Open Object Rexx 5.1
Classic Short Reference

Jochem Peelen

36th International Rexx Language Symposium
Vienna, Austria 2025

What it is

- The Short Reference is a 64 page document
- It originally started on a VM mainframe (Document Composition Facilty, IBM 3800 compatible laser printer)
- Two language versions exist:

Open Object Rexx 5.1
Kurzreferenz für Klassiker

Open Object Rexx 5.1
Classic Short Reference

Starting Point

- ooRexx Reference has 795 pages
- IBM Reference Card style is a little too short for my taste

```
►►─DATATYPE(string──┬──────────┬──)──────►◄
                    └─,type────┘
```

returns NUM if *string* is a valid number; otherwise, returns CHAR. To test for specific data types, *type* can be: **A**lphanumeric, **B**inary, **L**owercase, **M**ixed case, **N**umber, **S**ymbol, **U**ppercase, **W**hole number, or **X**(hexadecimal). Returns 1 if string matches the type; otherwise returns 0.

What I did

```
flag = datatype( — string — , ──┬── 'A' ──┬── )   -- Alphanumeric    a-z, A-Z, 0-9
                                ├── 'M' ──┤        -- Mixed case      a-z, A-Z
 0     false                    ├── 'L' ──┤        -- Lowercase       a-z
 1     true                     ├── 'U' ──┤        -- Uppercase       A-Z
                                ├── 'X' ──┤        -- heXadecimal     0-9, a-f, A-F, ''
                                ├── 'B' ──┤        -- Binary          0,   1,   ' ', ''
                                ├── 'O' ──┤        -- lOgical         == 0 | == 1 | .false | .true
                                ├── 'N' ──┤        -- Numeric         any format
                                ├── 'W' ──┤        -- Whole number    e.g. 12, -2.0, 3E4
                                │         │        -- "digits only"   use VERIFY()
                                │         │        --
                                ├── '9' ──┤        -- 9digits         whole number <= 999999999 (9 digits)
                                ├── 'I' ──┤        -- Internal        32bit: <= 9 | 64bit: <= 18 digits
                                ├── 'S' ──┤        -- Symbol          valid as name or constant
                                └── 'V' ──┘        -- Variable        valis as name
Alternative format:

 datatype( ── string ── )    ⇒    NUM if number | CHAR if anything else or null string
```

Returns 1 if *string* is of the data type indicated by the letter, else 0. If the alternative format without type letter is used, NUM will be returned if *string* is numeric, else CHAR.

Limitations

- From the beginning I made no attempt to cover everything
- My goal was limited to those functions (plus methods later) which make up 99 % of my work with REXX (now ooRexx)
- The language itself is so simple that no memory aid is required
- For the remaining 1 %, I refer to the Reference

Railroad Diagrams 1

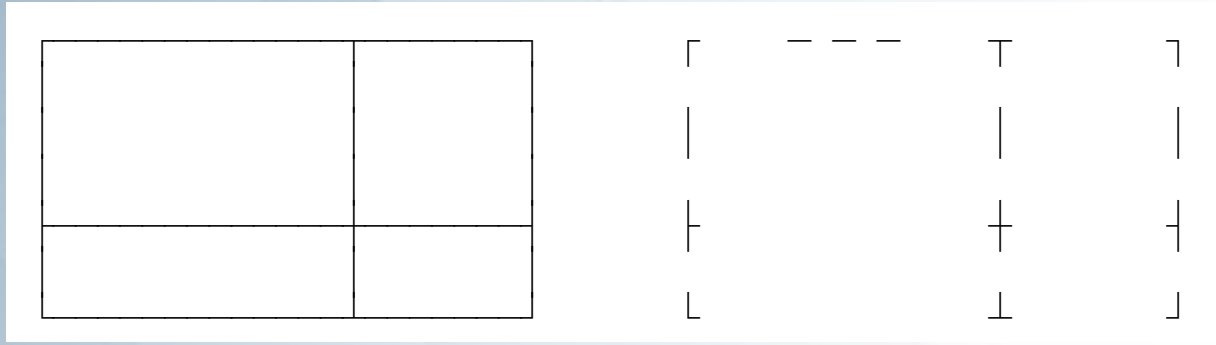
- Source code for the DATATYPE diagram: § starts and ends *italic*; $ starts and ends **bold**; both are replaced by a blank in the output

```
§flag§=$datatype($-§string§- , -+- 'A' -+-$)$  ** Alphanumeric    a-z, A-Z, 0-9
                               +- 'M' -+      ** Mixed case      a-z, A-Z
 §0     false §                +- 'L' -+      ** Lowercase       a-z
 §1     true  §                +- 'U' -+      ** Uppercase       A-Z
                               +- 'X' -+      ** heXadecimal     0-9, a-f, A-F, ''
                               +- 'B' -+      ** Binary          0,   1,   ' ', ''
                               +- 'O' -+      ** lOgical         == 0 | == 1 | .false | .true
                               +- 'N' -+      ** Numeric         any format
                               +- 'W' -+      ** Whole number    e.g. 12, -2.0, 3E4
                               |       |      ** "digits only"   use VERIFY()
                               |       |      **
                               +- '9' -+      ** 9digits         whole number <= 999999999 (9 digits)
                               +- 'I' -+      ** Internal        32bit: <= 9 | 64bit: <= 18 digits
                               +- 'S' -+      ** Symbol          valid as name or constant
                               +- 'V' -+      ** Variable        valis as name
Alternative format:

$datatype($--§string§--$)$   >>   NUM if number | CHAR if anything else or null string
```

