# 2025 RexxLA International Rexx Language Symposium Proceedings

**René Vincent Jansen (ed.)**

**THE REXX LANGUAGE ASSOCIATION**
**RexxLA Symposium Proceedings Series**
**ISSN 1534-8954**

## Publication Data

A publication of **RexxLA Press**

The responsible publisher of this edition is identified as *IBizz IT Services and Consultancy*, Amsteldijk 14, 1074 HR Amsterdam, a registered company governed by the laws of the Kingdom of The Netherlands.

The RexxLA Symposium Series is registered under ISSN 1534-8954
The 2025 edition is registered under ISBN 978-94-038-0342-5

ISSN 1534-8954

9 771534 895004 >

ISBN 978-94-038-0342-5

9 789403 803425 >

**2025-05-16** First Edition

# Introduction

## History of the International Rexx Language Symposium

In 1990, Cathie Dager of SLAC[1] convened the organizing committee for the first independent Rexx[2] Symposium for Developers and Users. SLAC continued to organize this annual event until the middle of the 1990's when the RexxLA took over that responsibility. Symposia have been held annually since 1990.

## About RexxLA

During the 1993 Symposium in La Jolla, California, plans for a Rexx User Group materialized. The Rexx Language Association (RexxLA), as it was called, is an independent, non-profit organization dedicated to promoting the use and understanding of the Rexx programming language. RexxLA manages several open source implementations of Rexx.

## The selection procedure

Presentation proposals are solicited yearly using a CFP[3] procedure, after which the RexxLA symposium comittee reviews them and votes which presentations are selected for the symposium. The presentations are peer reviewed before being presented. Presenters are not compensated for their presentations.

## Location

The 2025 symposium was held in The Wirtschaftsuniversität Vienna, Austria and online from 4 May 2025 to 7 May 2025.

## Organizing Committee

- Chip Davis
- Gil Barmwater
- Jon Wolfers
- Mark Hessling
- René Jansen
- Rony G. Flatscher
- Terry Fuller

[1]Stanford Linear Accelerator Center, since 2008 SLAC National Accelerator Laboratory
[2]Cowlishaw, M. F., **The REXX Language** (second edition), ISBN 0-13-780651-5, Prentice-Hall, 1990.
[3]Call For Papers.

# Contents

1

# ooRexx Tutorial – Rony G. Flatscher

## Date and Time

4 May 2025, 11:15:00 GMT

## Presenter

Rony G. Flatscher

## Presenter Details

Rony works as a professor for Business informatics ("Wirtschaftsinformatik") at the Vienna University of Economics and Business Administration (Wirtschaftsuniversität Wien) and uses Open Object Rexx for teaching Business Administration and MIS students the object-oriented paradigm, as well as remote-controlling (automating) Windows and Windows end-user applications (e.g. MS Office, Open Office) as well as Java and Java applications (he is the author of BSF4ooRexx, the ooRexx-Java bridge, which uses Apache BSF and had Rony invited to become an ASF member). He consults and trains in all of his research fields.

## Session Abstract

The ooRexx programming language is based on Rexx and adds object-oriented concepts like classes, objects, and the message paradigm (inspired by SmallTalk) to the Rexx language. In addition, ooRexx introduces directives that get carried out by the interpreter in the new setup phase, followed by the execution phase, in which the features introduced by the directives can be immediately exploited. This tutorial first sketches the fundamental concepts of the programming language Rexx and then introduces the most important additions of ooRexx to the Rexx language, which are demonstrated in short, nutshell examples.

# "ooRexx Tutorial"

The 2025 International Rexx Symposium

Vienna, Austria

May 4th – May 7th 2025

© 2025 Rony G. Flatscher (Rony.Flatscher@wu.ac.at)
Wirtschaftsuniversität Wien, Austria (http://www.wu.ac.at)

## Agenda

- Brief History
- Rexx Basics
- Object Rexx
  - Some new features like
    - USE ARG
  - New: Directives
    - ::ROUTINE, ::REQUIRES
    - ::CLASS, ::ATTRIBUTE, ::METHOD
    - (::ANNOTATE, ::CONSTANT, ::OPTIONS, ::RESOURCE)
- Roundup

## Some Historical Bits on Rexx

- Created for IBM mainframes to make programming easier compared to the rather awkward EXEC2
  - **Rexx design goals:** "human centric", "keep the language small", "easy to learn", "easy to understand hence easy to maintain"
  - Rexx is **still instrumental for IBM mainframe operating systems** today!
- Extremely successful in the 80'ies
  - Companies selling Rexx interpreters successfully, **ANSI/INCITS standard** (!)
- Object-oriented successor ("Object Rexx") in the 90'ies
  - **Open-sourced** in 2005 by RexxLA.org – "open object Rexx" (ooRexx)
    - Available for **all major operating systems**
    - Possible to programme even MS Windows applications via OLE ...

## Fundamental Rexx Concepts, 1

- "Everything is a string"
  - If a string represents a number, one can carry out arithmetic
- Three instruction types
  1) Assignment
     - Variable name followed by the assignment operator (=) and an expression
  2) Keyword instruction
     - Keywords are English words conveying the intent of the keyword instruction, e.g. SAY, DO, IF, LOOP, CALL, PARSE, SELECT, ITERATE, LEAVE, INTERPRET, ...
     - Makes Rexx code legible as if it was pseudocode
  3) Commands
     - A string passed to the operating system for execution (as if typed in a window)

## Fundamental Rexx Concepts, 2

- White space can be freely used to format code for better legibility
  – Space around operators gets removed
  – White space between symbols will be reduced to a single space serving as concatenation operator
  – Hence indentations with white space not significant
- Case of symbols irrelevant
  – Rexx uppercases everything outside of quoted strings
  – No (frustrating) casing errors for novices

```
sum = 17 +   19
      hint   =     "/ 17+19:"      sum
  say   hint  "/"    upper( "aü ẞ äöü ÄÖÜ A/ ? \\--// :-)"   )
```

$\downarrow$

```
SUM=17+19
HINT="/ 17+19:" SUM
SAY HINT "/" UPPER("aü ẞ äöü ÄÖÜ A/ ? \\--// :-)")
```

**Output:**

```
 / 17+19: 36 / Aü ẞ äöü ÄÖÜ A/ ? \\--// :-)
```

## Fundamental Rexx Concepts, 3

- Rexx nutshell examples to stress fundamental concepts
  – Illustrate the Rexx language
    - Code intuitive and easy understandable as it looks like pseudo code
  – Same  examples in the popular Python language to allow direct comparisons
    - Cannot be understood without an introduction to many concepts of the Python language

Nutshell Example, 1

## Instructions

```
    /* an assignment instruction:      */
a="hello world"   /* assigns "hello world" to a variable named a  */

    /* a keyword instruction:          */
say a             /* output: hello world */

    /* a command instruction:          */
    /* a command (could be typed into a command line window)          */
"echo Hello World 2" /* execute command                               */
   /* variable RC contains the command's return code, 0 means success */
if rc=0 then say "success!"
        else say "some problem occurred, rc="rc /* show return code   */
```

**Output:**

```
hello world
Hello World 2
Success!
```

```
# an assignment instruction
a="hello world"     # assigns "hello world" to a variable named a

# no keyword instruction for output, using built-in function print()
print(a)

# no command instruction using module subprocess instead
import subprocess   # import subprocess module
# execute command
completedProcess=subprocess.run("echo Hello World 2", shell=True) # run
rc=completedProcess.returncode  # fetch return code, an int
if rc==0:
    print("found!") # indentation mandatory (forcing a block)
else: # must use + (concatenation operator) with str() function
    print("some problem occurred, rc="+str(rc)) # turn rc into a string
```

**Output:**

```
hello world
Hello World 2
Success!
```

Nutshell Example, 2

## Blocks, Selection, Multiple Selections

```
max=5             /* number of repetitions   */
loop a=1 to max   /* loop block              */
   select         /* nested block # 1        */
      when a=1 then say a": first round"
      when a=2 then say a": second round"
      when a=3 then say a": third round"
      otherwise say "(a="a")"
   end

   if a=max then
   do             /* nested block # 2        */
      say "-> a=max"
      say "-> last round!"
      say "-> loop will end"
   end
end
```

