Purpose of this Tutorial

• Intended for Classic Rexx and/or IBM Object Rexx users
• With a working REXX knowledge
• And the Need/Desire to quickly learn NetRexx basics
• Based on the language differences
From classic Rexx to NetRexx

- Same/Similar language constructs
- But with subtile differences
- Both in Semantics
- … and Syntax (Notation)
- We Focus on the differences now
Notation of String Literals

- Backslash(\) used as an ESCAPE-character
- Rexx Literal „C:\tutor\Tutorial.PPT“
- Must be denoted as „C:\\tutor\\Tutorial.PPT“
- Attention: special escape sequences!!
Escape Sequences in String Literals

- \t Tabulation (tab)
- \n new-line (line-feed)
- \r return (carriage return)
- \f formfeed
- \" double quote
- \' single quote
- \0 null character
- \xhh hexadecimal character defined by hex digits (hh)
- \uhhhh unicode character defined by hex digits (hhhh)
- \ represents single backslash!
**Notation of Hexadecimal and Binary Literals**

- `'0123456789ABCDEF`x in Rexx
- Is: `16x‘0123456789ABCDEF‘` in NetRexx
- `'01000100`b in Rexx
- Is: `8b‘01000100‘` in NetRexx

- Both upper/lowercase x/b allowed
- Length 0 may be used (literal length counts)
Notation of Variable Names

- As usual in Programming languages, but
  - NO exclamation points (!) allowed in Variable names
  - NO question marks (?) allowed in variable names
  - In general: NO special characters (except ,,$‘ and underline „_“)
  - So why we did allow them in the first place?
Notation of Stems

- **Rexx** notation is `abc.def`
- **Object Rexx** notation is `abc.def`
- **OR** `abc[def]`
- **NetRexx** notation is **ONLY** `abc[def]`
- And **Stem** must be defined as a **Rexx** Variable before first usage, i.e.

  \[
  abc = \text{Rexx} \ <\text{default value}>\]
Notation of Stems (2)

- With multiple Indices:
  - Rexx notation is abc.x.y.z
  - Object Rexx notation is abc.x.y.z
  - OR abc[x,y,z]
  - NetRexx notation is ONLY abc[x,y,z]
  - And each Stem must be defined as a Rexx Variable before first usage, i.e.
  - abc = Rexx <default value>
Notation of Stems (3)

• Stems are now called 'Indexed Strings' in NetRexx
• Wrong, wrong, Mike
• Better we would be able to define a Stem as
  – X = RexxStem ','
  – Or Y=Stem ',' etc
• in NetRexx, you never know from the 'first Declaration' whether a Variable (Property) is a (Rexx) Stem or a (Rexx) String !! (it's a pity)
Attention (NetRexx specifics)

• X = Rexx , ‘
• May be
  – a simple ,Rexx‘ ,String‘ (to be able to use the NetRexx String functions (like length, index, pos, lastpos, etc, etc)
  – A Word-List ( to be able to use words(), wordpos(), etc)
  – A ,classic Rexx‘ Stem
  – A ,Rexx‘ Decimal Number
  – or each/any of that.
• But you cannot see from the NOTATION which variation is used.!
Using Functions vs. Methods (in Object Oriented Languages)

• It’s a PITY!
• When I do have a simple (Java) String, I can NOT use the ‘Rexx‘ WORDS or WORDPOS functions, for instance, directly, on this String.
• I will have to declare/convert it to a REXX String before – anyway, you may use Rexx(String)!
• Correct ??
• So why cannot we use Functions here (which will be applicable to all cases) ? Sorry, but why?
Attention

- Same notation for INDEXED ARRAYS and INDEXED Strings (formerly called 'Stems') in NetRexx, i.e.

- abc[x,y,z]

- may be
  - A NetRexx Indexed String (Stem) reference OR
  - A NetRexx/Java Array reference!
  - depending on initial 'TYPE' Definition
• Object REXX Array Indices start with 1
• but NetRexx/Java Indices start with 0
  – hence abc[1] is the FIRST element in Object Rexx
  – But abc[1] is the SECOND Element in NetRexx or Java
  – This difference applies ONLY to ARRAYS, NOT to Stems !!
CONTINUATION character

