Department of Information Systems and Operations Management

Debugging Multithreaded ooRexx Programs

Making TRACE Even More Powerful

The 2024 International Rexx Symposium
Brisbane, Queensland, Australia
March 3rd – March 6th 2024

Overview

- Bird eye's view of multithreading (MT) concepts in ooRexx
- ooRexx and TRACE
- The new TraceObject class in ooRexx 5.1.0beta
- Some examples
- Roundup

Bird Eye's View of MT in ooRexx

- ooRexx is a multithreaded programming language
  - Keyword statement REPLY returns from a method, but has the remainder of that method execute in parallel as a new activity on a new thread
  - Keyword statement GUARD
    - Controls whether guarded method routines of the same class (“scope”) get serialized using the object's scope lock as a semaphore
    - By default method routines are guarded but the programmer can override this default
    - The keyword GUARD allows to change the state of a method from guarded to unguarded and vice versa
    - Unguarded method routines can always run in parallel to any other method defined in the same class (“scope”)
  - Using the start method of the Message or Object class allows to dispatch messages on a new thread to carry out the desired activity

TRACE, 1

- Tracing in Rexx – and therefore in its successor ooRexx – is a very powerful means to analyze and to understand what the ooRexx code does at runtime
- There is a TRACE keyword statement and a TRACE() built-in-function (BIF) to control tracing of ooRexx programs, both offering the options:
  - **A**ll: the statement will be traced (shown) *before* it gets executed
  - **C**ommands: the command will be traced (shown) *before* it gets executed, in case of an error *or* failure condition the command's return code will be displayed
  - **E**rror: traces a command with an error *or* failure condition together with the return code *after* it got executed
  - **F**ailure: traces a command with a failure condition together with the return code *after* it got executed; this option is a synonym for option **N**ormal which is in effect by default

TRACE, 2

- Options (continued)
  - Intermediates: traces (shows) all clauses *before* they get executed, traces the results of expressions and of name substitutions
  - Labels: traces method and routine invocations, internal subroutine calls, transfer of control using the SIGNAL keyword instruction and labels passed during program execution
  - Normal: sets tracing to trace failures in commands, unless the ooRexx ::OPTIONS TRACE directive sets a different program wide default option
  - Off: traces nothing and sets the trace prefix option to off
  - Results: traces all statements *before* execution, displays values assigned during ARG, PARSE, PULL and USE and the final result

::OPTIONS TRACE, 2

- The ::OPTIONS directive statement of ooRexx allows to define the *default trace option* for the *entire* program
  - Its TRACE subkeyword is followed by one of the aforementioned trace options

Sample doc_event.rex, 1

- The ooRexx reference book (rexxref.pdf) includes a multithreaded sample in section “5.4.7. EventSemaphore Class” to demonstrate how one can use an event semaphore to synchronize the threads (activities)
  - The main program creates an event semaphore
  - It then creates a few instances of a class named Task and sends each a waitFor message which will cause the receiving objects to invoke the method waitFor defined in the class Task
  - The method waitFor will
    - Return immediately control to the main program using the REPLY keyword statement
    - On a new thread it will fetch the supplied arguments, output its supplied name and then *wait*s for the event semaphore to be posted by the main program
  - After the loop and a short sleep the main program will *post* the event semaphore releasing all the threads that have been *wait*ing for this event to happen

Sample doc_event.rex, No Trace 2

```
say "main starts tasks"
do nr = 1 to 3          -- create tasks that wait on semaphore
    .task~new~waitFor(event, "task" nr) -- create object, send
message
end
call SysSleep 0.1       -- sleep a bit
say "main posts"
event~post              -- now post the event semaphore
say "main ends"

::class Task
::method waitFor
    reply   -- returns to caller, remaining code runs on new thread
    use strict arg event, name  -- fetch event semaphore and name
    say name "waits"
    event~wait          -- wait until semaphore gets posted
    say name "runs"
```

Output (last three lines may be shown in a different sequence):

```
main starts tasks
task 1 waits
task 2 waits
task 3 waits
main posts
main ends
task 1 runs
task 3 runs
task 2 runs
```

Sample doc_event.rex Trace All, 3

```
-- doc_event.rex
event = .EventSemaphore~new
say "main starts tasks"
do nr = 1 to 3
    .task~new~waitFor(event, "task" nr)
end
call SysSleep 0.1
say "main posts"
event~post
say "main ends"

::class Task
::method waitFor
    reply
    use strict arg event, name
    say name "waits"
    event~wait
    say name "runs"

::options trace all
```