Railroad Diagrams 2

- An ooRexx program translates the source into an Encapsulated Postscript (EPS) program that uses light, italic, bold and regular variants of Adobe SourceCodePro, which is monospaced – based on „neighbours“ of „+“
- The box characters of the light variant are used for the lines:



Railroad Diagrams 3

- Modification of the code points is at the core:

```
/Boxzeichen {
Encoding 16#FC  /udieresis   put  % ü
Encoding 16#E4  /adieresis   put  % ä
Encoding 16#F6  /odieresis   put  % ö
Encoding 16#DC  /Udieresis   put  % Ü
Encoding 16#C4  /Adieresis   put  % Ä
Encoding 16#D6  /Odieresis   put  % Ö
Encoding 16#DF  /germandbls  put  % ß
Encoding 16#A0  /uni250C     put  % Ecke li ob
Encoding 16#A1  /uni2500     put  % Strich hori
Encoding 16#A2  /uni252C     put  % T oben
Encoding 16#A3  /uni2510     put  % Ecke re ob
Encoding 16#A4  /uni2502     put  % Strich verti
Encoding 16#A5  /uni251C     put  % T links
Encoding 16#A6  /uni253C     put  % Kreuz
Encoding 16#A7  /uni2524     put  % T rechts
Encoding 16#A8  /uni2514     put  % Ecke li un
Encoding 16#A9  /uni2534     put  % T unten
Encoding 16#AA  /uni2518     put  % Ecke re un
Encoding 16#AB  /uni2574     put  % Strichende
Encoding 16#AC  /uni2576     put  % Strichanfang
Encoding 16#AD  /uni21D2     put  % Pfeil vor Rückgabebeispielen
Encoding 16#27  /quotesingle put  % optisch besseres Hochkomma
} def
```

Railroad Diagrams 4

- The Postscript program is then converted into a PDF file
- This is done by a ooRexx program that calls **Ghostscript** with the right arguments (in particular to find the fonts)
- The document text is a **pdfLaTeX** source, which imbeds the PDF files as images.

Content Overview

- Working with **Character Strings** is described on pages 3-9 (DATATYPE example shown)
- Working with **Word Strings** is on p. 10-11

```
                                   ┌─ {all} ─┐
delword( ─ wordstring ─ , ─ n ─ , ─┤         ├─ )
                          > 0      └─ count ─┘
                                     >= 0

-- delword('   abcd efgh    ijkl ',1)      ⇒    '   '
-- delword('   abcd efgh    ijkl ',2,1)    ⇒    '   abcd ijkl '
-- delword('   abcd efgh    ijkl ',4)      ⇒    '   abcd efgh    ijkl '     no 4th word
-- delword('   abcd efgh    ijkl ',1,0)    ⇒    '   abcd efgh    ijkl '     count 0
```

Returns *wordstring* after removal of *count* words, starting with the *n*-th word. With each deleted word, its **trailing blanks** are also deleted.

