"Leaping from Classic to Object"

2019 International Rexx Symposium
Hursley, Great Britain
(September 2019)

© 2019 Rony G. Flatscher (Rony.Flatscher@wu.ac.at)
Wirtschaftsuniversität Wien, Austria (http://www.wu.ac.at)
Agenda

• History
• Getting Object Rexx
• New procedural features
• New object-oriented features
• Roundup
• Begin of the 90'ies
  – OO-version of Rexx presented to the IBM user group "SHARE"
  – Developed since the beginning of the 90'ies
    • Originally led by IBM's Simon Nash (UK, Hursley)
    • Later led by IBM's Rick McGuire (USA)
  – 1997 Introduced with OS/2 Warp 4
    • Support of SOM and WPS
  – 1998 Free Linux version, trial version for AIX
  – 1998 Windows 95 and Windows/NT
History, 2

• 2004
  – Spring: RexxLA and IBM join in negotiations about opensourcing Object REXX
  – November: RexxLA gets sources from IBM
  – Opensource developers taking responsibility
    • David Ashley, USA, OS2 guru, Linux freak, ooRexx aficionado
    • Rick McGuire, USA, original lead developer
    • Mark Hessling, Australia, Regina maintainer, author of numerous great, opensource, openplatform Rexx function packages
    • Rony G. Flatscher, Austria (Europe!), author of BSF4Rexx, ooRexx tester of many years

• 2005
  – Spring (March/April): RexxLA makes ooRexx freely available as opensource and openplatform
    • 2005-03-25: ooRexx 3.0
• Summer 2009
  – ooRexx 4.0.0
  – Kernel completely rewritten
    • 32-bit and 64-bit versions possible for the first time
    • New OO-APIs into the ooRexx kernel
      – e.g. BSF4ooRexx allows for implementing Java methods in Rexx!
• Latest release as of September 2019
  – ooRexx 4.2, Feb 24, 2014
  – AIX, Linux, MacOS, Windows
• ooRexx 5.0 in beta, about to be released?
Getting "Open Object Rexx" ("ooRexx") … for Free!

  - Choose the link to "ooRexx"
- [http://www.ooRexx.org](http://www.ooRexx.org)
  - Homepage for ooRexx
  - Links to Sourceforge
    - Source
    - Precompiled versions for AIX, Linux (Debian, K/Ubuntu, Red Hat, Suse, ), MacOS, Windows
    - Consolidated (great!) PDF-rendered documentation!
• Compatible with classic Rexx, TRL 2
  – New: execution of a Rexx program
    • Full syntax check of the Rexx program
    • Interpreter carries out all directives (leadin with "::")
    • Start of program
• "rexxc.exe": explicit tokenization of Rexx programs
• **USE** ARG in addition to PARSE ARG
  – among other things allows for retrieving stems by reference (!)
Example (ex_stem.rex)  
"USE ARG" with a Stem

/* ex_stem.rex: demonstrating USE ARG */

info.1 = "Hi, I am a stem which could not get altered in a procedure!"
info.0 = 1  /* indicate one element in stem */
call work info. /* call procedure which adds another element (entry) */
do i=1 to info.0 /* loop over stem */
   say info.i    /* show content of stem.i */
end
exit

work: procedure
   use arg great. /* note the usage of "USE ARG" instead of "PARSE ARG" */
   idx = great.0 + 1 /* get number of elements in stem, enlarge it by 1 */
   great.idx = "Object Rexx allows to directly access and manipulate a stem!"
   great.0 = idx /* indicate new number of elements in stem */
   return

/* yields: 
   Hi, I am a stem which could not get altered in a procedure!
   Object Rexx allows to directly access and manipulate a stem! */
New Procedural Features, 2

• Routine-directive
  – same as a function/procedure
  – if public, then even callable from another (!) program

• Requires-directive
  – allows for loading programs ("modules") with public routines and public classes one needs

• User definable exceptions
OO-Features Simply Usable by Classic Rexx Programs

- "Environment"
  - a directory object
    - allows to store data with a key (a string)
    - sharing information (coupling of) among different Rexx programs
  - ".local"
    - available to all Rexx programs within the same Rexx interpreter instance in a process
  - ".environment"
    - available to all Rexx programs running under all Rexx interpreter instances within the same process
    - gets searched after ".local"
Example (dec2roman.rex)