Output (maybe):

```
     2 *-* event = .EventSemaphore~new
     3 *-* say "main starts tasks"
main starts tasks
     4 *-* do nr = 1 to 3
     5 *-*   .task~new~waitFor(event, "task" nr)
       >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
    14 *-* reply
     6 *-* end
       >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
     4 *-* do nr = 1 to 3
    15 *-* use strict arg event, name
     5 *-*   .task~new~waitFor(event, "task" nr)
    16 *-* say name "waits"
task 1 waits
       >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
    17 *-* event~wait
    14 *-* reply
     6 *-* end
     4 *-* do nr = 1 to 3
     5 *-*   .task~new~waitFor(event, "task" nr)
       >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
    14 *-* reply
       >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
     6 *-* end
       >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
    15 *-* use strict arg event, name
     4 *-* do nr = 1 to 3
    15 *-* use strict arg event, name
    16 *-* say name "waits"
task 2 waits
     7 *-* call SysSleep 0.1
    16 *-* say name "waits"
task 3 waits
    17 *-* event~wait
    17 *-* event~wait
     8 *-* say "main posts"
main posts
     9 *-* event~post
    10 *-* say "main ends"
main ends
    18 *-* say name "runs"
task 2 runs
    18 *-* say name "runs"
task 3 runs
    18 *-* say name "runs"
task 1 runs
```

Some Remarks

- TRACE works in multithreaded programs as well!
- However the trace prefix does not include any thread related information like
  - Thread number
  - Which of the guarded methods owns the object's scope lock, which one must wait for it (or with other words which guarded method is currently blocked)
- In complex ooRexx deployments the following information in the trace prefix may be helpful for debugging MT programs additionally
  - Which Rexx interpreter instance executes the statement, which invocation identifier is the current statement located at, which method runs against which attribute pool (i.e. for which object, instance)

New TraceObject Class, 1

- ooRexx 5.1.0beta introduces a new class: TraceObject (a subclass of StringTable)
- TRACE will create an instance of this class and fills in all trace related information, including multithreaded related ones (see next slide)
- *TraceObject* defines the following class attributes
  - *collector* – by default .nil, if set to an object that understands the *append* message each created *TraceObject* will be appended to it
  - *counter* – keeps a count of created *TraceObjects*
  - *option* - allows to set an option (only the first character gets used): **N**ormal (default), **P**rofiling, **T**hread, **S**tandard, **F**ull

New TraceObject Class, 2

- TraceObject has a makeString method that returns by default a string formatted in the classic trace layout using the contained information
  - One can use the *TraceObject* class method *setMakeString(myMakeString)* to change the method and *unsetMakeString()* to use the default implementation
  - The default *makeString* implementation of *TraceObject,* if its class attribute *option* is currently set to
    - 'N' (normal) or 'P' (profiling/probing) then the normal trace string (trace prefix plus the traced line) gets returned
    - 'T' (thread) then the return string consists of the trace prefix with the thread number inserted after its second character and then concatenated with the trace line
    - 'S' (standard) or 'F' (full): the normal trace string gets prepended with additional square bracketed information

New TraceObject Class, 3

- A TraceObject instance will have entries with the following indexes
  - ATTRIBUTEPOOL
    - a number, *makeString* prepends it with the letter A if option is set to F
  - HASOBJECTLOCK (may be subject to be renamed to HASSCOPELOCK)
    - *.true*/*.false*, *makeString* uses an asterisk, if *.true*, a blank character else if option is set to F
  - INTERPRETER
    - a number, *makeString* prepends it with the letter R if option is set to F
  - INVOCATION
    - a number, *makeString* prepends it with the letter I if option is set to F or S
  - ISGUARDED
    - *.true*/*.false*, *makeString* uses the letter G, if *.true*, the letter U else if option is set to F or S
  - NR
    - a sequential whole number, the default *makeString* implementation does not use it

New TraceObject Class, 4

- A TraceObject instance will have entries with the following indexes (continued)
  - OBJECTLOCKCOUNT (may be subject to be renamed to SCOPELOCKCOUNT)
    - a number, *makeString* prepends it with the letter L if option is set to F or S
  - OPTION
    - The value of the class attribute *option* that was in effect when this instance got created, the default *makeString* implementation does not use it
  - THREAD
    - a number, *makeString* prepends it with the letter T if option is set to F or S, or the number gets inserted in the trace prefix if option is set to T