- Program Loops is on p. 12-15

```
                  ┌──────────────────────────────┐
                  ↓                              │
do ... with ─┬─ item ─ elementvar ─┬─ over ─ myarray ─┬──────────────────┬─...
             └─ index ─ ixvar ─────┘                  └─ for ─ count ────┘

--Neither ixvar nor elementvar may be used as label
```

Arithmetic, including external library **rxm.cls** from *rosettacode.org*, p. 16-21

```
number ──┬─ ~ceiling ─┬─        -- natural number towards +infinity
         └─ ~floor ───┘          -- natural number towards -infinity

  (2.12)~ceiling     ⇒   3
  (3)~ceiling        ⇒   3
  (-2.12)~ceiling    ⇒  -2
  -2.12~ceiling      ⇒  -3    -- wrong because interpreted as -(2.12~ceiling)
  x = -2.12
  x~ceiling          ⇒  -2

  (2.12)~floor       ⇒   2
  (3)~floor          ⇒   3
  (-2.12)~floor      ⇒  -3
```

Trigonometric Functions

```
┬─ .my.rxm~sin( ─┬─ angle ─ , ┌─ 16 ─────┐ , ┌─ D ─┐ )     360 Degrees
├─ .my.rxm~cos( ─┤            └─ precisi ─┘   ├─────┤
├─ .my.rxm~tan( ─┤                            ├─ R ─┤       2 pi Radian
└─ .my.rxm~cotan( ─┘                          └─ G ─┘       400 Gon
```

```
┬─ .my.rxm~arcsin( ─┬─ number ─ , ┌─ 16 ─────┐ , ┌─ D ─┐ )
├─ .my.rxm~arccos( ─┤             └─ precisi ─┘   ├─────┤
└─ .my.rxm~arctan( ─┘                             ├─ R ─┤
                                                  └─ G ─┘
```

[4] **Attention:** Libraries `rxm.cls` downloaded before 2024-12-23, default to 16 digits and the **360 degree (D)** circle. The old defaults are still shown in the RXM method diagrams above.

Time and Date on p. 22-23

Convert Date Formats

```
-- Converting date format for any date from Monday, 0001-01-01 to Friday, 9999-12-31

                 ┌─ 'N' ─┐                          ┌─ 'N' ─┐                          ┌─────────────┐   ┌─ {default} ─┐
date( ───────────┼─ 'I' ─┼── , ── indate ── , ──────┼─ 'I' ─┼── , ──────────────────── , ─└── sepa ──┘─── , ─────────────┼── )
                 ├─ 'S' ─┤                          ├─ 'S' ─┤                          output          └── insepa ───┘
                 ├─ 'E' ─┤                          ├─ 'E' ─┤                                          indate separator
                 ├─ 'O' ─┤                          ├─ 'O' ─┤                                          if \= default
                 ├─ 'U' ─┤                          ├─ 'U' ─┤                                          for indate format
                 ├─ 'M' ─┤                          ├─ 'B' ─┤
                 ├─ 'W' ─┤                          ├─ 'F' ─┤
                 ├─ 'B' ─┤                          └─ 'T' ─┘
                 ├─ 'D' ─┤                        indate format
                 ├─ 'F' ─┤
                 └─ 'T' ─┘
              output format

--  Convertig basedate to calendar date:
--  date('I',737179,'B')            ⇒    2019-05-01
--  date('I',737179,'B','.')        ⇒    2019.05.01    -- changing default separator

--  Obtaining weekday as a digit:  0 = Monday ... 6 = Sunday
--  date('B','2019-05-01','I') // 7  ⇒    2  means Wednesday
```

Managing Files and Directories, p. 24-26

```
SysGetShortPathName( — pathlongname — )

-- sysgetshortpathname('C:\Program Files (x86)')   ⇒   'C:\PROGRA~2'
-- sysgetshortpathname('C:\doesnt exist')          ⇒   ''
-- sysgetshortpathname('C:\Users')                 ⇒   'C:\Users'
-- sysgetshortpathname('C:\Benutzer')              ⇒   ''            Explorer GUI quirk

-- reverse direction:
SysGetLongPathName( — pathshortname — )
```

Note that the Explorer GUI in non-English Windows in part uses „translated“ directory names which are unknown to Windows, like German *Benutzer* for the *Users* directory.

