparser
- ./lib contains the latest shipped NetRexx jar files
- ./META-INF contains jar service definition meta information, services only
  - javax.script.ScriptEngineFactory
- ./tools contains some useful NetRexx tools
  - the ant build task, used during build
  - java2nrx.nrx, translate java source code to NetRexx
  - editor config files for support of NetRexx

Building the translator

- Based on ant, Apache's Another Neat Tool
  - https://ant.apache.org/
- build.xml is default configuration file
  - Defines targets and tasks
- Build results are created in ./build
  - ./build/classes
    - .nrx files are copied here (in their directory structure)
    - .java and .class files are generated here
  - ./build/lib
    - Newly built NetRexx?.jar files are created here

Building the translator

```
Buildfile: netrexx-code/build.xml

Main targets:

 apidocs           create API documentation
 clean             delete all built files
 clean.jar         delete built jars
 clean.javadocs    delete built javadocs
 clean.process     delete built translator files
 clean.runtime     delete built runtime files
 clean.tests       delete test files
 compile           compile all (except tests)
 compile.process   compile translator
 compile.runtime   compile runtime
 compile.tests     compile tests
 default           build and test distribution
 init              Set build number and document version level
 jars              create jars
 package           build distribution package
 post.jar.prepare  post jar build - define new NetRexx compiler
 setecj            set compiler to ecj
 setjavac          set compiler to javac
 showprops         Displays default property settings
 tests             compile and run tests
 withecj           build and test distribution with ecj
 withjavac         build and test distribution with javac
 withjavadocs      build distribution and javadocs with test
Default target: default
```

Building the translator

Properties defined in the build.xml, configuring Ant NetRexx build task

```
<property name="ant.netrexxc.binary"                        value="yes"/>
<property name="ant.netrexxc.comments"                      value="true"/>
<property name="ant.netrexxc.compact"                       value="true"/>
<property name="ant.netrexxc.crossref"                      value="no"/>
<property name="ant.netrexxc.format"                        value="true"/>
<property name="ant.netrexxc.keep"                          value="true"/>
<property name="ant.netrexxc.keepasjava"                    value="true"/>
<property name="ant.netrexxc.logo"                          value="true"/>
<property name="ant.netrexxc.removeKeepExtension"           value="no"/>
<property name="ant.netrexxc.replace"                       value="yes"/>
<property name="ant.netrexxc.strictargs"                    value="yes"/>
<property name="ant.netrexxc.strictcase"                    value="yes"/>
<property name="ant.netrexxc.strictsignal"                  value="yes"/>
<property name="ant.netrexxc.suppressDeprecation"           value="true"/>
<property name="ant.netrexxc.suppressExceptionNotSignalled" value="false"/>
<property name="ant.netrexxc.suppressMethodArgumentNotUsed" value="true"/>
<property name="ant.netrexxc.suppressPrivatePropertyNotUsed" value="true"/>
<property name="ant.netrexxc.suppressVariableNotUsed"       value="false"/>
<property name="ant.netrexxc.verbose"                       value="verbose1"/>
<property name="ant.netrexxc.console"                       value="false"/>
<property name="ant.netrexxc.java"                          value="true"/>
<property name="ant.netrexxc.time"                          value="true"/>
<property name="ant.netrexxc.source"                        value="1.8"/>
<property name="ant.netrexxc.target"                        value="1.8"/>
```

Building the translator

- Build sequence for target(s) 'default' is
  - [setecj, prepare, compile.runtime, compile.process, compile, init, jars, post.jar.prepare, compile.tests, -checkRunTestsRequired, run.tests, tests, withecj, default]
- Prepare defines buildnrc task which uses the Ant NetRexx program from ./tools/ant-task
  - It copies the .nrx files to ./build/classes and generates .java and .class files
- Java compile is done using the included modified eclipse compiler, which will -in this case- always create class format version 52 files, compatible with Java 1.8 and higher
- Test targets are achieved by putting the newly generated ./build/lib/NetRexxF.jar file on the CLASSPATH, and compiling and running all files in ./test and ./src/org/netrexx/diag
- Perform build:
  - ant clean
  - ant