New TraceObject Class, 5

- A TraceObject instance will have entries with the following indexes (continued)
  - TIMESTAMP
    - A DateTime instance representing the creation date and time of this TraceObject instance, the default *makeString* implementation does not use it
  - TRACELINE
    - The trace line string

Changing Sample doc_event.rex Option T (Thread)

- To get to see the thread number one simply changes *TraceObject*'s class attribute *option* to ***T**hread* (only the first letter is needed)
  - Any trace output thereafter will be formatted accordingly
  - One can now study which statement gets executed on which thread

Sample doc_event.rex Option T (Thread)

```
.traceObject~option="T"
event = .EventSemaphore~new
say "main starts tasks"
do nr = 1 to 3
    .task~new~waitFor(event, "task" nr)
end
call SysSleep 0.1
say "main posts"
event~post
say "main ends"

::class Task
::method waitFor
    reply
    use strict arg event, name
    say name "waits"
    event~wait
    say name "runs"

::options trace all
```

Output (maybe):

```
     1 *-* .traceObject~option="T" -- show thread number in trace prefix
     2 *-1* event = .EventSemaphore~new
     3 *-1* say "main starts tasks"
main starts tasks
     4 *-1* do nr = 1 to 3
     5 *-1*   .task~new~waitFor(event, "task" nr)
       >I1> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
    14 *-1* reply
     6 *-1* end
       >I2> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
     4 *-1* do nr = 1 to 3
    15 *-2* use strict arg event, name
     5 *-1*   .task~new~waitFor(event, "task" nr)
    16 *-2* say name "waits"
task 1 waits
       >I1> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
    17 *-2* event~wait
    14 *-1* reply
     6 *-1* end
     4 *-1* do nr = 1 to 3
     5 *-1*   .task~new~waitFor(event, "task" nr)
       >I1> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
       >I3> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
    14 *-1* reply
    15 *-3* use strict arg event, name
     6 *-1* end
       >I4> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
    16 *-3* say name "waits"
task 2 waits
     4 *-1* do nr = 1 to 3
    15 *-4* use strict arg event, name
    17 *-3* event~wait
     7 *-1* call SysSleep 0.1
    16 *-4* say name "waits"
task 3 waits
    17 *-4* event~wait
     8 *-1* say "main posts"
main posts
     9 *-1* event~post
    10 *-1* say "main ends"
main ends
    18 *-4* say name "runs"
task 3 runs
    18 *-3* say name "runs"
task 2 runs
    18 *-2* say name "runs"
task 1 runs
```

Changing Sample doc_event.rex Option S (Standard)

- To get to see the standard additional bracketed trace information one simply changes *TraceObject*'s class attribute *option* to ***S**tandard* (only the first letter is needed)
  - Any trace output thereafter will be formatted accordingly
  - The bracketed additional trace information letters indicate
    - T: thread on which activity runs
    - I: invocation identifier
    - For method routines in addition
      - G or U to indicate a guarded or an unguarded method
      - L the number of object locks
      - * the method owns the object's scope lock, else blank

Sample doc_event.rex Option S (Standard)

```
.traceObject~option="S"
event = .EventSemaphore~new
say "main starts tasks"
do nr = 1 to 3
    .task~new~waitFor(event, "task" nr)
end
call SysSleep 0.1
say "main posts"
event~post
say "main ends"

::class Task
::method waitFor
    reply
    use strict arg event, name
    say name "waits"
    event~wait
    say name "runs"

::options trace all
```

