



Useful ooRexx Features missing from REXX

2019 – International Rexx Symposium
Hursley, September 2019

Rony G. Flatscher (Rony.Flatscher@wu.ac.at, <http://www.ronyRexx.net>)
Wirtschaftsuniversität Wien, Austria (<http://www.wu.ac.at>)





Overview

- Brief history
- Some interesting ooRexx features
 - `USE ARG`
 - `Routine`-directive
 - `Requires`-directive
 - `Array`
- Roundup



History, 1

- Begin of the 90's
 - OO-version of Rexx introduced to SHARE by IBM
 - 1997 delivered with OS/2 Warp 4
 - "Object REXX"
 - *Direct support for SOM and WPS*
 - 1998 free Linux-version, test version for AIX
 - 1998 Windows 95 and Windows/NT



History, 2

- 2004
 - Spring: RexxLA and IBM (Böblingen) start negotiations about transferring the source code of Object REXX
 - November: RexxLA receives the source code from IBM
 - Opensource-developers of RexxLA take over, among them
 - Rick McGuire, *the* IBM-core developer of Object REXX at that time!
- 2005-03-25
 - RexxLA releases
 - "Open Object Rexx (ooRexx) 3.0"
 - Accompanied with the respective, complete source code!



History, 3

- Summer 2009
 - "ooRexx 4.0"
 - Kernel *completely rewritten* !
 - 32-bit and 64-bit versions possible for the first time!
 - New OO-APIs into the ooRexx-kernel for C++
 - E.g.. BSF4ooRexx: Java-Methods can be implemented in Rexx code!
- Current, official version from February 2014
 - "ooRexx 4.2"
 - AIX, Linux, MacOSX, Windows



History, 4

- "ooRexx 5.0" in Beta
 - More stable than 4.2
 - Faster than 4.2 (20% to 2000%!)
 - Many great, new features



WWW-Ressources for "ooRexx", 1

- <http://www.RexxLA.org>
 - Homepage of the "Rexx Language Association"
- <https://sourceforge.net/projects/oorex/files/oorex/5.0.0beta/>
 - "ooRexx 5.0" Beta
 - Windows, Linux
- <https://sourceforge.net/projects/bsf4oorex/files/beta/20190203>
 - MacOSX-Version (includes ooRexx 5.0 beta), e.g.
 - [b4r_641_500_64Bit_macosx-20190821-r11896-macosx.zip](#)
 - BSF4ooRexx for Linux and Windows



WWW-Ressources for "ooRexx", 2

- <https://sourceforge.net/projects/bsf4oorexx/files/Sandbox/aseik/ooRexxIDEA/beta/1.0.3.1/>
 - *GREAT* "ooRexx 5.0"-Plugin for IntelliJ/IDEA
 - Handles REXX and ooRexx code
 - Checks syntax, color highlights code
 - Instructions for downloading and installing IntelliJ/IDEA
 - Supplies download link to IntelliJ/IDEA
 - Community edition under AL 2.0, available for free
 - Versions for Windows, Linux and MacOS



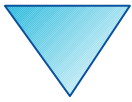
For REXX-Programmers

Interesting New Features

- **USE ARG** instead of **PARSE ARG**
 - Allows one to directly refer to supplied stems!
- **Routine-Directive**
 - Enables the definition of private and public routines
- **Requires-Directive**
 - Makes sure that external Rexx programs get automatically called by the interpreter
- **Array**
 - Allows one to collect and process any values

▼ USE ARG, 1

- "PARSE ARG"
 - Allows one to fetch arguments as *strings*
 - Problem, if one needs to refer to stems
 - Only the string value will be fetched, not the caller supplied stem!
 - Solution in Rexx
 - "PROCEDURE **EXPOSE ...**"