**Output:**

```
1: first round
2: second round
3: third round
(a=4)
(a=5)
-> a=max
-> last round!
-> loop will end
```

```
max=5       # number of repetitions
for a in range(1,max+1): # loop with range() function, must add 1 to max
            # must use str() function with + (concatenation operator)
    match a:    # must be indented, "match" needs Python 3.10 or higher
        case 1: print(str(a)+": first round")   # nested block # 1
        case 2: print(str(a)+": second round")  # nested block # 1
        case 3: print(str(a)+": third round")   # nested block # 1
        case _: print("(a="+str(a)+")")  # default, nested block # 1

    if a==max:  # must be indented, must use == instead of =
        print("-> a==max")          # nested block # 2
        print("-> last round!")     # nested block # 2
        print("-> loop will end")   # nested block # 2
```

**Output:**

```
1: first round
2: second round
3: third round
(a=4)
(a=5)
-> a==max
-> last round!
-> loop will end
```

Nutshell Example, 3

## Parsing Strings

```
text = " John      Doe   Vienna Austria"
parse var text firstName lastName city country
say "first name:" firstName", last name:" lastName", city:" city
```

```
text = "Mary Doe Tokyo Japan"
parse var text firstName lastName city .   /* ignore country */
say "first name:" firstName", last name:" lastName", city:" city
```

**Output:**

```
first name: John, last name: Doe, city: Vienna
first name: Mary, last name: Doe, city: Tokyo
```

```
text      = " John      Doe   Vienna Austria"
words     = text.split()    # create list of words
firstName = words[0]        # assign to variable
lastName  = words[1]        # assign to variable
city      = words[2]        # assign to variable
print("first name:",firstName+",","last name:",lastName+",","city:",city)
```

```
text  = "Mary Doe Tokyo Japan"
words = text.split()    # create list of words
# assign multiple elements in a single statement
firstName, lastName, city = [words[i] for i in (0, 1, 2)]
print("first name:",firstName+",","last name:",lastName+",","city:",city)
```

**Output:**

```
first name: John, last name: Doe, city: Vienna
first name: Mary, last name: Doe, city: Tokyo
```

## ooRexx: Some New Features

- Compatible with classic Rexx, TRL 2
  - New sequence of execution of Rexx programs:

    Phase **1** (**load**): Full syntax check of the Rexx program upfront

    Phase **2** (**setup**): Interpreter carries out all directives (lead in with "::")

    Phase **3** (**execution**): Start of program execution with line # 1
- rexxc[.exe]: compiles Rexx programs
  - If *same bitness* and *same endianness*, on *all* platforms
- USE ARG (in addition to PARSE ARG)
  - among other things allows for retrieving stems *by reference (!)*
- Line comments, led in by two dashes ("--")

  *-- comment until the line ends*

## Stem, Classic REXX "stemclassic.rex"

```
s.1="Entry # 1"
s.2="Entry # 2"
s.0=2          /* total number of entries in stem          */

call add2stem  /* add to stem using an (internal) routine  */

do i=1 to s.0  /* iterate over all stem array entries      */
   say "#" i":" s.i
end
exit

add2stem: procedure expose s. -- allow access to stem
  n=s.0+1      /* add after last current entry             */
  s.n="Entry #" n "added in add2stem()"
  s.0=n        /* update total number of entries in stem   */
  return

/* yields:

   # 1: Entry # 1
   # 2: Entry # 2
   # 3: Entry # 3 added in add2stem()

*/
```

## Stem, REXX with USE ARG "stemusearg.rex": *No EXPOSE*

```
s.1="Entry # 1"
s.2="Entry # 2"
s.0=2             /* total number of entries in stem          */

call add2stem s.  /* supply stem as an argument!              */

do i=1 to s.0     /* iterate over all stem array entries      */
   say "#" i":" s.i
end
exit

add2stem: procedure  /* no "expose s." needed anymore !       */
  use arg s.     /* USE ARG allows to directly refer to the stem */
  n=s.0+1        /* add after last current entry              */
  s.n="Entry #" n "added in add2stem()"
  s.0=n          /* update total number of entries in stem    */
  return

/* yields:

   # 1: Entry # 1
   # 2: Entry # 2
   # 3: Entry # 3 added in add2stem()

*/
```

## Stem, ooRexx USE ARG "stemroutine1.rex": *No EXPOSE*

```
s.1="Entry # 1"
s.2="Entry # 2"
s.0=2               /* total number of entries in stem           */

call add2stem s.    /* supply stem as an argument!               */

do i=1 to s.0       /* iterate over all stem array entries       */
   say "#" i":" s.i
end

::routine add2stem
  use arg s.     /* USE ARG allows to directly refer to the stem */
  n=s.0+1        /* add after last current entry                 */
  s.n="Entry #" n "added in add2stem()"
  s.0=n          /* update total number of entries in stem       */
  return

/* yields:

   # 1: Entry # 1
   # 2: Entry # 2
   # 3: Entry # 3 added in add2stem()

*/
```

13

## Stem, ooRexx USE ARG "stemroutine2.rex": *No EXPOSE*

```
s.1="Entry # 1"
s.2="Entry # 2"
s.0=2               /* total number of entries in stem           */

call add2stem s.    /* supply stem as an argument!               */

do i=1 to s.0       /* iterate over all stem array entries       */
   say "#" i":" s.i
end

::routine add2stem   /* we can even use a different stem name  */
  use arg abc. /* USE ARG allows to directly refer to the stem */
  n=abc.0+1      /* add after last current entry                 */
  abc.n="Entry #" n "added in add2stem()"
  abc.0=n        /* update total number of entries in stem       */
  return

/* yields:

   # 1: Entry # 1
   # 2: Entry # 2
   # 3: Entry # 3 added in add2stem()

*/
```

14

## About Directives in ooRexx

- Always placed at the end of a Rexx program
  - led in by "::" followed by the name of the directive
    - "routine", "class", "attribute", "method", ...
- Instructions to the ooRexx interpreter before program starts
  - Interpreter sequentially processes and carries out directives in the ***setup** phase* (phase **2**) of startup
  - After all directives got carried out, the *execution phase of the Rexx program* starts by executing the first line
- An ooRexx program with directives
  - Defines a "package" of routines and classes
  - Rexx code before the first directive is also named "prolog"

## ::Routine Directive

- Syntax

  ::routine name [public]

  - Interpreter maintains routines (and classes) per Rexx program ("package")
  - If optional keyword public is present, the routine can be also ***directly** invoked by another (!) Rexx program*

## ::ROUTINE Directive, Example "routine.rex"

```
r=" 1 "
s=2
say "r="pp(r)
say "s="pp(s)
say
say "The result of 'r || 3 ' is:" pp(r || 3 )
say "The result of 's || 3 ' is:" pp(s || 3 )
say "The result of 'r + 3'   is:" pp(r +  3)
say "The result of 's + 3'   is:" pp(s +  3)
say
say "The result of 'r s'     is:" pp(r s)
say "The result of 'r || s'  is:" pp(r || s)
say "The result of 'r+s'     is:" pp(r+s)

::routine pp            -- enclose argument in square brackets
  parse arg value
  return "["value"]"

/* yields:

   r=[ 1 ]
   s=[2]

   The result of 'r || 3 ' is: [ 1 3]
   The result of 's || 3 ' is: [23]
   The result of 'r + 3'   is: [4]
   The result of 's + 3'   is: [5]

   The result of 'r s'     is: [ 1  2]
   The result of 'r || s'  is: [ 1 2]
   The result of 'r+s'     is: [3]
*/
```

## ::ROUTINE Directive, Example "toolpackage.rex"

```
-- collection of useful little Rexx routines

::routine pp    public -- enclose argument in square brackets
  parse arg value
  return "["value"]"

::routine quote public -- enclose argument in double-quotes
  parse arg value
  return '"' || value || '"'
```

