RexxLA

NetRexx Language Reference in a nutshell

by Marc Remes

remesm@gmail.com

The NRL

- Syntax and structure
- Types and classes
- Terms
- Methods
- Type conversions
- Expressions and operators
- Clauses and instructions
- Indexed strings and arrays
- Assignments and variables
- Keyword instructions
- Built-in methods

Structure and syntax

- clauses
  - zero or more blanks (ignored), a sequence of tokens, zero or more blanks (ignored), and the delimiter ';' (implied by line end)
    - `'one = 1'`
    - `'if one = 1 then; say "one"`
- comments, replaced by a blank
  - block comment `/* this is a comment */`
  - line comment `-- this also`
  - Shebang `#!/usr/bin/env nr`

Structure and syntax

- implied semicolons and continuations
  - ; (clause end) is implied at end of each line, unless
    - line ends with multi-line block comment, continues after block comment
    - hyphen as last token –, replaced by blank
- case sensitivity
  - NetRexx is case-insensitive, unless 'option strictcase'
  - '`LOOP forever`' equals '`loop FOREVER`', as lookup for names of variables, methods etc
  - Lookup is case-insensitive and case-preserving, first exact-case, then case-insensitive, error when duplicate matches
  - External names (class, property, method) have defined spelling, first defined or used

Structure and syntax

- tokens
  - literal string, a sequence of characters delimited by " or '
    - escape sequences, the obvious ones (\n, \t, \r, \f, \", \', \\, \-, \0) and \xhh, \uhhhh
    - "This is a 'literal' string\n"
  - symbol
    - a group of characters
      - A-Z, a-z, 0-9, _$€
    - a number
      - 1, 1.00, 0.1E+9, 0x81(129), 2X81(-127), 0b1 (1), 1B1 (-1)
    - operator character
      - + - * % | & \= < >
    - special character,
      - . , ;) ( ] [

Types and classes

- NetRexx programs manipulate values
- Values have an associated type or class
- The type identifies
  - the nature of the value
    - properties
  - and the operations that can be carried out on the value
    - Methods
- Optionally qualified by package name ('package' instruction)
- NetRexx has its own default class in package netrexx.lang
  - Rexx
    - A sequence of characters with well-known rexx methods for arithmetic operations and string manipulation
    - substr, overlay, pos, translate, abs, format changestr, etc
- Primitives types
  - boolean char byte short int long float double
  - while not defined as class (not a subclass of java.lang.Object), no syntax distinction

Types and classes

- Dimensioned types
  - Types that have an associated dimension
  - Represented by square brackets [], with zero or more comma's
  - Dimension is number of comma's +1
    - Rexx type is distinct from Rexx[] type
    - int[10,10,10], a three-dimensional array
- Minor and dependent classes
  - Qualified by the 'parent'
  - Foo.Bar, to any depth, Foo.Bar.Pod
  - Short name access to methods and properties

Terms

- A syntactic unit which describes some value
- Simple term
  - A literal string
    - `'hello world'`
  - A symbol
    - `one`
  - A method call, '(' must be followed immediately after method name
    - `add(one, '2')`
  - An indexed reference
    - `in[one, two]`
  - An array initialiser
    - `[1, 2]`
  - A sub-expression
    - `(one / '2')`

Terms

- Compound term
  - A simple term, or qualified class, or qualified constructor (the 'stub')
  - Optionally followed by a continuation
    - one or more symbols (non-numeric), method calls or indexed references
    - separated by connector .
  - **`'hello world'`**`.word(2).pos('o')`
  - **`java.lang.math`**`.PI`
  - **`('hello' 'world')`**`.wordpos('hello')`
  - **`in[1, two]`**`.length()`

Methods

- Named routines belonging to a class
  - Referenced in a term, possibly part of an expression
    - `x = whatIsX()`
  - A clause on its own
    - a method on 'this', returned value discarded
      - `this.Is('X')`
    - or a constructor method
      - `X('wasTwitter')`
- '(' must immediately follow the name of the method which must be non-numeric
- Variable number of arguments
- When no arguments '()' can be omitted

Methods

- Method resolution
  - If in 'stub' of term
    - Search current class
    - Search super classes, which this current class 'extends'
    - Search 'uses' class-list
    - Search constructor
  - Else, stub must evaluate to a value of a type (or just a type)
    - Search type for method
    - Search super classes of type
- Finding the method
  - Same name
  - Same number of arguments and argument types
  - Return type must match
  - If more than one candidate
    - Conversion cost of arguments determines (lower is better)

Methods

- Method overriding
  - Same name as other class
  - The method in the other class is not 'private'
  - The other class is a super class of this class ('extends') or this class 'implements' the other class
  - The number and type of arguments match exactly
  - Must return same type (or a subclass of the type)
- Return types
  - When method declaration 'returns' a type, the value of the type must be returned
  - Otherwise, anything or nothing can be returned

Methods

- Constructor methods
  - Used to create a value of given type
  - Named identical to class name
  - Returns an 'instance' of the class, a value of the type
- If not present, default constructor with no arguments is implicitly created
  - Unless all 'static' or 'constant', or 'interface'
- Always parentheses ()
- Must call constructor in super class
  - If not present call to super() is implicitly added
- Returns 'this'