USE ARG, 2

REXX-Problem

```
person. = "no person" /* default value for stem variable "person." */
person.1 = "Anton"
person.2 = "Berta"
person.0 = 2 /* number of entries in stem array "person." */
call show person.
say "person.0=["person.0"] person.3=["person.3"]"
exit

show: procedure /* local scope */
parse arg person. /* a local stem "PERSON.", default value from caller */
say "show(): person.=["person.0"]"
do i=1 to person.0
    say "show(): person #" i ":" person.i
end
person.3 = "Caesar" /* add a new element */
person.0 = 3 /* now we have three elements ! */
return
```

Output:

```
show(): person.=[no person]
12 *-* do i=1 to person.0
5 *-* call show person.
Error 41 running E:\DropBox\code_use_arg_01.rex line 12: Bad arithmetic conversion.
Error 41.1: Nonnumeric value ("no person") used in arithmetic operation.
```



USE ARG, 3

REXX-Solution: "EXPOSE stem."

```
person. = "no person"      /* default value for stem variable "person." */
person.1 = "Anton"
person.2 = "Berta"
person.0 = 2                /* number of entries in stem array "person." */
call show person.
say "person.0=["person.0"] person.3=["person.3"]"
exit

show: procedure expose person. /* local scope, break insulation */
  say "show(): person.=["person.0"]"
  do i=1 to person.0
    say "show(): person #" i ":" person.i
  end
  person.3 = "Caesar"      /* add a new element */
  person.0 = 3              /* now we have three elements ! */
  return
```

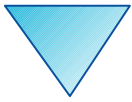
Output:

```
show(): person.=[no person]
show(): person # 1: Anton
show(): person # 2: Berta
person.0=[3] person.3=[Caesar]
```

USE ARG, 4

ooRexx, New Feature "USE ARG"

- "USE ARG"
 - Fetches arguments as *references*
 - Strings will be *always* fetched as (immutable) strings
 - Stems as any other data type will always be fetched by *reference*
 - Makes it possible to interact directly with the stem of the caller!
 - It is possible to force arguments (**USE STRICT ...**)
 - It is possible to define default values, e.g.
 - **USE ARG** arg1="aha", counter=(99-1.9)



USE ARG, 5

ooRexx-Solution: "USE ARG stem."

```
person. = "no person"      /* default value for stem variable "person." */
person.1 = "Anton"
person.2 = "Berta"
person.0 = 2                /* number of entries in stem array "person." */
call show person.
say "person.0=["person.0"] person.3=["person.3"]"
exit

show: procedure            /* local scope */
use arg person.           /* refers to person. in caller! */
say "show(): person.=["person.0"]"
do i=1 to person.0
    say "show(): person #" i ":" person.i
end
person.3 = "Caesar"       /* add a new element */
person.0 = 3               /* now we have three elements ! */
return
```

Output:

```
show(): person.=[no person]
show(): person # 1: Anton
show(): person # 2: Berta
person.0=[3] person.3=[Caesar]
```



USE ARG, 6

ooRexx-Solution: "USE ARG stem."

```
person. = "no person"      /* default value for stem variable "person." */
person.1 = "Anton"
person.2 = "Berta"
person.0 = 2                /* number of entries in stem array "person." */
call show person.
say "person.0=["person.0"] person.3=["person.3"]"
exit

show: procedure            /* local scope */
use arg aaaaaa.           /* refers to person. in caller! */
say "show(): aaaaaa.=["aaaaaa."]"
do i=1 to aaaaaa.0
    say "show(): person #" i ":" aaaaaa.i
end
aaaaaa.3 = "Caesar"       /* add a new element */
aaaaaa.0 = 3              /* now we have three elements ! */
return
```

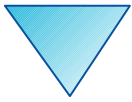
Output:

```
show(): aaaaaa.=[no person]
show(): person # 1: Anton
show(): person # 2: Berta
person.0=[3] person.3=[Caesar]
```