Building the translator

- Demo

Building the translator

- Build new release
  - Update ./src/org/netrexx/process/NrVersion.nrx
  - Create documentation pdfs
  - Final check, swap lib/NetRexx?.jar with build/lib/NetRexx?.jar
  - And compile NetRexx itself with the new translator;
    - ant clean
    - ant
- If successful, build the new delivery package
  - ant package

Extending the translator

- NetRexx programs consist of one or more of the following clauses
  - Null clauses
    - Ignored
  - Assignments
    - term = expression
  - Method call
    - A method invocation()
  - Keyword instruction
    - one or more clauses, the first of which starts with a non-numeric symbol which is not the name of a variable or property in the current class
- Interestingly, you can have **if** as a variable name,
- extend NetRexx by creating new keyword instructions
- and stay backwards compatible

Extending the translator

- RxClauseParser.nrx, the interface used to group classes that parse and process individual clauses

```
class RxClauseParser interface

  method scan(pass=int)                 -- lexical/syntax scan (pass=0, 1, or 2)

  method getAssigns returns String[]    -- returns names (in lowercase) of
                                        -- any variables given a new value by this
                                        -- clause.  Should be null if none.

  method generate                       -- generate Java or bytecodes [only]
                                        -- must have no side-effects except on
                                        -- code generation (it's not called if NOJAVA is in effect).

  method interpret(cursor=RxCursor)     -- interpret (run from clauses)
```

  - NrSay.nrx
  - NrNop.nrx
  - NrExit.nrx
  - NrLoop.nrx
  - ...

Extending the translator

- RxTranslator.nrx passes four times through NetRexx source file
  - pass 0 : tokenises the source and parses prolog (options, package and import statements) and class instructions
  - pass 1 : processes resolution of properties and methods
  - pass 2 : parses method bodies and generates Java code
  - pass 3 : code interpretation (-exec)
  - pass 4 : code compilation (.java creation and compilation from memory)
- Pass 3 and 4 are mutually exclusive

The INTERPRET instruction

- RxParser.nrx parses NetRexx source file (3 times)

```
method parsemethodbody(reparse=boolean)
…
when iskey(tok.value,'INTERPRET')    then do
-- say '#MRE INTERPRET'
        tracer.traceclause(cursor.curclause)
        item=NrInterpret(rxt);
end
…
if item<=RxClauseParser then do
  cursor.curclause.lookaside=item
  item.scan(rxt.pass)
  if rxt.program.flag.java then item.generate()
end
…
```

Interpreting INTERPRET

- NrInterpret.nrx
  - RxClauseParser implementation for the INTERPRET instruction
- scan() parses the following expression and stores it as property `interpret`
- interpret() 'interprets' the property
  - mini-translator logic
  - New RxCursor(), RxFileReader(), RxClauser() and RxParser()
  - Clauses are tokenized (pass0)
  - Clauses are parsed as methodbody
    - Cannot return
    - Entry and exit must be same (block) level
  - If all well, clauses are interpreted
    - New variables are tracked, and removed from method-pool after interpretation

Compiled INTERPRET

- NrInterpret.nrx
  - generate() for compiled INTERPRET instruction
- Based on KK's NetRexxA API
- javax.script interface, a framework which allows to execute programs written in scripting languages in Java applications
  - Script execution
  - Variable bindings
  - Compilation (not used)
  - Discovery

Compiled INTERPRET

- NrInterpret.nrx generate()
  - create java file to call the javax.script.ScriptEngine (RxScriptEngine.nrx)
  - Local method and class (instance + static) variables are collected and bound ('put') to the javax.scriptengine
  - scriptengine eval() is called on string to interpret
  - Values of bound variables are 'get' from scriptengine
  - Bound variables are removed from scriptengine