## ::ROUTINE Directive, Example "call_package.rex"

```
call toolpackage.rex    -- get access to public routines in "toolpackage.rex"
say quote('hello, my beloved world')

r=" 1 "
s=2
say "r="pp(r)
say "s="pp(s)
say
say "r="quote(r)
say "s="quote(s)
say
say "The result of 'r || 3 ' is:" pp(r || 3 )
say "The result of 's || 3 ' is:" quote(s || 3 )
say "The result of 'r + 3'   is:" pp(r +  3)
say "The result of 's + 3'   is:" quote(s +  3)

/* yields:

   "hello, my beloved world"
   r=[ 1 ]
   s=[2]

   r=" 1 "
   s="2"

   The result of 'r || 3 ' is: [ 1 3]
   The result of 's || 3 ' is: "23"
   The result of 'r + 3'   is: [4]
   The result of 's + 3'   is: "5"
*/
```

## ::REQUIRES Directive

- Syntax

  ::requires "package.rex"

  – Interpreter in (setup) phase 2 will either
    - Call (execute) the Rexx program in the file named "package.rex" on behalf of the current Rexx program and make all its public routines and classes upon return directly available to us
    - *Or* if the interpreter already has *required* that "package.rex" it will *immediately* make all its public routines and classes available to us
      – In this case "package.rex" will ***not*** *be called (executed) anymore!*

## ::REQUIRES-Directive, Example "requires_package.rex"

```
say quote('hello, my beloved world')

r=" 1 "
s=2
say "r="pp(r)
say "s="pp(s)
say
say "r="quote(r)
say "s="quote(s)
say
say "The result of 'r || 3 ' is:" pp(r || 3 )
say "The result of 's || 3 ' is:" quote(s || 3 )
say "The result of 'r + 3'   is:" pp(r +  3)
say "The result of 's + 3'   is:" quote(s +  3)

::requires toolpackage.rex -- get access to public routines in "toolpackage.rex"

/* yields:

   "hello, my beloved world"
   r=[ 1 ]
   s=[2]

   r=" 1 "
   s="2"

   The result of 'r || 3 ' is: [ 1 3]
   The result of 's || 3 ' is: "23"
   The result of 'r + 3'   is: [4]
   The result of 's + 3'   is: "5"
*/
```

## The Message Paradigm, 1

- A programmer sends messages to objects
  - The *object* looks for a method routine with the same name as the received message
  - If arguments were sent the *object* forwards them
  - The *object* returns any value the method routine returns
- *C.f. <https://en.wikipedia.org/wiki/Alan_Kay>*
  - One of the fathers of Smalltalk's "object-orientation"
- Programming languages with this paradigm, e.g.
  - Smalltalk, Objective C, ...

## The Message Paradigm, 2 ooRexx

- Proper message operator "~" (tilde, "twiddle")
- In ooRexx everything is an *"object"*
  - Hence one can send messages to everything!
- Example

```
say "hi, Rexx!"~reverse
-- same as in classic REXX:
say reverse("hi, Rexx!")
-- both yield (actually run the same code):
!xxeR ,ih
```

## The Message Paradigm, 3 ooRexx

- Creating "*values*" a.k.a. "*objects*", "*instances*"

  Classic Rexx-style (strings only)

  ```
  str="this is a string"
  ```

  ooRexx-style (*any* class/type including .string class)

  ```
  str=.string~new("this is a string")
  ```

## About Classic REXX Structures, 1
## Important Usage of Stems

- Whenever structures ("records") are needed, *stems* get used in classic REXX
- Example
  - A person may have a name and a salary, e.g.

```
p.name = "Doe, John"
p.salary= "10500"
```

  - E.g. a collection of data with a person structure

```
p.1.name = "Doe, John"; p.1.salary=10500
p.2.name = "Doe, Mary"; p.2.salary=8500
p.0 = 2
```

## About Classic REXX Structures, 2
## Important Usage of Stems

- Whenever *structures* ("*records*") need to be processed, *every* Rexx programmer *must* know the *exact stem encoding!*
- *Everyone* must implement routines like increasing the salary *exactly* like everyone else!
- If *structures* are simple and not used in many places, this is o.k., but the more complex the more places the *structure* needs to be accessed, the more error prone this becomes!

## About ooREXX *Structures*, 1

### Classes (Types, Structures)

- Any object-oriented language makes it easy to define and implement *structures*!
  - That is what they were designed for!
- The *structure* ("*class*", "*type*") usually consists of
  - *Attributes* (data elements like "name", "salary"), a.k.a. "*object variables*", "*fields*", ...
  - *Method* routines (like "increaseSalary")

## About ooREXX *Structures*, 2

### Classes (Types, Structures)

- ::CLASS Directive
  - Denotes the name of the *structure*
  - Can optionally be public
- ::ATTRIBUTE Directive
  - Denotes the name of a *data element, field*
- ::METHOD Directive
  - Denotes the name of a routine of the *structure*
  - Defines the *Rexx code* to be run, when invoked

## About ooREXX *Structures*, 3

## Classes (Types, Structures)

- Once
  - A *structure* ("*class*", "*type*" both of which are synonyms of each other) got defined
  - One can create an *unlimited (!) number* of persons ("*instances*", "*objects*", "*values*", all of which are synonyms)
    - *Each* person will have its *own copy of attributes (data elements, fields)*
    - *All* persons will share/use the *same method routines* that got defined for the structure (class, type)

## ooRexx Structure "Person"

## "personstructure.rex"

```
p=.person~new("Doe, John", 10500)
say "name:  " p~name
say "salary:" p~salary

::class person              -- define the name

::attribute name            -- define a data element, field, object variable
::attribute salary          -- define a data element, field, object variable

::method    init            -- constructor method routine (to set the attribute values)
  expose name salary        -- establish direct access to attributes
  use arg name, salary      -- fetch and assign attribute values

/* yields:

   name:   Doe, John
   salary: 10500

*/
```

## Defining the ooRexx Class (Type) "person.cls"

```
::class person PUBLIC    -- define the name, this time PUBLIC

::attribute name         -- define a data element, field, object variable
::attribute salary       -- define a data element, field, object variable

::method    init         -- constructor method routine (to set the attribute values)
  expose name salary     -- establish direct access to attributes
  use arg name, salary   -- fetch and assign attribute values
```

## Defining the ooRexx Class (Type) "requires_person.rex"

```
p.1 = .person~new("Doe, John", 10500)
p.2 = .person~new("Doe, Mary",  8500)
p.0 = 2

sum=0
do i=1 to p.0
   say p.i~name "earns:" p.i~salary
   sum=sum+p.i~salary
end
say
say "Sum of salaries:" sum

::requires person.cls    -- get access to the public class "person" in "person.cls"

/* yields:

   Doe, John earns: 10500
   Doe, Mary earns: 8500

   Sum of salaries: 19000
*/
```

## ooRexx *Classes* and Beyond ...

- ooRexx comes with a wealth of *classes*
  - A lot of tested functionality for "free" ;-)
  - E.g., the collection classes augment what stems are capable of doing!
    - Explore the collection classes and you will immediately be much more productive!
    - If seeking arrays, you have them: .Array class
  - Consult the pdf-books coming with ooRexx, e.g.,
    - "ooRexx Programming Guide" (rexxpg.pdf)
    - "ooRexx Reference" (rexxref.pdf)

33

## Roundup

- ooRexx is great and compatible to classic REXX
  - You can continue to program in classic REXX, yet use ooRexx on Linux, MacOS, Windows, s390x...
- ooRexx adds a lot of flexibility and power to the REXX language and to your fingertips
  - One can take advantage of all of it immediately
  - Simple to use because of the *message paradigm*
    - Send ooRexx *messages* to Windows and MS Office ...
    - Send ooRexx *messages* to Java ...
    - Send ooRexx *messages* to ...
- ***Get it and have fun! :-)***