▼ Routines

Internal Routines, 1

- Internal Routines
 - Label
 - Identifier followed by a colon
 - Optional **PROCEDURE** keyword instruction
 - Creates a local scope
 - **EXPOSE**-subkeyword
 - Allows direct access to listed variables in caller
 - Followed by instructions
 - **RETURN**-keyword instruction to return to caller



Routines

Internal Routines, 2

```
say pp("anton") quote("berta") quote("caesar", '<', '>')  
exit
```

```
pp: procedure /* "pretty print" */  
  parse arg val  
  return "["val"]"
```

```
quote: procedure /* arguments with default values */  
  use strict arg val, left='', right=''  
  return left || val || right
```

Output:

```
[anton] "berta" <caesar>
```



Directives, 1

- Directives
 - Start with two colons (::) at the end of a program
 - A new directive ends a previous one
 - Followed by the name of the directive
 - As of August 2019: ooRexx 5.0 beta
 - ANNOTATE (new in 5.0), ATTRIBUTE, CLASS, CONSTANT, METHOD, OPTIONS, REQUIRES, RESOURCE (new in 5.0), ROUTINE
 - Possibly followed by additional information depending on the directive



Directives, 2

- Executing a program with the ooRexx-Interpreter
 - Reads the programs
 - Checks for syntax errors
 - *If* directives contained
 - Interpreter carries out the directives in sequential order
 - *Therefore directives are instructions to the interpreter to set up something on our behalf **before** our program gets interpreted with the first instruction in the first line!*
 - Instructions get interpreted starting with line 1

▼ Routine-Directive, 1

- **::ROUTINE**

- Followed by the name of the routine
- Optionally followed by one of the subkeywords **PUBLIC** or **PRIVATE** (= default)
- Followed by the Rexx-instructions of the routine
 - **RETURN** or **EXIT** instructions for returning from it
- Noteworthiness
 - Each **Routine**-directive gets managed as if it was a proper, isolated Rexx program and therefore possesses the scope of a Rexx program!



Routine-Directive, 2

```
say pp("anton") quote("berta") quote("caesar", '<', '>')  
exit
```

```
::routine pp      /* "pretty print" */  
  parse arg val  
  return "["val"]"
```

```
::routine quote  /* arguments with default values */  
  use strict arg val, left='', right=''  
  return left || val || right
```

Output:

```
[anton] "berta" <caesar>
```

▼ Routine-Directive, 3

- Advantage?
 - Superficially none ! :-)
- Advantage!
 - Storing a collection of frequently used Rexx routines in a proper file
 - Add subkeyword **PUBLIC** to the routines
 - Once a Rexx program calls another Rexx program, upon return all the public routines of the called Rexx program become directly accessible!



Routine-Directive, 4

```
/* file: "my_routines.rex" */  
  
::routine pp public /* "pretty print" */  
  parse arg val  
  return "["val"]"  
  
::routine quote public /* arguments with default values */  
  use strict arg val, left="", right=""  
  return left || val || right
```

```
/* file: "use_my_routines.rex" */  
  
call my_routines.rex /* upon return all its public routines accessible! */  
say pp("anton") quote("berta") quote("caesar", '<', '>')
```

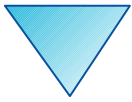
Output of executing "use_my_routines.rex":

```
[anton] "berta" <caesar>
```

▼ Requires-Directive, 1

- **::REQUIRES**

- Followed by the name of a Rexx program
- Interpreter will call that Rexx program on our behalf
 - Interpreter is able to optimize: if a required Rexx program was already required, it merely reuses the cached runtime information
 - Upon return all public routines become directly available!
- Also enables defining explicitly call dependencies and have the interpreter resolve them!