Read and Write Files (New), p. 27-29

```
mystream = .stream~new( — outfileid — )     -- 1. Define stream to acces file

mystream — ~open('write replace')           -- 2. Open file for writing and replacing

mystream — ~arrayout( — datarray — )        -- 3. Copy data to array (creating new or replacing)

mystream — ~close                           -- 4. Cleanup after use


-- Faster alternative to step 3:

mystream~charout(datarray~tostring,1)       -- 3a. Proposed by Jon Wolfers
mystream~charout('0D0A'x)                   -- 3b. add missing CRLF of last line
```

Read and Write Files (Conventional), p. 30-32

Writing File Lines

```
                                          ┌─ {append} ─┐
rc = lineout( ┬─ fileid ─ , ─ string ─ , ─┤            ├─ )
              │                           └─ lineno ───┘
              └─ fileid ──────────────────────────────┘   -- closes file fileid

method for stream:
                                          ┌─ {append} ─┐
flag = mystream ─ ~lineout( ┬─ string ─ , ─┤            ├─ )
                            │               └─ lineno ───┘
                            └──────────────────────────┘   -- closes the stream
```

Bits and Bytes, p. 33-34

```
c2x('Z')          ⇒    5A
x2c('5A')         ⇒    Z
x2b('5A')         ⇒    01011010
b2x('0101 1010')  ⇒    5A
```

This marks the end of the actual „quick reference“ part of the document.

The „Multitools“ 1

- Written mainly as introduction for newbies
- Stem Variable, p. 35-39
- Array [of strings], p. 40-46

```
-- Stem variable:
 mystem.6    = 'Stralsund'          -- usual notation
 mystem.[7] = 'Greifswald'          -- alternative notation
 say mystem.[i+2]                   ⇒  Greifswald   -- assuming: i = 5
 mystem.['HRO'] = 'Rostock'         -- index is not limited to digits
 mystem.4.6.2                       -- using a 3-dimensional stem variable
 parse var mystem.i ...             -- use parse syntax for variables
 mystem~isA(.stem)                  ⇒  1 (else 0)   -- name here without period

-- Array:
 hanse = .array~new                 -- array must be defined before use
 hanse[7] = 'Greifswald'            -- notation is without period
 say hanse[i+2]                     ⇒  Greifswald   -- assuming: i = 5
 say hanse[0007]                    ⇒  Greifswald   -- 0007 = 7 numerically
 hanse[4,6,2]                       -- using a 3-dimensional array
 parse value hanse[i] with  ...     -- different parse syntax required
 hanse~isA(.array)                  ⇒  1 (else 0)
```

The „Multitools“ 2

- Array description also covers SORT2

SORT2 Syntax

Sort2 is called as a function (not as a method), but expects an array as input. Apart from sorting this array in place, it also returns a new array as result.

```
newarray = sort2( — datarray — , ┌──<──,──┐
                                 └─ sortfield ─┘ )

sortfield:   — start — , ┌─ {all} ─┐ , ┌─ 'A' ─┐ , ┌─ 'I' ─┐
                         └─ length ─┘   └─ 'D' ─┘   ├─ 'C' ─┤
                                                    └─ 'N' ─┘
-- By default, also datarray  is sorted.
-- If datarray is not to be sorted, append method ~copy to its name.
```

- USE ARG capabilities, p. 47-50

```
call myprog                  -- calling MYPROG without argument

-- Different treatment when MYPROG expects an argument:

arg aparm                    -- arg   sees a null string
say aparm                    ⇒  ''

use arg bparm                -- use arg   sees an undefined Variable
say bparm                    ⇒  'BPARM'
```

Classic versus Object Oriented, p. 51-54

13.1 Using SORTWITH in a Classic Program

When starting the program, the desired sort (by area code or town name) is the only argument:

```
arg a1 .                          -- Sort: A by area code or T by town
select case a1
when 'A' then do
     start = 1
     len   = 5
     end
when 'T' then do
     start = 7
     len   = 10
     end
otherwise
     say 'This program expects A or T as argument.'
     exit 24
     end
```

Dependent on argument A[rea code] or T[own name], the sort columns are assigned. Town names shorter that 10 characters pose no problem.

```
infile  = .stream~new('hanse.dat')    -- the file stream to read ...
tabelle = infile~arrayin              -- Array TABELLE receives
```

Method `arrayin` creates array `tabelle` and copies the file lines to it.

```
tabelle~sortwith(.ColumnComparator~new(start,len))
```

This is the sort instruction. First an object of class `ColumnComparator` is created, which will use the columns defined in variables *start* und *len*. This object is then used by method `SortWith` to perform sorting of array `tabelle`.

```
do idx over tabelle
   say idx
   end
```

To write the result to the screen using the `say` keyword, a `do ... over` loop goes through all used items of array `tabelle`. In place of `idx` any convenient variable name could be used.

```
0381  Rostock
03831 Stralsund
03834 Greifswald
03841 Wismar
040   Hamburg
0421  Bremen
0451  Lübeck
```

This is the screen output of the above `do ... over` loop.

```
-- alternative: conventional loop
do i=1 for tabelle~items
   say tabelle[i]
   end
```

Alternatively this conventional loop with an iteration variable could be used. Each item is accessed using the [] syntax (without period, because this is an array, not a stem variable). Method `items` returns the array size. Because we know there are no unused items, this is the number of loop iterations.
This concludes the classic style example.