34

## Links

- RexxLA-Homepage (non-profit SIG, owner of ooRexx, BSF4ooRexx)
  <http://www.rexxla.org/>
- OoRexx 5.1.0 on Sourceforge
  <https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0/>
  - Introduction to ooRexx on Windows, Slides ("Business Programming 1")
    - <http://wi.wu.ac.at/rgf/wu/lehre/autowin/material/foils/>
- BSF4ooRexx850 on Sourceforge (ooRexx-Java bridge)
  <https://sourceforge.net/projects/bsf4oorexx/>
  - Introduction to BSF4ooRexx (Windows, Mac, Unix), Slides ("Business Programming 2")
    - <http://wi.wu.ac.at/rgf/wu/lehre/autojava/material/foils/>
- Student's work, including ooRexx, BSF4ooRexx
  <http://wi.wu.ac.at/rgf/diplomarbeiten/>
- JetBrains "IntelliJ IDEA", powerful IDE for all operating systems
  - <https://www.jetbrains.com/idea/download>, free "Community-Edition"
    - Students and lecturers can use the professional edition for free
  - Alexander Seik's ooRexx-Plugin with readme (as of: 2025-05-07)
    - <https://sourceforge.net/projects/bsf4oorexx/files/Sandbox/aseik/ooRexxIDEA/GA/2.5.0/>
- "Introduction to Rexx and ooRexx" (254 pages, covers ooRexx 4.2)
  Google et.al., or, <https://www.facultas.at>

# 2

# Meet the Message Paradigm – Rony G. Flatscher

## Date and Time

4 May 2025, 12:15:00 GMT

## Presenter

Rony G. Flatscher

## Presenter Details

Rony works as a professor for Business informatics (”Wirtschaftsinformatik”) at the Vienna University of Economics and Business Administration (Wirtschafts-universität Wien) and uses Open Object Rexx for teaching Business Administration and MIS students the object-oriented paradigm, as well as remote-controlling (automating) Windows and Windows end-user applications (e.g. MS Office, Open Office) as well as Java and Java applications (he is the author of BSF4ooRexx, the ooRexx-Java bridge, which uses Apache BSF and had Rony invited to become an ASF member). He consults and trains in all of his research fields.

## Session Abstract

ooRexx introduces and implements the message paradigm (inspired by SmallTalk), making it easy for programmers to conceptually interact with objects of any complexity and environment. This tutorial explains and demonstrates messages using short, nutshell examples, thereby explaining how ooRexx’s object-oriented features work. It should become understandable why it is easy for beginners to employ the message paradigm successfully and to understand important object-oriented concepts like method resolution and inheritance.

# Meet the Message Paradigm

International Rexx Symposium
May 4th through May 7th 2025, Vienna

*I'm sorry that I long ago coined the term "objects" for this topic because it gets many people to focus on the lesser idea. The* ***big idea is "messaging".***

**Alan Kay (https://en.wikipedia.org/wiki/Alan_Kay)**

## Developing Business Programming

- Specialisation in **"(Business) Information Systems"**
  - As customary at the time, the *most popular languages* were used to teach beginners: Pascal, BASIC, COBOL, C, PROLOG, Visual Basic Script (VBS) / Applications (VBA), Java, ...
- **Surprise** when experimenting with the Rexx programming language
  - Novices learn ***much faster and more in-depth*** than with popular languages
  - Analysing the **critical success** factors showed that the most important aspect was **the programming language**
- 35 years of **participant observation** (two lectures per semester)
  - Observed difficulties yielded changes in: content, slides, nutshell examples, infrastructure, presentation, ...

## Some Historical Bits on Rexx

- Created for IBM mainframes to make programming easier compared to the rather awkward EXEC2
  - **Rexx design goals:** "human centric", "keep the language small", "easy to learn", "easy to understand hence easy to maintain"
  - Rexx is **still instrumental for IBM mainframe operating systems** today!
- Extremely successful in the 80'ies
  - Companies selling Rexx interpreter successfully, **ANSI/INCITS standard** (!)
- Object-oriented successor ("Object Rexx") in the 90'ies
  - **Open-sourced** in 2005 by RexxLA.org – "open object Rexx" (ooRexx)
    - Available for **all major operating systems**
    - Possible to program even MS Windows applications via OLE ...

## Fundamental Rexx Concepts, 1

- "Everything is a string"
  - If a string represents a number, one can carry out arithmetic's
- Three instruction types:
  - 1) Assignment
    - Variable name followed by the assignment operator (=) and an expression
  - 2) Keyword instruction
    - Keywords are English words conveying the intent of the keyword instruction, e.g. SAY, DO, IF, LOOP, CALL, PARSE, SELECT, ITERATE, LEAVE, INTERPRET, ...
    - Makes Rexx code legible as if it was pseudo code
  - 3) Commands
    - A string passed to the operating system for execution (as if typed in a window)

## Fundamental Rexx Concepts, 2

- White space can be freely used to format code for better legibility
  - Space around operators gets removed
  - White space between symbols will be reduced to a single space serving as abuttal concatenation operator
  - Hence indentations with white space not significant
- Case of symbols irrelevant
  - Rexx uppercases everything outside of quoted strings
  - No (frustrating) casing errors for novices
- Rexx nutshell examples to stress fundamental concepts
  - Illustrate the language
  - Same  examples in the popular Python language to allow direct comparisons

## Nutshell Example
## "Instructions"

```
    /* an assignment instruction:      */
a="hello world"   /* assigns "hello world" to a variable named a  */

    /* a keyword instruction:          */
say a               /* output: hello world */

    /* a command instruction:          */
    /* a command (could be typed into a command line window)        */
"echo Hello World 2" /* execute command                             */
   /* variable RC contains the command's return code, 0 means success */
if rc=0 then say "success!"
        else say "some problem occurred, rc="rc /* show return code   */
```

**Output:**

```
hello world
Hello World 2
Success!
```

```
# an assignment instruction
a="hello world"     # assigns "hello world" to a variable named a

# no keyword instruction for output, using built-in function print()
print(a)

# no command instruction using module subprocess instead
import subprocess   # import subprocess module
# execute command
completedProcess=subprocess.run("echo Hello World 2", shell=True) # run
rc=completedProcess.returncode  # fetch return code, an int
if rc==0:
    print("found!") # indentation mandatory (forcing a block)
else: # must use + (concatenation operator) with str() function
    print("some problem occurred, rc="+str(rc)) # turn rc into a string
```

**Output:**

```
hello world
Hello World 2
Success!
```

## Concepts Added by ooRexx, 1

- ooRexx has been influenced by SmallTalk including its **message paradigm**
- ooRexx adds *message expressions* and *directive instructions*
- "In ooRexx everything is an *object* (synonyms: *value*, *instance*)"
  - An object is conceptually regarded as if it was a living thing
  - One can only interact with an object by sending it *messages*
- A *message expression* consists of a *receiver*, the message operator **~** (tilde) and the *message name*, optionally followed by arguments in parentheses
  - The *receiver* will search a method by the name of the received message, invokes it and returns any result to the sender
  - No one can invoke methods directly but the *receiver* (encapsulation)!
  - The *sender* does not need to know anything about implementation details

Nutshell Example

## Messages

```
say reverse("olleh")     -- classic Rexx BIF (built-in function)

say "olleh"~reverse      -- message to string object
```

**Output:**

```
hello
hello
```

```
a="dlrowolleh"     -- assign string to variable
   -- use built-in-functions (BIFs) reverse(), substr()
say substr(reverse(a),1,5) substr(reverse(a),6)

   -- use String methods reverse and substr
say a~reverse~substr(1,5) a~reverse~substr(6)
```

**Output:**

```
hello world
hello world
```

## Concepts Added by ooRexx, 2

- Directive instruction
  - If present then always placed at the end of a program
  - Led in by two consecutive colons (**::**) serving as an eye catcher
    - Directives can be used to cause ooRexx to create classes with attributes and methods during the setup phase
      - ::CLASS name, ::ATTRIBUTE name, ::METHOD name, ...
- Classes with attributes and methods
  - Can be defined with directive instructions or dynamically at runtime
  - Instances get created by sending the class the message new
    - The new method will create the object and before returning it, the newly created object gets the message init sent with the arguments supplied to the new message, if any
      - Hence, defining a method named init will always run at construction time (constructor)

Nutshell Example

## Creating A Class with Directives and Dynamically

```
say ".dog:" .dog      -- string value of the class
d=.dog~new            -- create and assign a dog
d~bark                -- let the dog bark
say "d:" d", an instance of:" d~class

::class dog           -- class directive
::method bark         -- method routine directive
  say "wuff!"         -- code to run
```