Compiled INTERPRET

- RxScriptEngine.nrx
  - `class RxScriptEngine extends AbstractScriptEngine implements ScriptEngine uses javax.script.ScriptEngine`
  - Eval() receives the string to interpret
  - Builds complete NetRexx source around it
    - Get values (and types) of bound variables and define as properties
    - Define NetRexx source file, with class, main method, instance runScript method, and the actual INTERPRET clause(s)
  - Instantiates NetRexxI interpreter
- NetRexxI.nrx as API to 'translate' and 'exec' the provided NetRexx source

Compiled INTERPRET

- RxScriptEngine.nrx
  - `class RxScriptEngine extends AbstractScriptEngine implements ScriptEngine uses javax.script.ScriptEngine`
  - Eval() receives the string to interpret
  - Builds complete NetRexx source around it
    - Get values (and types) of bound variables and define as properties
    - Define NetRexx source file, with class, main method, instance runScript method, and the actual INTERPRET clause(s)
  - Instantiates NetRexxI interpreter
- NetRexxI.nrx as API to 'translate' and 'exec' the provided NetRexx source

Compiled INTERPRET

- Consider following program

```
it='Cheers'
interpret "it='Salut'"
say it
```

- NrInterpret.generate () generates following Java code at compile time
  - The 'it' variable is bound to the ScriptEngine which is given the "it='Salut'" string to evaluate, i.e. to interpret
  - After evaluation, the value of 'it' is 'gotten' and removed

```
javax.script.ScriptEngine nrEngine;
nrEngine=(new javax.script.ScriptEngineManager()).getEngineByName("NetRexxI");
nrEngine.put(" it",(java.lang.Object)it);
try {
 Object $rc = nrEngine.eval("it=\'Salut\'".toString());
 } catch (Exception $e) {
  throw(new netrexx.lang.InterpretException(" +++ Interpret error ("+$e.getMessage()+") : '"+"it=\'Salut\'"+"'" ));
}
it = (netrexx.lang.Rexx) nrEngine.get(" it");
(nrEngine.getBindings(javax.script.ScriptContext.ENGINE_SCOPE)).remove(" it");
```

Compiled INTERPRET

- At runtime, RxScriptEngine.nrx constructs the following NetRexx code around the string to evaluate
  - After the needed import statements, the interpret class is created (just a name)
  - The bound variables are created as properties, with differentiation between static and non-static (if needed), and their values 'gotten' from from the ScriptEngine bindings
  - Dummy variables are created to avoid unused variables warnings
  - A 'main' method and a 'runScript' method is created
  - An instance of the class is created
    - Note: static properties can be accessed from instance methods, but non-static properties cannot be accessed from static methods
  - The runScript method is invoked and the interpret string is added
  - The values of the properties are 'put' back into the ScriptEngine bindings so the calling program sees possible updated values

Compiled INTERPRET

```
import javax.script.Bindings
import javax.script.ScriptContext
import org.netrexx.process.RxScriptEngine
class interpret public
properties private static
 it = netrexx.lang.Rexx RxScriptEngine.instance.get(" it")
 nrxscriptengine = org.netrexx.process.RxScriptEngine RxScriptEngine.instance.get(" nrxscriptengine")
properties private static unused
  nowarn$it= it
  nowarn$nrxscriptengine= nrxscriptengine
method main($001=String[]) static
  $i = interpret()
  $i.runScript()
  (nrxscriptengine.getBindings(ScriptContext.ENGINE_SCOPE)).put(" it", java.lang.Object  it)
  (nrxscriptengine.getBindings(ScriptContext.ENGINE_SCOPE)).put(" nrxscriptengine", java.lang.Object  nrxscriptengine)
method runScript()
  it='Salut'
  return return
```

Compiled INTERPRET

- This NetRexx source code is fed from memory to an instantiated NetRexxI API class to interpret
- When run, output is in Spanish

```
java interp

Salut
```