Other Sort Sequence

```
tabelle~sortwith(.CaselessColumnComparator~new(start,lenght))
```

If upper- and lowercase letters are to be treated as equal, the `Caseless` version of the sorting class is used. As already mentioned, this recognizes only the 26 letters of the English alphabet.

```
-- To invert sorting result use:
--          .InvertingComparator~new( -- myComparatorClass~new(args) -- )

tabelle~sortwith(.InvertingComparator~new(.ColumnComparator~new(start,length)))
```

Class `InvertingComparator` changes the sorting sequence from ascending to descending. It expects the name of the class which will do the actual sort as argument of its `new` method.

13.2 Object Oriented Use of SORTWITH

This section[1] is expressly not intended as a "how to". It is limited by what I learned when trying out object oriented code.
First, a number of object names have to be defined. Each line in `hanse.dat` contains 2 fields (attributes). They will be called `areacod` and `townname` here. We also need a name for the resulting records structure and choose `hansesort` for this.

Setting Up a Class File

To this end, we create a fittingly named file `oohanse.cls` :

```
::class hansesort public inherit comparable
```

This defines class `hansesort` as being useable by any program (`public`). Because its purpose is sorting, it inherits the properties of predefined class `comparable`.

```
::attribute areacod
::attribute townname

::method init
  expose
                    areacod townname
  use strict arg areacod, townname
```

[1] It is derived from ooRexx sample program `sortComposite.rex`.

The ::ATTRIBUTE directives define the data fields in structure `hansesort`. This automatically triggers in the background the creation of a method with the same name as the attribute for read and write access.
When defining a data object, a method with the the prescribed name `init` must be defined. Use of keyword `expose` is required here –and for all other methods– to define, which fields (attributes) this methode may read or change. All other variables inside the method remain invisible to the outside world. To create a record structure as defined in the `hansesort` class, `init` of course needs to access all its fields (attributes). Therefore, keyword `expose` has a complete, *blank separated* list of the fields defined through ::ATTRIBUTE directives.
When called, a method normally receives arguments. Instruction `use strict arg` defines how the arguments are parsed into attributes. Again, for our sort we need a list of all defined attributes. This time a *comma separated* list is required. Option `strict` ensures that the correct number of arguments is present.

```
::method string                    -- method STRING of class HANSESORT
 expose        areacod townname     -- converts objects ...
 return '>'areacod' -- 'townname'<'  -- into a string for SAY
```

The record format (the class) `hansesort` is a structure, not a simple character string. If a structure is encountered by string oriented processes, like keyword **say**, ooRexx uses a default method to convert the data into a chracter string. The alternative is to define a method named `string` for the class. Here a very trivial example is used which inserts some special characters to make obvious that method `string` of class hansesort is active.

Classes for Sorting

For each sorting function (by area code and by town name) a subclass of predefined class `comparator` must be defined. Each class needs a method with the predefined name `compare`which controls the sort.

```
::class AREAsorting public subclass comparator
::method compare
use strict arg lllll, rrrrr
return lllll-areacod~compareto(rrrrr~areacod)
```

Class `AREAsorting` sorts by contents of field (attribute) `areacod`. Method `compare` receives as arguments two consecutive values from field `areacod` übergeben. For this example, the variables `lllll` and `rrrrr` are used. Using the builtin comparison method `compareto` the rather involved expression after `return` returns value 1, 0 or -1. This way method `compare` tells ooRexx whether relation *larger*, *equal* oder *smaller* applies to comparing pair `lllll` and `rrrrr` applies.

```
::class TOWNsorting public subclass comparator
::method compare
use strict arg lllll, rrrrr
return lllll-townname~compareto(rrrrr~townname)
```

Class `TOWNsorting` functions the same way, but uses field (attribute) `townname`. For any additional sorting field, a corresponding `comparator` subclass with its method `compare` is needed.

```
return -lllll-townname~compareto(rrrrr~townname)
```