Type conversions

- When a value involved in an operation has a different type than needed
- Automatic conversion when no loss of information
  - Source and target are same type
  - Target type is superclass of source type
  - Source type has a dimension and target is Object
  - Source type is null and target is not primitive
  - Target and source types have well-known conversions
    - Rexx to binary number, char[], String, or char
    - String to binary number, char[], Rexx, or char
    - char to binary number, char[], String, or Rexx
    - char[] to binary number, Rexx, String, or char
    - binary number to Rexx, String, char[], or char
    - binary number to binary number (if no loss of information can take place)

Type conversions

- Explicit conversion (cast), possible loss of information
  - Permitted for all automatic conversions
  - Target type is a subclass of source type, or 'implements' it
  - Both target and source type are primitives
  - Target type is Rexx, or String
- Conversions have a cost
- Cost is calculated to select methods when several possibilities are there
- Automatic conversions have following cost
  - Zero when source and target have same type, or source type is null and target is primitive
  - Different costs for conversions between primitives
    - 8-bit to 64-bit number cost is higher than 8-bit to 32-bit number
  - Conversions which require creation of a new object have higher cost than those that don't
  - Conversions that may raise an exception cost more than those that never fail

Expressions and operators

- An expression is 'a general mechanism for combining one or more data items in various ways to produce a result, usually different from the original data'
- ..
- Consist of one or more terms and zero or more operators which denote operations to be carried out on the terms
  - Most operators act on two terms
  - Also prefix operations
- Evaluated from left to right, modified by parentheses (), and by normal algebraic precedence
- The result of evaluating any expression is a value, which has a known type

Expressions and operators

- Operators are constructed from one or more operator characters
- Five groups
  - Concatenation
  - Arithmetic
  - Comparative
  - Logical
  - Type operators
- First four work with strings or things converted to strings without information loss

Expressions and operators

- Concatenation operations
  - Blank'two' 'strings'
  - || 'two'|| ' strings'
  - Abuttal   'two ''strings'
- Arithmetic operations
  - + - * /
  - % integer divide
  - // Remainder
  - ** Power
  - Prefix -
  - Prefix +
  - Requires both terms to be numbers

Expressions and operators

- Comparative operators
  - Normal
    - = \= > <
    - <> ><   greater than or less than , \=
    - >= \<
    - <= \>
  - Strict
    - == \== >> <<
    - >>= \<<
    - <<= \>>
  - Some operators require both terms to be numbers

Expressions and operators

- Boolean operators
  - &
  - |
  - &&		Exclusive or
  - Prefix \	Not
  - In binary classes the operators act on all integers bits
- Type operator
  - String "abc"
  - Exception e
  - If i<=Object then say 'i is an Object'
  - If String => i then say 'i is a String'
  - If String <= Object then say 'String is an Object
    - <= or => tests whether value or type is a subclass of or same class, or vice versa

Expressions and operators

- Numbers
  - Well-known Rexx syntax
    - ’12’
    - ’-17.9’
    - ’0127.0650’
    - ’73e+128’
    - ’ + 7.9E-5 ’
    - ’00E+000’

Indexed strings and arrays

- Indexed strings aka stems
  - Must be a Rexx type, with value assigned before using sub-values
  - [must follow immediately after term
  - 'array' style syntax

```
surname = 'Unknown
surname['Fred']='Bloggs'
surname['Davy']='Jones'
try='Fred'
say surname[try] surname['Bert']
```

  - Multi-level

```
x=''
x['foo', 'bar']='OK'
say x['foo', 'bar']
y=x['foo']
say y['bar']
```

  - `loop name over surname; say surname[name];end`
  - `surname.exists('Bob')`
  - Assign null to drop sub-value

Indexed strings and arrays

- Arrays
  - Fixed size
  - [must follow immediately
  - Zero- based
  - Multi-dimensional
  - Declaration
    - a=int[], a one-dimensional array of integers
    - m=Rexx[,,], a three-dimensional array of Rexx types
  - Construction
    - a=int[3], a one-dimensional array for 3 integers
    - m=Rexx[3,3,3], a 3x3x3 array for Rexx types
  - Initialisation
    - a=[1, 2, 3] , an array of three integers, 1, 2 and 3
    - m=[[1,2], [3,4]], a two-dimensional array for integers, with values 1, 2 and 3, 4

Clauses and instructions

- 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

Assignments and variables

- term = expression
- Variable (term) has a type, determined by first assignment, cannot change
- Variable scope
  - Properties
    - Belongs to class
  - Method arguments
    - Belongs to method
  - Local variables
    - Belongs to method
- Names must be unique within a class, and are case-insensitive
  - Fred = FRED = fred
- Variables are handles, multiple variables can refer to same value

  first='A string'

  second=first

  first = 'A changed string'

  So is second
-

Keyword instructions

- Well-known Rexx instructions
  - Execution control
    - if, loop, iterate, leave, select, signal, do, return, exit
  - Class definition
    - class, properties, method
  - Meta
    - package, import, options, numeric, annotation
  - Miscellaneous
    - trace, say, parse, interpret, address, nop

Built-in methods

- Well-known Rexx built-in methods are available on the Rexx type (see netrexx/lang/Rexx.nrx)
  - String manipulation
    - changestr, insert, pos, lastpos, right, left, overlay, upper, translate..
  - Number methods
    - format, abs, d2x, x2d, x2b, max, min..
  - Misc
    - datatype, exists, date, time ..
- All available as method on the Rexx instance
  - `say 'Now is the time'.subword(1, 2)`
- And per netrexx/lang/RexxRexx.nrx in 'classic' non-oo style
  - `say subword('Now is the time', 3, 2)`