**Output:**

```
.dog: The DOG class
wuff!
d: a DOG, an instance of: The DOG class
```

Dynamic creation

```
clz=.object~subclass("DOG")    -- create the dog class
say "clz:" clz -- string value of the class
m  =.method~new("bark", 'say "wuff!"') -- create method
clz~define("bark",m) -- define as instance method for class

d=clz~new       -- create and assign a dog
d~bark          -- let the dog bark
say "d:" d", an instance of:" d~class
```

**Output:**

```
clz: The DOG class
wuff!
d: a DOG, an instance of: The DOG class
```

## Ad Messages, 1

- Quickly familiar, intuitive for novices
- Seeing **objects as living things** makes it easy to accept behaviours and concepts like
  - The new method of a class will send the init message to the newly created object (a method named init is therefore a constructor)
  - An object using the *class hierarchy* to locate the method to invoke (inheritance)
  - *Multiple inheritance* (!) deviating the search carried out by the object
  - Intercepting messages for which no method could be found as the object then sends the unknown message to itself (simply implement a method unknown)
  - The variables self (reference to the object that invoked the method) and super (reference to the immediate superclass) in methods
  - As objects know how to find and invoke methods, the sender does not need to know that (black box) at all, alleviating the (novice) programmer

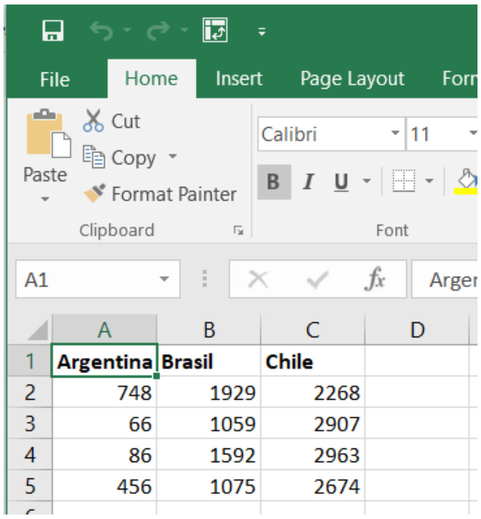
## Ad Messages, 2

- Addressing complex software infrastructures can be made easy for message senders (programmers)
  - Create a proxy class in ooRexx for the sender that processes the received messages, marshals the received arguments and unmarshals the return value
- Example Windows and Windows programs
  - ooRexx for Windows has ooRexx classes for Windows support
  - The ooRexx OLEObject class is the proxy class for interacting via OLE (Object Linking and Embedding) with *any* OLE Windows component
    - Its unknown method will intercept all messages for which no method can be found on the ooRexx side, such that it gets forwarded to the proxied Windows object by searching and invoking the appropriate Windows method
    - To exploit this functionality no implementation knowledge of COM or OLE is needed!

Nutshell Example

## Programming Excel Using ooRexx Messages

```
excApp = .OLEObject~new("Excel.Application") -- create Excel object
excApp~visible = .true           -- make Excel visible
sheet  = excApp~Workbooks~Add~Worksheets[1]  -- add and get sheet
    -- set titles from an ooRexx array
titleRange=sheet~range("A1:C1")  -- get title cell range
titleRange~value      = .array~of("Argentina", "Brasil", "Chile")
titleRange~font~bold = .true     -- make font bold
sheet~range("A2:C5")~value = createRows(4)   -- create and assign array
excApp~displayAlerts = .false    -- no alerts (should file exist already)
fileName=directory()"\test.xlsx" -- save in current directory
Say 'fileName:' fileName         -- show fully qualified file name
sheet~SaveAs(fileName)           -- save file (no alerts, see above)
excApp~quit                      -- quit (end) Excel

::routine createRows    -- return two-dimensional array with random data
  use arg items         -- fetch argument
  arr=.array~new        -- create Rexx array
  do i=1 to items       -- create random(min,max) numbers
     arr[i,1] = random(    0,1000)    -- Argentina
     arr[i,2] = random(1001,2000)    -- Brazil
     arr[i,3] = random(2001,3000)    -- Chile
  end
  return arr            -- return two-dimensional Rexx array
```



|   | A | B | C | D |
|---|---|---|---|---|
| 1 | **Argentina** | **Brasil** | **Chile** | |
| 2 | 748 | 1929 | 2268 | |
| 3 | 66 | 1059 | 2907 | |
| 4 | 86 | 1592 | 2963 | |
| 5 | 456 | 1075 | 2674 | |

**Possible Output:**

```
fileName: C:\Program Files\JetBrains\IntelliJ IDEA 2023.3.6\jbr\bin\test.xlsx
```

## Ad Messages, 3

- Addressing complex software infrastructures can be made easy for message senders (programmers)
  - Create a proxy class in ooRexx for senders that processes the received messages, marshals the received arguments and unmarshals the return value
- Example Java and Java class libraries
  - BSF4ooRexx850 for Windows, macOs and Linux implements an ooRexx-Java bridge
  - Its BSF class is the ooRexx proxy class for interacting with Java
    - Its unknown method will intercept all messages for which no method can be found on the ooRexx side, such that it gets forwarded to the proxied Java object by searching and invoking the appropriate Java method
    - To exploit this functionality no implementation knowledge of BSF4ooRexx850 is needed!

Nutshell Example

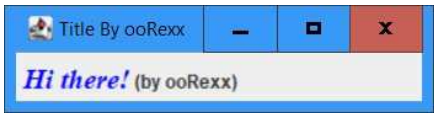
## Communicating with Java Objects Using ooRexx Messages

```
dim=.bsf~new("java.awt.Dimension",111,222)
say "dim:          " dim", dim~class:" dim~class
say "dim~toString:" dim~toString -- Java method
    -- use Java fields as if ooRexx attributes
say "dim~width:    " dim~width    -- Java field
say "dim~height:   " dim~height   -- Java field
dim~setSize(333,444) -- Java method
say "dim~toString:" dim~toString -- Java method
    -- use Java fields as if ooRexx attributes
dim~width=555         -- setting Java field
dim~height=666        -- setting Java field
say "dim~toString:" dim~toString -- Java method

::requires "BSF.CLS" -- get ooRexx-Java bridge
```

**Output:**

```
dim:          java.awt.Dimension@1c4af82c, dim~class: The BSF
class
dim~toString: java.awt.Dimension[width=111,height=222]
dim~width:    111
dim~height:   222
dim~toString: java.awt.Dimension[width=333,height=444]
dim~toString: java.awt.Dimension[width=555,height=666]
```

```
jf  = .bsf~new("javax.swing.JFrame", "Title By ooRexx")  -- create JFrame
style   = 'style="color: blue; font-family: serif; font-size: 18;"'
lblText = '<html><em' style'>&nbsp;Hi there!</em> (by ooRexx) </html>'
lbl = .bsf~new("javax.swing.JLabel", lblText)   -- create JLabel
jf~add(lbl)                -- add JLabel to JFrame
jf~setSize(280,70)         -- set size
jf~setLocation(50,200)     -- set JFrame's location on screen
jf~visible=.true           -- make JFrame visible
jf~toFront                 -- place JFrame in front of all windows
say 'Hit <enter> on the keyboard to proceed (end) ...'
parse pull data            -- wait until user presses <enter>

::requires "BSF.CLS"       -- get ooRexx-Java bridge
```



**Output:**

```
Hit <enter> on the keyboard to proceed (end) ...
```

## Roundup

- Message paradigm
  - **Easy and intuitive** (easy for novices as well)
  - All important object-oriented concepts can be informally (!) explained and understood (easy to understand for novices as well)
- **Proxy classes** allow **the message paradigm to be extended to other software systems**
  - Windows COM/OLE, proxy class OLEObject (supplied by ooRexx)
  - Java, proxy class BSF (supplied by BSF4ooRexx850)
  - **interestingly, novice students do not care and are not afraid! :-)**
    - They "only" send messages and need not know any implementation details!
    - The supplied nutshell examples allow novices to exploit OLE and Java
      - Windows: MS Excel, MS Word, MS PowerPoint, AOO swriter, LO scalc, ...
      - Java: from (secure!) socket programming to JavaFX GUIs!