Classic Style

/* dec2roman.rex: turn decimal number into Roman style */
Do forever
call charout "STDOUT:", "Enter a number in the range 1-3999: "; PARSE PULL number
If number = 0 then exit
say " --->" number ":=" dec2rom(number)
End

dec2rom: procedure
PARSE ARG num, bLowerCase  /* mandatory argument: decimal whole number */
a.
   /* 1-9 */   /* 10-90 */   /* 100-900 */   /* 1000-3000 */
a.1.1 = "i" ; a.2.1 = "x" ; a.3.1 = "c" ; a.4.1 = "m" ;
a.1.2 = "ii" ; a.2.2 = "xx" ; a.3.2 = "cc" ; a.4.2 = "mm" ;
a.1.3 = "iii" ; a.2.3 = "xxx" ; a.3.3 = "ccc" ; a.4.3 = "mmm" ;
a.1.4 = "iv" ; a.2.4 = "xl" ; a.3.4 = "cd" ;
a.1.5 = "v" ; a.2.5 = "l" ; a.3.5 = "d" ;
a.1.6 = "vi" ; a.2.6 = "lx" ; a.3.6 = "dc" ;
a.1.7 = "vii" ; a.2.7 = "lxx" ; a.3.7 = "dcd" ;
a.1.8 = "viii" ; a.2.8 = "lxxx" ; a.3.8 = "dccc" ;
a.1.9 = "ix" ; a.2.9 = "xc" ; a.3.9 = "cm" ;
IF num < 1 | num > 3999 | \DATATYPE(num, "W") THEN
   DO
      SAY num": not in the range of 1-3999, aborting ..."
   EXIT -1
END

num = reverse(strip(num))  /* strip & reverse number to make it easier to loop */
tmpString = ""
DO i = 1 TO LENGTH(num)
   idx = SUBSTR(num,i,1)
   tmpString = a.i.idx || tmpString
END

bLowerCase = (translate(left(strip(bLowerCase),1)) = "L")  /* default to uppercase */
IF bLowerCase THEN RETURN tmpString
ELSE RETURN TRANSLATE(tmpString)  /* x-late to uppercase */
Example (routine1_dec2roman.rex)

/* routine1_dec2roman.rex: initialization */

a.
  /* 1-9 */ /* 10-90 */ /* 100-900 */ /* 1000-3000 */
a.1.1 = "i" ; a.2.1 = "x" ; a.3.1 = "c" ; a.4.1 = "m" ;
a.1.2 = "ii" ; a.2.2 = "xx" ; a.3.2 = "cc" ; a.4.2 = "mm" ;
a.1.3 = "iii" ; a.2.3 = "xxx" ; a.3.3 = "ccc" ; a.4.3 = "mmm" ;
a.1.4 = "iv" ; a.2.4 = "xl" ; a.3.4 = "cd" ;
a.1.5 = "v" ; a.2.5 = "l" ; a.3.5 = "d" ;
a.1.6 = "vi" ; a.2.6 = "lx" ; a.3.6 = "dc" ;
a.1.7 = "vii" ; a.2.7 = "lxx" ; a.3.7 = "dcc" ;
a.1.8 = "viii" ; a.2.8 = "lxxx" ; a.3.8 = "dccc" ;
a.1.9 = "ix" ; a.2.9 = "xc" ; a.3.9 = "cm" ;
.local-dec.2.rom = a. /* save in .local-environment for future use */

::routine dec2roman public
PARSE ARG num, bLowerCase /* mandatory argument: decimal whole number */

a. = .local-dec.2.rom /* retrieve stem from .local-environment */
IF num < 1 | num > 3999 | DATATYPE(num, "W") THEN DO
  SAY num": not in the range of 1-3999, aborting ..."
  EXIT -1
END

num = reverse(strip(num)) /* strip & reverse number to make it easier to loop */
tmpString = ""
DO i = 1 TO LENGTH(num)
  idx = SUBSTR(num,i,1)
  tmpString = a.i.idx || tmpString
END

bLowerCase = (translate(left(strip(bLowerCase),1)) = "L") /* default to uppercase */
IF bLowerCase THEN RETURN tmpString
ELSE RETURN TRANSULATE(tmpString) /* x-late to uppercase */
Example (use_routine1_dec2roman.rex)