Output (maybe):

```
     1 *-* .traceObject~option="S" -- show thread number in trace prefix
[T1   I1   ]               2 *-* event = .EventSemaphore~new
[T1   I1   ]               3 *-* say "main starts tasks"
main starts tasks
[T1   I1   ]               4 *-* do nr = 1 to 3
[T1   I1   ]               5 *-*   .task~new~waitFor(event, "task" nr)
[T1   I2    G L0    ]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[T1   I2    G L1   *]     14 *-* reply
[T1   I1   ]               6 *-* end
[T2   I2    G L1   *]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[T1   I1   ]               4 *-* do nr = 1 to 3
[T2   I2    G L1   *]     15 *-* use strict arg event, name
[T1   I1   ]               5 *-*   .task~new~waitFor(event, "task" nr)
[T2   I2    G L1   *]     16 *-* say name "waits"
task 1 waits
[T1   I3    G L0    ]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[T2   I2    G L1   *]     17 *-* event~wait
[T1   I3    G L1   *]     14 *-* reply
[T1   I1   ]               6 *-* end
[T1   I1   ]               4 *-* do nr = 1 to 3
[T3   I3    G L1   *]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[T1   I1   ]               5 *-*   .task~new~waitFor(event, "task" nr)
[T3   I3    G L1   *]     15 *-* use strict arg event, name
[T1   I4    G L0    ]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[T3   I3    G L1   *]     16 *-* say name "waits"
task 2 waits
[T1   I4    G L1   *]     14 *-* reply
[T3   I3    G L1   *]     17 *-* event~wait
[T4   I4    G L1   *]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[T1   I1   ]               6 *-* end
[T4   I4    G L1   *]     15 *-* use strict arg event, name
[T1   I1   ]               4 *-* do nr = 1 to 3
[T4   I4    G L1   *]     16 *-* say name "waits"
task 3 waits
[T1   I1   ]               7 *-* call SysSleep 0.1
[T4   I4    G L1   *]     17 *-* event~wait
[T1   I1   ]               8 *-* say "main posts"
main posts
[T1   I1   ]               9 *-* event~post
[T1   I1   ]              10 *-* say "main ends"
main ends
[T3   I3    G L1   *]     18 *-* say name "runs"
[T2   I2    G L1   *]     18 *-* say name "runs"
task 2 runs
[T4   I4    G L1   *]     18 *-* say name "runs"
task 1 runs
task 3 runs
```

Changing Sample doc_event.rex Option F (Full)

- To get to see the standard additional bracketed trace information one simply changes *TraceObject*'s class attribute *option* to ***F****ull* (only the first letter is needed)
  - Any trace output thereafter will be formatted accordingly
  - The bracketed additional trace information letters indicate
    - *R: Rexx interpreter instance that runs the activity*
    - T: thread on which activity runs
    - I: invocation identifier
    - For method routines in addition
      - *A the attribute (object variable) pool number*
      - G or U to indicate a guarded or an unguarded method
      - L the number of object locks
      - * the guarded method owns the object's scope lock, else blank

Sample doc_event.rex Option F (Full)

```
.traceObject~option="F"
event = .EventSemaphore~new
say "main starts tasks"
do nr = 1 to 3
    .task~new~waitFor(event, "task" nr)
end
call SysSleep 0.1
say "main posts"
event~post
say "main ends"

::class Task
::method waitFor
    reply
    use strict arg event, name
    say name "waits"
    event~wait
    say name "runs"

::options trace all
```

Output (maybe):

```
     1 *-* .traceObject~option="F" -- show thread number in trace prefix
[R1   T1   I1   ]                     2 *-* event = .EventSemaphore~new
[R1   T1   I1   ]                     3 *-* say "main starts tasks"
main starts tasks
[R1   T1   I1   ]                     4 *-* do nr = 1 to 3
[R1   T1   I1   ]                     5 *-*   .task~new~waitFor(event, "task" nr)
[R1   T1   I2    G A1    L0    ]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[R1   T1   I2    G A1    L1   *]     14 *-* reply
[R1   T1   I1   ]                     6 *-* end
[R1   T1   I1   ]                     4 *-* do nr = 1 to 3
[R1   T1   I1   ]                     5 *-*   .task~new~waitFor(event, "task" nr)
[R1   T1   I3    G A2    L0    ]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[R1   T1   I3    G A2    L1   *]     14 *-* reply
[R1   T2   I3    G A2    L1   *]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[R1   T2   I3    G A2    L1   *]     15 *-* use strict arg event, name
[R1   T2   I3    G A2    L1   *]     16 *-* say name "waits"
task 2 waits
[R1   T2   I3    G A2    L1   *]     17 *-* event~wait
[R1   T3   I2    G A1    L1   *]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[R1   T3   I2    G A1    L1   *]     15 *-* use strict arg event, name
[R1   T3   I2    G A1    L1   *]     16 *-* say name "waits"
task 1 waits
[R1   T3   I2    G A1    L1   *]     17 *-* event~wait
[R1   T1   I1   ]                     6 *-* end
[R1   T1   I1   ]                     4 *-* do nr = 1 to 3
[R1   T1   I1   ]                     5 *-*   .task~new~waitFor(event, "task" nr)
[R1   T1   I4    G A3    L0    ]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[R1   T1   I4    G A3    L1   *]     14 *-* reply
[R1   T1   I1   ]                     6 *-* end
[R1   T1   I1   ]                     4 *-* do nr = 1 to 3
[R1   T1   I1   ]                     7 *-* call SysSleep 0.1
[R1   T4   I4    G A3    L1   *]        >I> Method "WAITFOR" with scope "TASK" in package "doc_event.rex".
[R1   T4   I4    G A3    L1   *]     15 *-* use strict arg event, name
[R1   T4   I4    G A3    L1   *]     16 *-* say name "waits"
task 3 waits
[R1   T4   I4    G A3    L1   *]     17 *-* event~wait
[R1   T1   I1   ]                     8 *-* say "main posts"
main posts
[R1   T1   I1   ]                     9 *-* event~post
[R1   T1   I1   ]                    10 *-* say "main ends"
main ends
[R1   T4   I4    G A3    L1   *]     18 *-* say name "runs"
task 3 runs
[R1   T2   I3    G A2    L1   *]     18 *-* say name "runs"
task 2 runs
[R1   T3   I2    G A1    L1   *]     18 *-* say name "runs"
task 1 runs
```