Requires-Directive, 2

```
/* file: "my_routines.rex" */  
  
::routine pp public /* "pretty print" */  
  parse arg val  
  return "["val"]"  
  
::routine quote public /* arguments with default values */  
  use strict arg val, left="", right=""  
  return left || val || right
```

```
/* file: "use_my_routines_via_requires.rex" */  
  
say pp("anton") quote("berta") quote("caesar", '<', '>')  
  
::requires my_routines.rex /* interpreter CALLs on our behalf */
```

Output of executing "use_my_routines_via_requires.rex":

```
[anton] "berta" <caesar>
```

▼ Arrays, 1

- ooRexx comes with a built-in **Array** class
 - Makes it possible to store and retrieve values with an integer index
 - First element has the index value **1**, the second element has the index value **2** etc.
 - Square brackets enclose the index value
 - ooRexx offers to simply loop over all collected values in sequence with the new loop variant
 - **DO...OVER**

▼ Arrays, 2

- ooRexx 5.0 Beta (as of: August 2019)
 - Makes it easy to define an array by merely denoting its items in a comma separated list
 - A comma separated list which is enclosed in parentheses
 - Parentheses cause Rexx to evaluate the contained expression
 - If the expression is just a comma separated list an array gets created for it
 - Parentheses can be omitted under certain conditions



Arrays, 3

Looping with "DO...OVER"

```
arr=("anton", "berta", "caesar") /* an array with three elements */
```

```
do element over arr  
  say element  
end
```

Output:

```
anton  
berta  
caesar
```



Arrays, 4

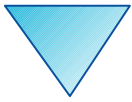
Looping with Numeric Index

```
arr=("anton", "berta", "caesar") /* an array with three elements */
```

```
do i=1 to 3  
  say i:"" arr[i]  
end
```

Output:

```
1: anton  
2: berta  
3: caesar
```



Arrays, 5

"DO...OVER" Yields Stored Values

```
arr=("anton", "berta", "caesar")  /* an array with three elements */  
  
arr[4]="dora"  
/* arr[5] is not being used */  
  
arr[6]="friedrich"  
do element over arr  
  say element  
end
```

Output:

```
anton  
berta  
caesar  
dora  
friedrich
```



Arrays, 6

Accessing Non-Existent Entries

```
/* as on the right hand side the expresion consists of a comma  
separated list, we can forgo the parenthesis */  
arr= "anton", "berta", "caesar" /* an array with three elements */  
  
arr[4]="dora" /* arr[5] is not being used */  
  
arr[6]="friedrich"  
  
do i=1 to 6  
  say i":" arr[i]  
end
```

Output:

```
1: anton  
2: berta  
3: caesar  
4: dora  
5: The NIL object  
6: friedrich
```



Roundup

- *ooRexx*
 - Most modern Rexx on earth! :)
 - "*Swiss Army Knife*" for modern programmers! 8)
 - Available e.g. for Windows, Linux, MacOS
 - Can be made available on "Linux on IBM Z" !!
 - Hence, the most modern Rexx can be run on IBM mainframes !
 - With *BSF4ooRexx* on the mainframe it becomes even possible to access mainframe software via their Java APIs from ooRexx
 - E.g. access to DB2 from ooRexx!
 - "The sky's the limit!" ;-)

URLs

- RexxLA-Homepage (non-profit SIG, owner of ooRexx, BSF4ooRexx)
<<http://www.rexxla.org/>>
- ooRexx 5.0 beta on Sourceforge
<<https://sourceforge.net/projects/ooress/files/ooress/5.0.0beta/>>
- BSF4ooRexx on Sourceforge (ooRexx-Java bridge)
<<https://sourceforge.net/projects/bsf4ooress/>>
- Introduction to ooRexx (254 pages)
<<https://www.facultas.at/Flatscher>>
- JetBrains "IntelliJ IDEA", powerful IDE for all operating systems
 - <<https://www.jetbrains.com/idea/download>>, free "Community-Edition"
 - Alexander Seik's ooRexx-Plugin with readme (as of: 2019-10-03)
 - <<https://sourceforge.net/projects/bsf4ooress/files/Sandbox/aseik/ooRexxIDEA/beta/1.0.5/>>