/* use_routine1_dec2roman.rex */
Do forever
   call charout "STDOUT:", "Enter a number in the range 1-3999: ">
   PARSE PULL number
   If number = 0 then exit
   say "   --->" number "=" dec2roman(number)
End

::requires "routine1_dec2roman.rex" /* directive to load module with public routine */
Example (routine2_dec2roman.rex)

```plaintext
/* routine2_dec2roman.rex: Initialization code */
d1 = .array~of("", "i", "ii", "iii", "iv", "v", "vi", "vii", "viii", "ix")
d10 = .array~of("", "x", "xx", "xxx", "xl", "l", "lx", "lxx", "lxxx", "xc")
d100 = .array~of("", "c", "cc", "ccc", "cd", "d", "dc", "dcc", "dccc", "cm")
d1000 = .array~of("", "m", "mm", "mmm")

.local~roman.arr = .array~of(d1, d10, d100, d1000)  /* save in local environment */

::ROUTINE dec2roman PUBLIC /* public routine to translate number into Roman*/
  USE ARG num, bLowerCase/* mandatory argument: decimal whole number */

  IF num < 1 | num > 3999 | \DATATYPE(num, "W") THEN
    RAISE USER NOT_A_VALID_NUMBER /* raise user exception */
  END

  num = num~strip~reverse /* strip & reverse number to make it easier to loop */
  tmpString = ""
  DO i = 1 TO LENGTH(num)
    tmpString = .roman.arr[i]~at(SUBSTR(num,i,1)+1) || tmpString
  END

  bLowerCase = (bLowerCase~strip~left(1)~translate = "L") /* default to uppercase */
  IF bLowerCase THEN RETURN tmpString
  ELSE RETURN TRANSLATE(tmpString) /* x-late to uppercase */
```
/* use_routine2_dec2roman.rex */
Do forever
    call charout "STDOUT:", "Enter a number in the range 1-3999: ":
    PARSE PULL number
    If number = 0 then exit
    say "   --->" number "=" dec2roman(number)
End
::requires "routine2_dec2roman.rex" /* directive to load module with public routine */
New Object-oriented Features, 1

- Allows for implementing abstract data types (ADT)
  - "Data Type" (DT)
    - *a data type defines the set of valid values*
    - *a data type defines the set of valid operations for it*
    - examples
      - *numbers*: adding, multiplying, etc
      - *strings*: translating case, concatenating, etc.
  - "Abstract Data Type" (ADT)
    - *a generic schema defining a data type with*
      - *attributes*
      - *operations on attributes*
New Object-oriented Features, 2

• **Object-oriented features of Rexx**
  – allow for implementing ADTs
  – a predefined classification tree
  – allow for (multiple) inheritance
  – explicit use of metaclasses
  – tight security manager (!)

  • *allows for implementing any security policy w.r.t. Rexx programs*
    – untrusted programs from the net
    – roaming agents
    – company policy w.r.t. executing code in secured environment
About Implementing ADTs, 1

• Rexx and ADTs
  – Cannot define routines confined to a datatype!
  – Attributes can be encoded as
    • Rexx strings, e.g.
      ```rexx
birthday="19590520 13:01"
```
    • Rexx stems, e.g.
      ```rexx
birthday.date="19590520"
Birthday.time="13:01"
```
  – Quite complicated and can be error prone
    • Rexx programmers must know exactly the structure and all operations to implement!
About Implementing ADTs, 2

- ooRexx
  - Designed to easily implement ADTs
  - Directives
    ::CLASS adt_name
    ::ATTRIBUTE attr_name
    ::METHOD meth_name
  - An implemented ADT is sometimes termed "class", sometimes "type", sometimes "structure"
  - "Black box"
- Rexx users do not need to know any implementation details in order to use classes/types/structures!
• "object"
  – A synonym for "value of a specific type", "instance"
  – Possesses all attributes and methods of its class
  – Only reacts upon receiving messages
    • Message operator ~ (tilde, dubbed "twiddle")
    • Followed by a message name, optionally with arguments in parenthesis
    • Searches and invokes the method with the same name as the message name and returns any return value from the method
Example (dog.rex)
Defining Dogs ...