Customize Trace Output
doc_event_mkStr.rex

- It is possible to tailor the trace output
  - Create a routine that fetches the *traceObject* as its single argument
  - Use the information in the *traceObject* to your likings
  - Create a string that meets your debugging needs and return it

Customize Trace Output

doc_event_mkStr.rex

```
-- set to the code of the  myMTprefix resource
.traceObject~setMakeString(.resources~myMTprefix)
event = .EventSemaphore~new
say "main starts tasks"
do nr = 1 to 3
    .task~new~waitFor(event, "task" nr)
end
call SysSleep 0.1
say "main posts"
event~post
say "main ends"

::class Task
::method waitFor
    reply
    use strict arg event, name
    say name "waits"
    event~wait
    say name "runs"

::options trace all

::RESOURCE myMTprefix   -- define own trace format
  use arg traceObj      -- fetch traceObject
  tod=traceObj["TIMESTAMP"]~timeOfDay
  return '#' adjRight(traceObj["NR"])"," tod"," -
         "T"traceObj["THREAD"]":" traceObj["TRACELINE"]
  adjRight: procedure   -- adjust right
    use strict arg value, width=3
    if value~length>=width then return value
    return value~right(width)
::END
```

Output (maybe):

```
     2 *-* .traceObject~setMakeString(.resources~myMTprefix)
#   2, 22:27:11.562507, T1:      3 *-* event = .EventSemaphore~new
#   3, 22:27:11.562645, T1:      4 *-* say "main starts tasks"
main starts tasks
#   4, 22:27:11.562749, T1:      5 *-* do nr = 1 to 3
#   5, 22:27:11.562822, T1:      6 *-*   .task~new~waitFor(event, "task" nr)
#   6, 22:27:11.562909, T1:        >I> Method "WAITFOR" with scope "TASK" in package "doc_event_mkStr.rex".
#   7, 22:27:11.562968, T1:     15 *-* reply
#   8, 22:27:11.563064, T1:      7 *-* end
#   9, 22:27:11.563128, T1:      5 *-* do nr = 1 to 3
#  10, 22:27:11.563200, T1:      6 *-*   .task~new~waitFor(event, "task" nr)
#  11, 22:27:11.563259, T1:        >I> Method "WAITFOR" with scope "TASK" n package "doc_event_mkStr.rex".
#  12, 22:27:11.563321, T1:     15 *-* reply
#  13, 22:27:11.563425, T2:        >I> Method "WAITFOR" with scope "TASK" n package "doc_event_mkStr.rex".
#  14, 22:27:11.563505, T1:      7 *-* end
#  15, 22:27:11.563548, T2:     16 *-* use strict arg event, name
#  16, 22:27:11.563583, T3:        >I> Method "WAITFOR" with scope "TASK" n package "doc_event_mkStr.rex".
#  17, 22:27:11.563688, T1:      5 *-* do nr = 1 to 3
#  18, 22:27:11.563788, T2:     17 *-* say name "waits"
task 2 waits
#  19, 22:27:11.563908, T3:     16 *-* use strict arg event, name
#  20, 22:27:11.564005, T1:      6 *-*   .task~new~waitFor(event, "task" nr)
#  21, 22:27:11.564084, T2:     18 *-* event~wait
#  22, 22:27:11.564179, T3:     17 *-* say name "waits"
task 1 waits
#  23, 22:27:11.564262, T1:        >I> Method "WAITFOR" with scope "TASK" in package "doc_event_mkStr.rex".
#  24, 22:27:11.564412, T3:     18 *-* event~wait
#  25, 22:27:11.564510, T1:     15 *-* reply
#  26, 22:27:11.564704, T1:      7 *-* end
#  27, 22:27:11.564841, T4:        >I> Method "WAITFOR" with scope "TASK" in package "doc_event_mkStr.rex".
#  28, 22:27:11.564882, T1:      5 *-* do nr = 1 to 3
#  29, 22:27:11.564983, T4:     16 *-* use strict arg event, name
#  30, 22:27:11.565091, T1:      8 *-* call SysSleep 0.1
#  31, 22:27:11.565159, T4:     17 *-* say name "waits"
task 3 waits
#  32, 22:27:11.565619, T4:     18 *-* event~wait
#  33, 22:27:11.670017, T1:      9 *-* say "main posts"
main posts
#  34, 22:27:11.670196, T1:     10 *-* event~post
#  35, 22:27:11.670331, T1:     11 *-* say "main ends"
main ends
#  36, 22:27:11.670484, T3:     19 *-* say name "runs"
task 1 runs
#  37, 22:27:11.670638, T2:     19 *-* say name "runs"
task 2 runs
#  38, 22:27:11.670836, T4:     19 *-* say name "runs"
task 3 runs
```