## Some References

- **Open and free slides** (odp upon request)
  - R. G. Flatscher, “Introduction to Programming with ooRexx and BSF4ooRexx 1. 1-7.” [PDF slides]:
    - <https://wi.wu.ac.at/rgf/wu/lehre/autowin/material/foils/>
  - R. G. Flatscher, “Introduction to Programming with ooRexx and BSF4ooRexx 2. 8-14.” [PDF slides]:
    - <https://wi.wu.ac.at/rgf/wu/lehre/autojava/material/foils/>
- T. Winkler, "Collection of Rexx References". <https://wi.wu.ac.at/rgf/rexx/rexxref/searchref.html>
  - Maintained at: <https://gitlab.com/dylwi/rexx-references>
- R. G. Flatscher and G. Müller, “'Business Programming' – Critical Factors from Zero to Portable GUI Programming in Four Hours,” in 6th BEE-Conference, Plitvice Lakes, Croatia, 2021, pp. 76-82.
  - <https://research.wu.ac.at/files/32933925/2021_BusinessProgramming_BEE2021_accordingToGuidelines.pdf>
- R.G. Flatscher, "Proposing ooRexx and BSF4ooRexx for Teaching Programming and Fundamental Programming Concepts", in 2023 Program Guide ISECON: Information Systems Education Conference, Dallas/Plano, Tx, 2023, pp. 89-102.
  - <https://research.wu.ac.at/files/41301564/ISECON23_Flatscher_Proposing_ooRexx_article.pdf>
- T. Winkler and R. G. Flatscher, “Cognitive Load in Programming Education: Easing the Burden on Beginners with REXX.” In Central European Conference on Information and Intelligent Systems. 2023, pp. 171-178.
  - <https://research.wu.ac.at/files/46150789/CECIIS_CLT_REXX.pdf>

## Some Links

- Portable zip archives (no installation needed): ooRexx 5.1.0, oorexxshell, dbusoorexx, bsf4oorexx
  - <https://www.ronyrexx.net/xfer/portable>
    - Note: bsf4oorexx (ooRexx-Java bridge) needs Java installed
- Installation packages
  - ooRexx 5.1.0:
    - <https://sourceforge.net/projects/oorexx/files/oorexx/5.1.0>
  - BSF4ooRexx (ooRexx-Java bridge, needs Java preinstalled):
    - <https://sourceforge.net/projects/bsf4oorexx/files/GA/BSF4ooRexx-850.20240304-GA/>
- Selected seminar papers, Bachelor and Master thesis with ooRexx, BSF4ooRexx, dbusoorexx
  - <https://wi.wu.ac.at/rgf/diplomarbeiten/>
- Non-profit Rexx Language Association (owner of ooRexx):
  - <https://www.RexxLA.org>
- Web page with Rexx related resources maintained by R.G. Flatscher:
  - <https://ronyrexx.net>

3

# Introduction to BSF4ooRexx850 (ooRexx/Java Language Bindings) – Rony G. Flatscher

## Date and Time

4 May 2025, 13:15:00 GMT

## Presenter

Rony G. Flatscher

## Presenter Details

Rony works as a professor for Business informatics ("Wirtschaftsinformatik") at the Vienna University of Economics and Business Administration (Wirtschafts-universität Wien) and uses Open Object Rexx for teaching Business Administration and MIS students the object-oriented paradigm, as well as remote-controlling (automating) Windows and Windows end-user applications (e.g. MS Office, Open Office) as well as Java and Java applications (he is the author of BSF4ooRexx, the ooRexx-Java bridge, which uses Apache BSF and had Rony invited to become an ASF member). He consults and trains in all of his research fields.

## Session Abstract

The "Bean Scripting Framework for ooRexx bridges ooRexx and Java. It allows ooRexx programs to use all Java classes and Java objects as if they were ooRexx classes and ooRexx objects. This way, it becomes possible to take full advantage of all the functionality Java classes offer in a platform-independent manner. Among other things, it enables ooRexx programs to create and use the most complex graphical user interface applications without needing to learn Java or write Java code. Nutshell examples will demonstrate how easy ooRexx programmers can exploit all of Java. (The bridge also allows Java programmers to send Rexx messages to ooRexx objects.)

**Department of Information Systems
and Operations Management**

# IntroductionToBSF4ooRexx850

ooRexx/Java Language Bindings

**Easily exploiting Java from ooRexx on all operating system platforms**

The 2025 International Rexx Symposium
Vienna, Austria
May4th – May 7th 2025

## Overview

- Some information on Java and an example of using ooRexx to exploit it
- Some important things to know about Java
- Introducing the ooRexx package (program) BSF.CLS
  - Camouflages Java as ooRexx
  - Makes it possible to simply send ooRexx messages to Java (class) objects
  - Provides some important utility features
- Download links
- Roundup
- *Addenda!*
  - *Also demonstrates how Java can send ooRexx objects messages!*

## Java

- Programming language with the following notable features
  - Compiles to machine instructions (*"bytecode"*) of an *artificial processor*
  - Needs a "Java virtual machine (JVM)" to execute the bytecode
    - JVMs are available for all important operating systems and hardware architectures
    - *Hence, a Java class or a Java program, once compiled can be run everywhere!*
  - Distributed with a (huge) "Java runtime environment (JRE)"
    - A *huge Java class library* that offers everything that an application may possibly need
      - E.g. Socket classes for Internet programming, GUI classes for graphical user interfaces, ...
    - Uncountable third party Java class libraries, most available as open-source (e.g. ASF)
  - Most important programs get programmed with Java (even Android applications!)
  - Many professional applications that are not programmed in Java offer Java APIs
    - E.g. SAP, OpenOffice/LibreOffice, ...
- Hence Java is truly a programmer's "treasure trove" for all operating systems!

## BSF4ooRexx850

- External Rexx function package
  - Allows to interact with the Java runtime environment (JRE)
    - Exploit functionality of Java classes
    - Exploit functionality of Java objects
  - ooRexx 5.0 or later, Java 8 or later
  - Package "BSF.CLS"
    - Camouflages Java as ooRexx (Java appears to be dynamic and message based)
    - Supplies class BSF and public routines
- "Everything that is available in Java becomes directly available to ooRexx !"
  - Java: "write once, run everywhere!"
    - Windows, MacOS, Linux, ...

## BSF4ooRexx: An Example, 1

- The following example
  - Uses the ::requires directive to load the ooRexx-Java bridge
    `::requires "BSF.CLS"`
    - Directives get processed in the setup phase, right before the program starts
  - Creates an instance of the Java class named java.awt.Dimension and interacts with it via ooRexx messages that denote the method names to run
    - Studying the documentation of the Java class java.awt.Dimension one can see which Java methods are available for use
  - Displays the string that the message toString returns
  - Changes the values for the width and height fields
  - Displays the string that the message toString returns

## BSF4ooRexx: An Example, 2

```
dim=.bsf~new("java.awt.Dimension", 100, 200)     -- create with width and height
say dim~toString                                 -- show string value

::requires BSF.CLS      -- get Java support
```

Output:

```
java.awt.Dimension[width=100,height=200]
```

## Downloading Java (Usually Free and Open-source)

- JRE versus JDK
  - JRE: "**J**ava **R**untime **E**nvironment", no compiler
  - JDK: "**J**ava **D**evelopment **K**it", compiler & tools
- Java/OpenJDK 8 LTS ("long term support")
  - Released spring 2014, supported until 2030 (Oracle, Azul), 2031 (Liberica)
- Java/OpenJDK 21 LTS ("long term support", "modular Java")
  - Released fall 2023, supported at least until 2031 (Oracle, Azul), 2031 (Liberica)
- Suggestion: download OpenJDK *with JavaFX* support, e.g.
  - Scroll down to see all versions pick the *JavaFX* installation package
    - **Full JDK:** <https://bell-sw.com/pages/downloads/> ("Liberica", 2025-04-28)
    - **JDK FX:** <https://www.azul.com/downloads/> ("Azul", 2025-04-28)

