Department of Information Systems
and Operations Management

The Rexx TraceTool

(Taking advantage of ooRexx 5.1.0's new TraceObject Class)

Creating and Analyzing Tracelogs

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

Prof. Rony G. Flatscher

Brief Overview of TRACE

- REXX (1979, IBM)
  - TRACE keyword instruction and built-in function (BIF)
  - Trace "normal", "all", "results", "intermediates"
- Object Rexx (1996, IBM)
  - Introduces message paradigm
  - Adds ability to define classes, methods, attributes
- ooRexx (open object Rexx 2004, Rexx Language Association)
  - Introducing and demonstrating new features

TRACE Keyword Instruction and Built-in Function (BIF)

- REXX (1979, IBM)
  - A dynamic and dynamically typed language
  - Three instruction types
    - Assignment (second token an equal sign =)
    - Keyword instruction (first word is a defined REXX keyword)
    - Command instruction (any other string)
  - Allows for tracing all instruction types
    - TRACE keyword instruction and built-in function (BIF) with possible options
      - Normal: traces failures in command instructions (on by default)
      - All: displays instructions before executing them
      - Results: displays the instruction's result, if any
      - Intermediates: displays all intermediate evaluations in an expression

REXX Example, 1

REXX Instructions and TRACE Normal (Default)

```
a="hello, world"        -- an assignement instruction
say "a="a               -- a SAY keyword instruction
say "TRACE option in effect:" trace() -- 'N'='Normal'
say
'echo' a                -- a known system command
say "return code:" rc -- return code
say
address nope 'nixi' a -- a non-existing command environment
say "return code:" rc -- return code
```

Output:

```
a="hello, world"
TRACE option in effect: N

hello, world
return code: 0

     8 *-* address nope 'nixi' -- a non-existing command environment
       >>> "nixi hello, world"
       +++ "RC (30)"
return code: 30
```

"trace prefix"

REXX Example, 2

TRACE Options All, Results, Intermediates

```
do opt over 'All', 'Results', 'Intermediates'
   say "setting TRACE to '"opt"':"
   currOpt=trace(opt) -- TRACE (function)
   a=100+random() -- line # 4
   say "a:" a      -- line # 5
   TRACE N         -- line # 6 (keyword)
   say "---"
end
```

Output:

```
setting TRACE to 'All':
     4 *-*   a=100+random() -- line # 4
     5 *-*   say "a:" a     -- line # 5
a: 1088
     6 *-*   TRACE N        -- line # 6 (keyword)
---
setting TRACE to 'Results':
     4 *-*   a=100+random() -- line # 4
       >>>     "436"
     5 *-*   say "a:" a     -- line # 5
       >>>     "a: 436"
a: 436
     6 *-*   TRACE N        -- line # 6 (keyword)
---
setting TRACE to 'Intermediates':
       >F>     TRACE => "N"
       >=>     CURROPT <= "N"
     4 *-*   a=100+random() -- line # 4
       >L>     "100"
       >F>     RANDOM => "514"
       >O>     "+" => "614"
       >>>     "614"
       >=>     A <= "614"
     5 *-*   say "a:" a     -- line # 5
       >L>     "a:"
       >V>     A => "614"
       >O>     " " => "a: 614"
       >>>     "a: 614"
a: 614
     6 *-*   TRACE N        -- line # 6 (keyword)
---
```

ooRexx Example, 3

ooRexx Program With Multithreading (TRACE All)

```
t1=.test~new       -- create an instance
t2=.test~new       -- create another instance
t1~hey
t2~ho
say "-"~copies(50)
t2~start('hey')
t1~start('ho')
say "-"~copies(50)

::class test       -- some Rexx class
::method hey       -- by default: guarded
  say 'hey (guarded)'

::method ho unguarded   -- unguarded
  say 'ho (unguarded)'

::options trace all
```

Output:

```
     1 *-* t1=.test~new      -- create an instance
     2 *-* t2=.test~new      -- create another instance
     3 *-* t1~hey
       >I> Method "HEY" with scope "TEST"
    12 *-* say 'hey (guarded)'
hey (guarded)
       <I< Method "HEY" with scope "TEST"
     4 *-* t2~ho
       >I> Method "HO" with scope "TEST"
    15 *-* say 'ho (unguarded)'
ho (unguarded)
       <I< Method "HO" with scope "TEST"
     5 *-* say "-"~copies(50)
--------------------------------------------------
     6 *-* t2~start('hey')
     7 *-* t1~start('ho')
     8 *-* say "-"~copies(50)
--------------------------------------------------
       >I> Method "HEY" with scope "TEST"
       >I> Method "HO" with scope "TEST"
    12 *-* say 'hey (guarded)'
hey (guarded)
    15 *-* say 'ho (unguarded)'
ho (unguarded)
       <I< Method "HEY" with scope "TEST"
       <I< Method "HO" with scope "TEST"
```