Changing Sample doc_event.rex
Option P (Profiling/Probing)

- Complex MT programs may need to be analyzed programmatically
- To do so
  - Use the ::OPTIONS TRACE directive to activate tracing
  - Set the *TraceObject* class attribute *collector*
    - The collector object needs to understand the message *append*
      - E.g. all *OrderedCollection* classes of ooRexx can be used
  - Set the *TraceObject* class attribute *option to* ***P***
- Note: the following sample then uses traceutil.cls (WIP: work in progress) to create a CSV file from the collected *traceObjects* for documentation or for further analysis e.g. with a spreadsheet

Changing Sample doc_event.rex
Option P (Profiling/Probing)

```
.traceObject~collector=.array~new   -- from now on collecting
.traceObject~option="P" -- do not display trace
event = .EventSemaphore~new
say "main starts tasks"
do nr = 1 to 3
    .task~new~waitFor(event, "task" nr)
end
call SysSleep 0.1
say "main posts"
event~post
say "main ends"
trace n                 -- no tracing from here on
call SysSleep 0.1       -- let threads end
say "--- now creating a CSV file (tmp.csv) ..."
call toCsvFile  "tmp.csv",  .traceObject~collector

::class Task
::method waitFor
    reply    -- returns to caller
    use strict arg event, name
    say name "waits"
    event~wait
    say name "runs"

::requires "traceutil.cls"  --  toCsvFile(), WIP
::options trace all
```

Output (maybe):

```
     1 *-* .traceObject~collector=.array~new   -- from now on collecting
     2 *-* .traceObject~option="P" -- do not display trace
main starts tasks
task 2 waits
task 1 waits
task 3 waits
main posts
main ends
task 1 runs
task 3 runs
task 2 runs
```

tmp.csv (maybe):

```
option,nr,timestamp,interpreter,thread,invocation,isGuarded,attributePool,objectLockCount,hasObjectLock,traceline
"N","2","2024-02-28T18:00:24.248185","1","1","1",,,,,"     2 *-* .traceObject~option=""P"" -- do not display trace"
"P","3","2024-02-28T18:00:24.248218","1","1","1",,,,,"     3 *-* event = .EventSemaphore~new"
"P","4","2024-02-28T18:00:24.248235","1","1","1",,,,,"     4 *-* say ""main starts tasks"""
"P","5","2024-02-28T18:00:24.248255","1","1","1",,,,,"     5 *-* do nr = 1 to 3"
"P","6","2024-02-28T18:00:24.248269","1","1","1",,,,,"     6 *-*   .task~new~waitFor(event, ""task"" nr)"
… cut …
```

Roundup

- New TraceObject class (subclass of StringTable) in ooRexx 5.1.0beta
  - For each trace a *TraceObject* gets created and filled in with the trace information
  - The class attribute *option* allows for changing the output to include MT related information to help debug MT programs
  - The class attribute *collector* allows for collecting all created *TraceObjects* for documenttion or later analysis
- traceutils.cls defines utility routines, e.g. storing (and reading) collected traceObjects in (from) CSV and JSON text files
  - WIP: work in progress
  - Planned to come up with a routine that possilby flags deadlocks
- Can be used for analyzing (profiling) classic Rexx programs!