## Things to Know About Java, 1

- Strictly typed language
  - Primitive types
    - boolean, byte, char, short, int, long, float, double
  - Object-oriented types
    - Any Java class, e.g.
      - `java.awt.Dimension, java.lang.String, java.lang.System, ...`
    - Wrapper classes for primitive types
      - `java.lang.Boolean, java.lang.Byte, java.lang.Character, java.lang.Short, java.lang.Integer, java.lang.Long, java.lang.Float, java.lang.Double`
      - "boxing": wraps up a primitive value into a wrapper object
      - "unboxing": retrieves a primitive value from its wrapper object

## Things to Know About Java, 2

- Case sensitive
  - Upper- and lowercase significant!
- Classes organized in packages
  - Package names may be compound
    - E.g. "java.lang"
  - Fully "qualified class name" includes package name
    - e.g. "java.lang.String"
  - "Unqualified class name"
    - e.g. "String"

## Things to Know About Java, 3

- A Java class may consist of
  - Fields (comparable to ooRexx attributes) and
  - Methods (comparable to ooRexx methods)
- Fields and methods
  - Static fields and static methods
    - Sometimes dubbed "class fields" and "class methods"
    - Available to the class object *and* its instances
  - Otherwise "instance methods"
    - Only available to instances of a Java class

## Things to Know About Java, 4

- A Java class, its fields and methods may be
  - "public"
    - These can be accessed by the "world" (everyone)
  - "private"
    - Only accessible within the Java class
  - "protected"
    - Only accessible within Java classes of the same package and subclasses
  - None of the above modifiers given ("package private")
    - Only accessible within Java classes of the same package, but to noone else

## Things to Know About Java, 5

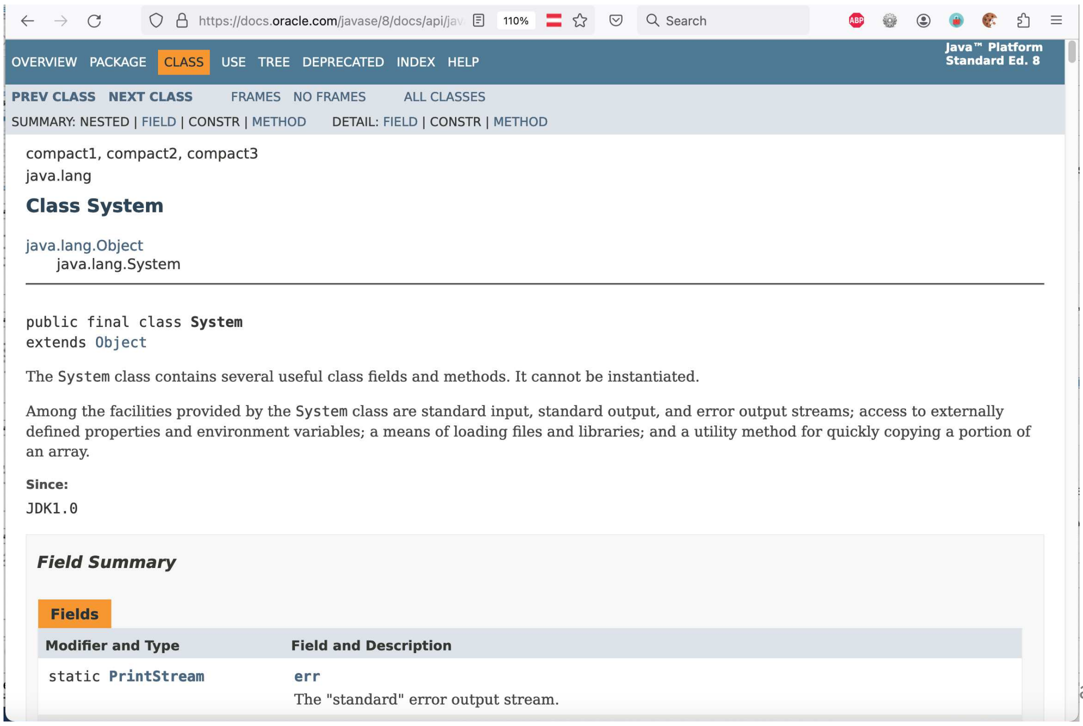
- Excellent documentation ("JavaDoc")
  - Extensive set of interlinked HTML documents
    - Created right from the comments in Java sources
  - Can be studied on the Internet, search e.g. with

```
javadoc 8 java.awt.Dimension
javadoc 8 Dimension
javadoc 21 java.awt.Dimension
javadoc 21 Dimension
```

- Documentation can be downloaded to local computer, e.g.
  - Java/JDK 8 LTS ("long term support"):
    - <https://www.oracle.com/java/technologies/javase-jdk8-doc-downloads.html> (2025-04-28)
  - Java/JDK 21 LTS ("long term support"):
    - <https://www.oracle.com/java/technologies/javase-jdk21-doc-downloads.html> (2025-04-28)

## A Javadoc Example (JDK8LTS), 1

Search keywords:
**Javadoc 8 System**


https://docs.oracle.com/javase/8/docs/api/jav
110%
Search
OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP
Java™ Platform Standard Ed. 8
PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

compact1, compact2, compact3
java.lang

**Class System**

java.lang.Object
    java.lang.System

```
public final class System
extends Object
```

The System class contains several useful class fields and methods. It cannot be instantiated.

Among the facilities provided by the System class are standard input, standard output, and error output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.

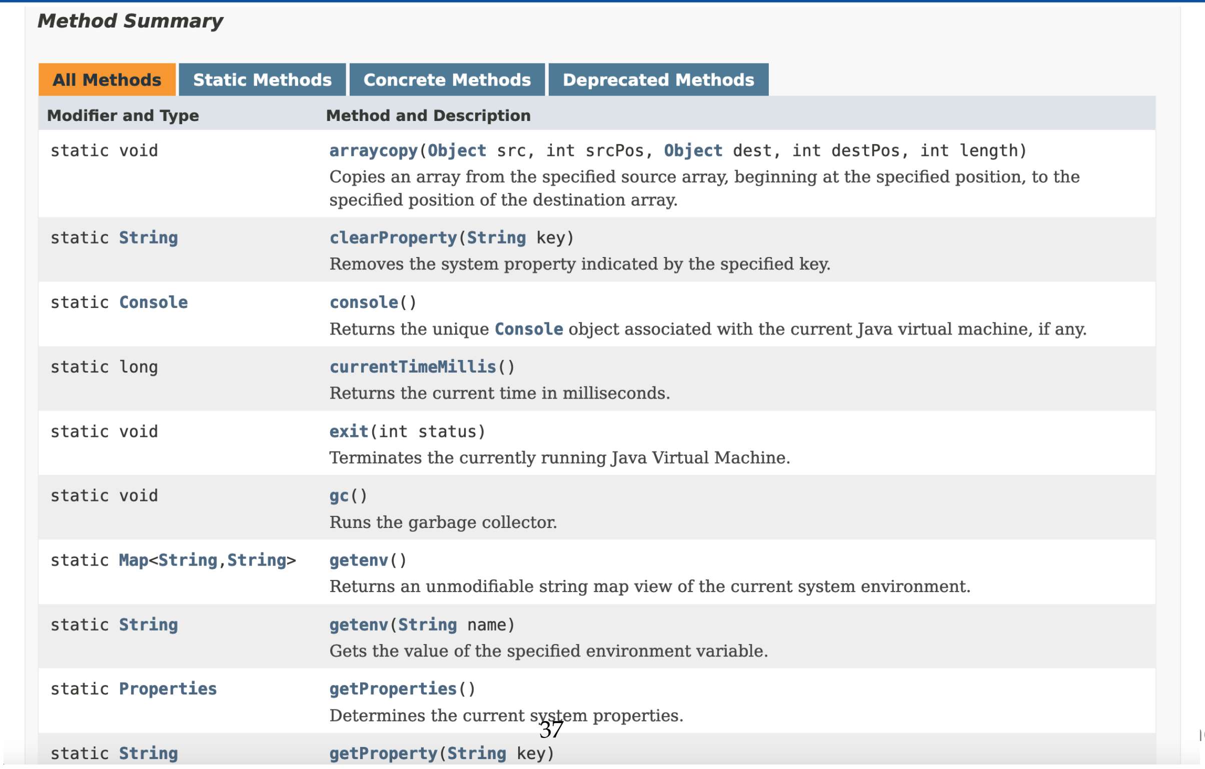
**Since:**
JDK1.0

***Field Summary***

**Fields**

| Modifier and Type | Field and Description |
|---|---|
| static PrintStream | err<br>The "standard" error output stream. |

