



Modul University Vienna



Meet the Message Paradigm

37th International Rexx Symposium
May 3rd through May 6th 2026, Barcelona

*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)

- 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** ...



- "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)

- 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 a buttal 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!

```



- 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



```
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
```

- 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)

```
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
```

- 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

- 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 `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!



```

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

```

The screenshot shows the Microsoft Excel interface. The 'Home' tab is active, and the 'Clipboard' and 'Font' groups are visible. The spreadsheet contains the following data:

	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



- 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!



```
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:

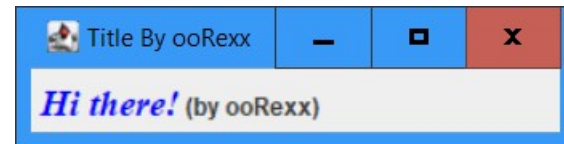
```
dim:          java.awt.Dimension@1c4af82c, dim~class: The BSF class
dim~toString: java.awt.Dimension[width=111,height=222]
dim~width:    111
dim~height:   222
dim~toString: java.awt.Dimension[width=333,height=444]
dim~toString: java.awt.Dimension[width=555,height=666]
```

```
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) ...
```



- Message paradigm
 - **Easy and intuitive for novices**
 - All important object-oriented concepts can be informally (!) explained and understood by novices
- **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](#))
 - **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!

- **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

- Portable zip archives (no installation needed): ooRexx 5.2.0, oorexxshell, dbusoorexx, bsf4oorexx
 - <https://www.ronyrexx.net/xfer/portable>
 - Note: Bsf4oorexx850 (ooRexx-Java bridge) needs Java installed
- Installation packages
 - ooRexx 5.2.0:
 - <https://sourceforge.net/projects/oorexx/files/oorexx/5.2.0>
 - 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>