• CONTINUATION character
  – is a trailing COMMA (,) in classic Rexx and Object Rexx
  – But is a trailing HYPHEN (-) in NetRexx
• Advantage / pitfall ??
• Why do we need it at all (except for 'abut') ???
• Rey Rule (1): If a line ends with an OPERATOR, the next line is a continuation.
• Rey Rule (2): If a line starts with an OPERATOR (like +,-,*,/,&,,|,\, etc,etc) it MUST BE a continuation!
• Or what ?
• Concept of NOTES was always missing in Rexx!
• A 'Note' is a COMMENT at the end of the line
  – Must be written as /* my note */ in classic Rexx
  – Object Rexx and NetRexx use the double hyphen (--) to introduce a NOTE (as in SQL)
  – Note that Java uses // to introduce a Note (and ,-- as the decrement operator (which means REMAINDER in REXX !!))
  – A NOTE is always finished on the same line!

• … By the rivers of BABYLON !!
Operators

- Same set of operators in NetRexx than in classic Rexx!
- But COMPARISON of Text strings is CASE-BLIND by default!!
  - Hence 'abc' = 'ABC' in NetRexx!!
  - Must use 'strict comparison' in NetRexx when needing CASE-sensitive Comparison.
  - Probably more natural than original REXX definition!
  - Good choice for a change, Mike!
Concept of TYPES

• 'classic REXX' and OBJECT REXX are essentially TYPE-LESS languages!
• NetRexx (and Java) use/need STRICT TYPING
• NetRexx uses type 'Rexx' as default (and type Rexx is essentially TYPE-LESS again in NetRexx!)
• But NetRexx Type 'Rexx' is overloaded with too many different semantical meanings (Rexx String, Rexx Indexed String (Stem), Rexx WordList, Rexx (Decimal) Number, etc, etc)
Standard (Primitive) TYPES

- Boolean  (0/1)
- Byte  
  (0,1,2,3,4,5,6,7)
- Short  
  (half word SIGNED integer)
- Int  
  (full word SIGNED integer)
- Long  
  (double word SIGNED integer)
- Float  
  (full word SIGNED Real Number)
- Double  
  (double word SIGNED Real Number)
- Char  
  (is a UNICODE Character in NetRexx/Java)
- Primitive Types identical to Java!
Dimensioned TYPES

- Any Variable may be DIMENSIONED
- Use square BRACKETS ( [, , and ] ) to define dimensions
  - X = int[3,5]
  - Y = char[17]
- But NOTE that first ELEMENT has Index 0 and NOT 1 !!! (ill designed by Java!!)
- Difficult to distinguish Stems and Arrays!
Dimensioned TYPES

- Any Variable may be DIMENSIONED
- Use square BRACKETS ([, and ]) to define dimensions
- \( X = \text{int}[3,5] \)
- \( Y = \text{char}[17] \)
- But NOTE that first ELEMENT has Index 0 and NOT 1 !!! (ill designed by Java!!)
Dimensioned TYPES (2)

- Empty Index bounds are acceptable
- Similar to the concept of ‘adjustable‘ arrays in other languages
- Hence the following declarations are OK
  - X = int[,,]
  - Y = char[]
  - Z = Rexx[]
Initial (default) Values

- NetRexx uses the EQUAL Sign for TYPE definitions
- Hence syntax is
  - name = <type> [ <dimensions> ] <default value>
- Probably using the colon instead of the equal sign would have been a BETTER decision !!!
• With the current NetRexx notation you NEVER know whether a clause is an assignment or a type definition!
• Would also correspond more naturally to languages as Pascal or UML (Unified modelling language)
• item_no = Rexx 0 /*Stem!*/
• What do you think?
Example 1: The QTSMALL program

- The (ONLY) example of Mike Cowlishaws books 'the REXX language' and 'the NetRexx language'.
- So what's different?
- 
- <BREAK>
So what’s different: Labels and Procedures vs Methods

- Rexx and Object Rexx have the concept of Labels
- Denoted by a colon following the label name
- And there is a GO TO statement (named SIGNAL) in Rexx!
So what‘s different: SIGNAL vs RAISE vs SIGNAL

- Simple SIGNAL in REXX is a GO TO
- Object Rexx also has RAISE for 'Raising an Exception'
- Which is THROW in Java and SIGNAL in NetRexx!
- … by the rivers of BABYLON!
Jumping FORWARD and BACKWARDS