## A Javadoc Example (JDK8LTS), 2



***Method Summary***

| Modifier and Type | Method and Description |
|---|---|
| static void | arraycopy(Object src, int srcPos, Object dest, int destPos, int length)<br>Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array. |
| static String | clearProperty(String key)<br>Removes the system property indicated by the specified key. |
| static Console | console()<br>Returns the unique Console object associated with the current Java virtual machine, if any. |
| static long | currentTimeMillis()<br>Returns the current time in milliseconds. |
| static void | exit(int status)<br>Terminates the currently running Java Virtual Machine. |
| static void | gc()<br>Runs the garbage collector. |
| static Map<String,String> | getenv()<br>Returns an unmodifiable string map view of the current system environment. |
| static String | getenv(String name)<br>Gets the value of the specified environment variable. |
| static Properties | getProperties()<br>Determines the current system properties. |
| static String | getProperty(String key) |

## BSF.CLS: *Camouflages Java* as ooRexx

- ooRexx proxy class "**BSF**"
  - Allows to create Java objects
  - Requires the fully qualified Java class name
- Invoking Java methods
  - Just send the name of the method as a message to the Java object
    - Supply the arguments as documented, if any
      - Type conversions between ooRexx and Java are done automatically by BSF4ooRexx, if necessary
      - Return values are automatically converted by BSF4ooRexx, if necessary

## BSF.CLS: Creating Java Objects

- ooRexx proxy class "**BSF**"
  - Allows to create Java objects
  - Needs at least fully qualified Java class name
- Possible arguments for creating Java objects
  - Can be found by studying the "*Constructor*" section in the Javadocs
  - Supply the arguments as documented after the fully qualified Java class name argument
    - Type conversions ("marshalling") between ooRexx and Java are done automatically by BSF4ooRexx, if necessary

## BSF.CLS: Creating Java Objects, Example

```
-- see Javadocs: search Internet with "javadoc java.awt.Color"
red=.bsf~new("java.awt.Color",255,0,0)       -- create color red
say "red:" red~toString -- toString will show the RGB values

myColor=.bsf~new("java.awt.Color",100,200,3) -- create an individual color
say "myColor:" myColor~toString
brighter=myColor~brighter    -- get a brighter color
say "brighter:" brighter~toString

::requires "BSF.CLS"    -- get ooRexx-Java bridge
```

Output (maybe):

```
red: java.awt.Color[r=255,g=0,b=0]
myColor: java.awt.Color[r=100,g=200,b=3]
brighter: java.awt.Color[r=142,g=255,b=4]
```

## BSF.CLS: *Camouflages Java* as ooRexx

- Allows to load any Java class
  - **bsf.loadClass(JavaClassName)**
    - Java class name
      - Use of the exact case is mandatory !
      - Java class name must be fully qualified !
- Allows accessing static (class) methods and fields (attributes)
  - Example uses java.lang.System's static getProperty() method to query the Java version from ooRexx

## BSF.CLS: *Loading* a Java Class, Example

```
-- see Javadocs: search Internet with "javadoc java.lang.System"
clz=bsf.loadClass("java.lang.System")    -- loads the Java class
say "java.version:" clz~getProperty("java.version")

::requires "BSF.CLS"    -- get ooRexx-Java bridge
```

Output (maybe):

```
java.version: 1.8.0_162
```

## BSF.CLS: *Camouflages Java* as ooRexx

- Allows to import any Java class
  - **bsf.import(JavaClassName)**
    - Java class name
      - Use of the exact case is mandatory !
      - Java class name must be fully qualified !
- Imported Java class can be treated as if it were an ooRexx class
  - Allows to use the ooRexx "**new**"-method to create instances of the imported Java class
    - Possible arguments for creating Java objects can be found by studying the "Constructor" section in the Javadocs

## BSF.CLS: *Importing* a Java Class, Example

```
-- see Javadocs: search Internet with "javadoc java.awt.Color"
clzColor=bsf.importClass("java.awt.Color")  -- import Java class

red=clzColor~red         -- get static field for red color
say "red:" red~toString -- toString will show the RGB values

myColor=clzColor~new(100,200,3) -- create an individual color
say "myColor:" myColor~toString
brighter=myColor~brighter    -- get a brighter color
say "brighter:" brighter~toString

::requires "BSF.CLS"   -- get ooRexx-Java bridge
```

Output (maybe):

```
red: java.awt.Color[r=255,g=0,b=0]
myColor: java.awt.Color[r=100,g=200,b=3]
brighter: java.awt.Color[r=142,g=255,b=4]
```

## BSF.CLS: *Camouflages Java* as ooRexx

- Accessing, setting Java fields
  - ooRexx treats public fields as ooRexx attributes
  - Java "get" and "set" pattern methods for Java fields honored by BSF4ooRexx
    - Just use the field name following "get" and "set" only
  - Static fields can be accessed via the
    - Java class object or
    - Any of its instances

## BSF.CLS: *Java Fields As ooRexx Attributes*

```
-- see Javadocs: search Internet with "javadoc java.awt.Dimension"
dim=.bsf~new("java.awt.Dimension", 100, 200)
say dim~toString
dim~height=321        -- treat field height as if it was an ooRexx attribute
dim~width =1024       -- treat field width as if it was an ooRexx attribute
say dim~toString

::requires BSF.CLS       -- get Java support
```

Output:

```
java.awt.Dimension[width=100,height=200]
java.awt.Dimension[width=1024,height=321]
```

## BSF.CLS

- About respecting case
  - Case of fully qualified Java class name
    - Always significant!
- Case of fields and method names insignificant!
  - Eases coding considerably

## **BSF.CLS**: Creating Java Arrays, 1

- Java arrays
  - Strictly typed
  - Fixed capacity
  - Indices start with value "**0**"
- Public routine "**bsf.createJavaArray(...)**"
  - Arguments
    - First argument gives the Java type
      - Fully qualified Java class name or Java class object
    - Each further argument is an integer value, denoting the maximum elements in that dimension

## **BSF.CLS**: Creating Java Arrays, 2

- Public routine "**bsf.createJavaArray(...)**"
  - Resulting Java array can be used as if it was an ooRexx array object!
    - Indices start at "**1**" as with ooRexx arrays!
    - Possesses the fundamental *ooRexx array methods* like "**AT**", "**[]**", "**PUT**", "**[]=**", "**supplier**", and "**makeArray**"
    - Can be therefore used in ooRexx "**DO ... OVER**" and "**DO WITH ... OVER**" loops

## BSF.CLS: Creating a Java Array

```
-- create a two-dimensional (5x10) Java Array of type String
arr=.bsf~bsf.createJavaArray("java.lang.String", 5, 10)

arr[1,1]="First Element in Java array."       -- place an element
arr~put("Last Element in Java array.", 5, 10) -- place another one

do o over arr      -- loop over elements in array (makearray)
   say o
end
say

do with index i item o over arr  -- loop over elements in array (supplier)
   say i":" o
end

::requires BSF.CLS -- loads Java support
```

Output:

```
First Element in Java array.
Last Element in Java array.

1,1: First Element in Java array.
5,10: Last Element in Java array.
```

## BSF4ooRexx: BSFCreateRexxProxy, 1

- RexxProxy
  - A *Java object* that proxies an ooRexx object
  - Allows Java to send messages to ooRexx objects
  - Any method invocations on the Java object will be forwarded as an ooRexx message to the proxied ooRexx object
    - All arguments supplied to the Java method are forwarded in the same sequence with the ooRexx message
    - BSF4ooRexx always appends an additional argument, "**slotDir**" (an ooRexx directory object) to the ooRexx message, which will contain information about the Java method invocation