This line is **not** present in the class file. It shows how to invert the sort order. The minus sign at the beginning of the expression after `return` negates the results (-1, 0, 1) to (1, 0, -1).
This way, two additional classes could be easily coded for *descending* sort by area code or town name.
This completes class file `oohanse.cls` and we can code the program to use it.

ooRexx Code Using the Sorting Classes

The program file is named `oohanse.rex`:

```
myfile  = 'hanse.dat'                -- name of data file
mytable = .array~new                 -- create empty array MYTABLE
```

These two lines are the same as in the classic example of page 51.

```
do i=1 while lines(myfile)
   myline = linein(myfile)                      -- read line and ...
   parse var myline 1 afield 7 tfield           -- parse into 2 fields
   mytable~append(.hansesort~new(afield,tfield)) -- uses class HANSESORT
   end i
```

Writing the data to array `mytable` is done using class `hansesort` to create objects as defined. Because this class expects 2 arguments, each file record has to be parsed into 2 fields (attributes). Number and sequence of attributes in the arguments passed to methode `new` must be identical to the actually used method `init` of class `hansesort`. In place of variables `afield` and `tfield` any names may be used.

```
arg a1 .                    -- A or T for sorting by area code or town
select case a1
when 'A' then mytable~sortwith(.AREAsorting~new)
when 'T' then mytable~sortwith(.TOWNsorting~new)
otherwise
     say 'This program expects A or T as argument.'
     exit 24
     end
```

This is the sorting step. Depending on using argument A or T when starting the program, the appropriate subclass is used by method `sortWith`.

```
do idx over mytable
   say idx          -- SAY implicitly uses the STRING method of HANSESORT
   end
```

The loop to write the result to the screen is the same as in the classic example.

```
::REQUIRES oohanse.cls
```

The ::REQUIRES directive at the end of the program tells ooRexx where to find the class definitions. Since February 2020 ooRexx automatically searches for files with extension `.cls` if none is given.
An alternative would be to copy the complete file `oohanse.cls` in place of the ::REQUIRES directive into the program file. In this case the the `public` options in the class definitions were not required. And it would make the class definitions inaccessible to other programs.

```
>0381  -- Rostock<
>03831 -- Stralsund<
>03834 -- Greifswald<
>03841 -- Wismar<
>040   -- Hamburg<
>0421  -- Bremen<
>0451  -- Lübeck<
```

Independent of placing the class definitions, the screen output –sorted by area code– will be as shown above.
The items in array `tabelle` are now objects as defined in class `hansesort`. Our method `string` has done the preprocessing for keyword SAY as the additional special characters show.

Comparing text sizes of describing a simple sort using method SORTWITH.

For the relatively very simple problems of my everyday work, I see no advantage in going obect oriented.

Back Matter

- ooRexx fundamentals, again for newbies
- Diagram syntax, index, table of contents

- Does a label **uprog:** exist in the current program file? This step is skipped, if the name is included in single or double quotes.
- Is **uprog** a builtin part of ooRexx? Which are:
  - functions implemented in the interpreter[5] and
  - the **Rx...** and **Sys...** functions from library **rexxutil.dll** that comes with ooRexx.[6]
- Does a directive **::ROUTINE uprog** exist in the current program file?
- Exists, in one of the files listed in ::REQUIRES directives, a directive **::ROUTINE uprog PUBLIC**?
- Does in *Rexx Macrospace*, among the files loaded with **Before** option, a program named **uprog** exist?
- Does name **uprog** exist in an *external function library* (DLL file) that was loaded via directive **::REQUIRES *name* LIBRARY**?
- Does a file named **uprog.rex** exist ...
  - in the current directory, or
  - in a directory of the PATH environment variable – which by default includes the ooRexx installation directory?
- Does in *Rexx Macrospace*, among the files loaded with **After** option, a program named **uprog** exist?
- ooRexx stops and reports: *Error 43: Could not find Routine UPROG*

R
RC – 57
~remove – 43
~removeitem – 43
~replaceAT – 8
RESULT – 57, 58
return – 48, 49, 57
reverse() – 8
rexxutil.dll – 58
rgf_util2.rex – 43, 46
right() – 6
Rosettacode – 21
RxCalc...() – 18, 19
rxm...() – 21
rxm.cls – 19, 21
rxmath.dll – 18

Are there any questions?