/* example3: simple loops */
F=,abc.def/* a simple sample file */
N=0
Loop1:
    x = linein(F)
    if length(x) = 0 then signal end_of_file
    n = n + 1
    say x
    signal loop1
End_of_file:
    say n,lines read
    exit
Jumping FORWARD and BACKWARDS (classic Rexx)

/* example3: simple loops */
F=,abc.def' /* a simple sample file */
N=0
Loop1:
    x = linein( F)
    if length(x) = 0 then signal end_of_file
    n = n + 1
    say x
    signal loop1
End_of_file:
    say n ,lines read'
    exit
import Rexx2Nrx.Rexx2RT.RexxFile

```java
class example3 uses RexxFile

properties public static
    FD_F = RexxFile Null
    F = Rexx 'abc.def'
    n = int 0
    xx = Rexx "

method main(args=String[]) static
    arg=Rexx(args) -- program arguments as single string
    arg=arg -- avoid NetRexx warning
    F = 'abc.def'
    FD_F = RexxFile.FD(F).access('READ')
    n = 0
    Loop1()
    exit
```
method Loop1() static public;

/* ... Attention: label: Loop1 is jumped back! */

loop label Loop1_again forever

    xx = FD_F.linein()

    if xx.length() = 0 then do
        End_of_file()
        return
    end--if

    n = n + 1

    say n||':'||xx

    iterate Loop1_again

end--Loop1_again
method End_of_file() static public ;
say n 'lines read'
exit
Summary

- Variables are called *Properties* in NetRexx.
- *GLOBAL variables* must be defined ahead of their usage (as *STATIC Properties after the CLASS statement*).
- As all variables are *LOCAL* by default (as in Object Rexx ::Methods and ::Routines !!)
- Avoid *Labels* whenever possible, use *STRUCTURED Statements* !!
Standard Program Layout
(Declarations)

- OPTIONS BINARY (when applicable)
- IMPORT package-name [.class-name]
- ...
- CLASS class-name [USES class-name-list]
- PROPERTIES PUBLIC STATIC
- Global 'Variable' declarations (visible outside class)
- PROPERTIES PRIVATE STATIC
- Global 'Variable' declarations (invisible outside class)
Standard Program Layout (Code)

- METHOD method-name PUBLIC STATIC
- METHOD method-name PRIVATE STATIC
- METHOD method-name (parameter-list) …
  - Where parameter-list is COMMA-delimited LIST of parameter-names (with types and default value)
  - E.g. Name1, Name2, … (default Type REXX)
  - Or Name1=Type1, Name2=Type2, …
Parameter Lists

- Semantically similar to USE ARG name-list in Object-Rexx METHODS.
- Parameter Names *must be different* to class PROPERTIES
- And *ARE INVISIBLE* (cannot be referenced) from out-side of the respective METHOD
- DEFAULT values may be provided for OPTIONAL parameters, e.g:
- METHOD ABC(par1=char[3], par2=int 0) PUBLIC STATIC
Parameter Lists

- Semantically similar to USE ARG name-list in Object-Rexx METHODS.
- Parameter Names **must be different** to class PROPERTIES
- And **ARE INVISIBLE** (cannot be referenced) from out-side of the respective METHOD
- DEFAULT values may be provided for OPTIONAL parameters, e.g:
- METHOD ABC(par1=char[3], par2=int 0) PUBLIC STATIC
Caution

• Notice that **PARSE ARG** is **ONLY** available for the **MAIN program** (main method)
• Notice that **PULL** and **PARSE PULL** are **NOT** available
• Do not forget the keyword **STATIC** for **methods associated with the CLASS**, and **NOT** the Objects constructed by the class.
Structured Statements

• Same structured statements than classic REXX

• With a few exceptions/additions:
  – Repetitive DO is called LOOP now
  – Additional key-words:
    • Label name
    • Protect term
    • Catch exception
    • Finally instruction-list

• Very well designed by M.F. Cowlishaw …
Structured Statements (2)

- Even PARSE-statement available
- PARSE statement variations no longer used (reserved Variable names like ARG, SOURCE, etc used in turn)
- With same Syntax and Semantics of the TEMPLATES than classic Rexx
- With a small exception:
  - No QUALIFIED Variables (like stems, etc) allowed in NetRexx (why?)
Notice that up to now we still didn’t use any OBJECTS

But we ARE now able to Write/Generate (procedural) NetRexx Code, at least.

Object Oriented Programming is another art, not part of this initial tutorial.

… Good LUCK