/* dog.rex: a program for dogs ... */

myDog = .Dog-new  /* create a dog from the class */
myDog-name = "Sweety"  /* tell the dog its name */
say "My name is:" myDog-name  /* now ask the dog for its name */
myDog-bark  /* come on show them who you are! */

::class Dog  /* name of the implemented ADT */
::attribute name  /* let it have an attribute */
::method bark  /* let it be able to bark */
say "Woof! Woof! Woof!"

/* yields: */

    My name is: Sweety
    Woof! Woof! Woof!

*/
Example (bigdog.rex)
Defining BIG Dogs ...

/* bgdoc.rex: a program for BIG dogs ... */

myDog = .BigDog~new  /* create a BIG dog from the class */
myDog~name = "Arnie"  /* tell the dog its name */
say "My name is:" myDog~name  /* now ask the dog for its name */
myDog~Bark  /* come on show them who you are! */

::class Dog  /* define the class "Dog" */
::attribute name  /* let it have an attribute */
::method bark  /* let it be able to bark */
say "Woof! Woof! Woof!"

/* the following class reuses most of what is already defined for the class "Dog" via inheritance; it overrides the way a big dog barks */
::class BigDog subclass Dog  /* define the class "BigDog" */
::method bark  /* let it be able to bark like big dogs do, all in uppercase! :) */
say "WOOF! WOOF! WOOF!"

/* yields: */

My name is: Arnie
WOOF! WOOF! WOOF!

*/
• Object Rexx' classification tree
  – Fundamental classes
    • Object, Class, Method, Message
  – Classic Rexx classes
    • String, Stem, Stream
  – Collection classes
    • Array, CircularQueue, List, Queue, Supplier
    • Directory, Properties, Relation and Bag, Table, Set
      – index is set explicitly by programs
  – Miscellaneous classes
    • Alarm, Monitor, ...
Example (fruit.rex)
A Bag Full of Fruits ...

/* fruit.rex: a bag, full of fruits ... */

Fruit_Bag = .bag-of("apple", "apple", "pear", "cherry", "apple", "banana", 
   "plum", "plum", "banana", "apple", "pear", "papaya", 
   "peanut", "peanut", "peanut", "peanut", "peanut", "apple", 
   "peanut", "pineapple", "banana", "plum", "pear", "pear", 
   "plum", "plum", "banana", "apple", "pear", "papaya", 
   "peanut", "peanut", "peanut", "apple", "peanut", "pineapple", 
   "banana", "peanut", "peanut", "peanut", "peanut", "peanut", 
   "peanut", "peanut", "pineapple", "banana", "peanut", "papaya", 
   "mango", "peanut", "peanut", "apple", "peanut", "pineapple", 
   "banana", "pear"")

SAY "Total of fruits in bag:" Fruit_Bag~items
SAY

Fruit_Set = .set~new~union(Fruit_Bag)
SAY "consisting of:"
DO fruit OVER Fruit_Set
   SAY right(fruit, 21) || ":" RIGHT( Fruit_Bag~allat(fruit)~items, 3 )
END
Example (fruit.rex)

Output

Total of fruits in bag: 56

consisting of:

- plum: 5
- cherry: 1
- pear: 6
- mango: 1
- banana: 7
- peanut: 20
- pineapple: 4
- papaya: 3
- apple: 9
Open Object Rexx ("ooRexx")

Roundup

- Adds features, long asked for, e.g.
  - Variables (stems) by reference (USE ARG)
  - Public routines available to other programs (concept of modules)
  - Very powerful and complete implementation of the OO-paradigm

- Availability
  - Free
  - Opensource
  - Openplatform
    - Precompiled versions for: AIX, Linux (rpm, deb), MacOSX, Solaris, Windows 98/NT/2000/XP/Vista/W7/W8

- Rony G. Flatscher, „Introduction to Rexx and ooRexx“, order form: http://www.facultas.at/flatscher