ooRexx and TRACE

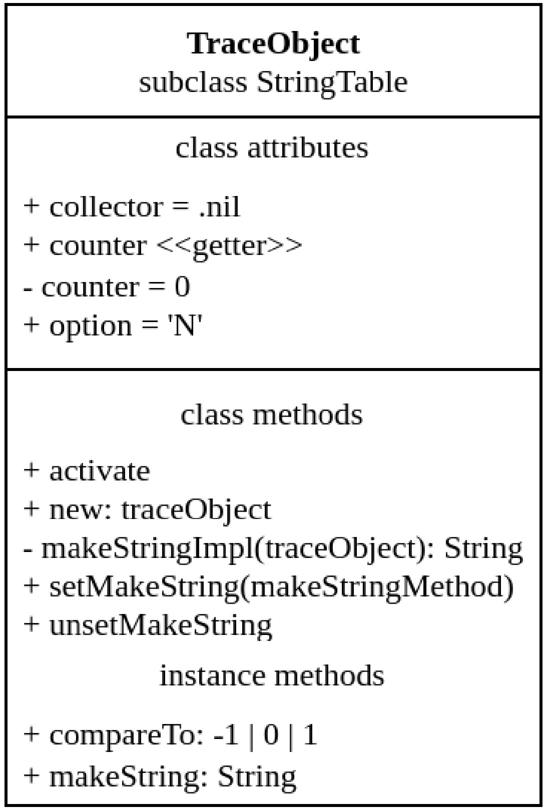
- ooRexx added new trace prefixes for new ooRexx features, e.g.
  - TRACE option All, Labels, Results
    - >I> (invocation entry)
    - <I< (invocation exit)
  - TRACE option Intermediates
    - >E> (name and value of an environment symbol)
    - >M> (name and result of a message)
    - >N> (name and result of a name-prefixed symbol)
    - >R> (name of argument and name of referenced variable)

ooRexx and TRACE Information, 2

- Missing information about
  - Rexx interpreter instances
    - E.g. each JavaFX scene gets controlled by a separate ooRexx interpreter instance
  - Invocation ID
  - Thread ID
  - Attribute pool ID
    - Attributes and methods of the same class scope share the same instance attribute pool
    - Access of attribute pools is controlled by
      - Guard state of attributes and methods
      - Guard lock owner
      - Guard lock count (scope lock count)
- Overwhelming, not all information is always needed!

ooRexx 5.1.0 TraceObject Class, 1

- Subclassing StringTable (a map collection with a string index)



TraceObject
subclass StringTable

class attributes

+ collector = .nil
+ counter <<getter>>
- counter = 0
+ option = 'N'

class methods

+ activate
+ new: traceObject
- makeStringImpl(traceObject): String
+ setMakeString(makeStringMethod)
+ unsetMakeString

instance methods

+ compareTo: -1 | 0 | 1
+ makeString: String

ooRexx 5.1.0 TraceObject Class, 2

- Class Attributes
  - collector, .nil (null) by default
    - If object is assigned it will get each created traceObject appended
      - Assigned object must understand the message append (any ordered collection is able to)
  - counter (getter) returns current count of trace objects created so far
  - option
    - N ("normal", display traceline)
    - T ("thread", like N, but inject thread ID in trace prefix)
    - S ("standard", like N, but prepend extended trace information in brackets indicating the thread ID, the invocation ID, in case of a method in addition the attribute pool ID, the method's defined and current guard state, the lock count, guard lock reserved indicator and waiting state)
    - F ("full", like S, but include Rexx interpreter instance ID in addition)
    - P ("profiling", "probing", allow collecting trace objects, but do not display traceline)

ooRexx 5.1.0 TraceObject Class, 3

- Class Methods
  - makeStringImpl, default formatting of trace output, handles all option settings
  - setMakeString, allows to change the makeString method to use
  - unsetMakeString, reverts to makeStringImpl
- Instance Methods
  - makeString: renders traceObject to a string using the currently set class method (makeStringImpl or the method supplied to setMakeString)
    - ooRexx will request a string rendering from objects under certain circumstances, e.g. when using a SAY keyword instruction on an object, by sending it the makeString message

ooRexx 5.1.0 TraceObject Class, 4

- Allows adding any number of trace related information in the trace object, like
  - ATTRIBUTEPOOL (ID number, makeStringImpl prefixes a 'A')
  - CALLERSTACKFRAME (a StringTable with the caller's stackframe information)
  - HASSCOPELOCK (boolean, if true makeStringImpl displays '*', a blank else)
  - INTERPRETER (ID number, makeStringImpl prefixes a 'R')
  - INVOCATION (ID number, makeStringImpl prefixes a 'I')
  - ISGUARDED (boolean, true if method is defined to be guarded, then makeStringImpl displays a 'G', a blank else)
  - ISWAITING (boolean, if true makeStringImpl displays a 'W', a blank else)
  - NUMBER (sequence number)
  - OPTION (the option character at time of creation)

ooRexx 5.1.0 TraceObject Class, 5

- RECEIVER (the receiver object for which the method runs)
- SCOPELOCKCOUNT (number, the current guard lock count, makeStringImpl prefixes a 'L')
- STACKFRAME (a StringTable with the stackframe information)
- THREAD (ID number, makeStringImpl prefixes a 'T')
- TIMESTAMP (DateTime of object creation)
- TRACELINE (the formatted trace line)
- VARIABLE (a StringTable with the name, value and usage)

ooRexx 5.1.0

.TraceObject~option="Normal"

```
t1=.test~new       -- create an instance
t2=.test~new       -- create another instance
t1~hey
t2~ho
say "-"~copies(50)
t2~start('hey')
t1~start('ho')
say "-"~copies(50)

::class test       -- some Rexx class
::method hey       -- by default: guarded
  say 'hey (guarded)'

::method ho unguarded    -- unguarded
  say 'ho (unguarded)'

::options trace all
```

Output:

```
     1 *-* t1=.test~new      -- instance
     2 *-* t2=.test~new      -- instance
     3 *-* t1~hey
       >I> Method "HEY" with scope "TEST"
    12 *-* say 'hey (guarded)'
hey (guarded)
       <I< Method "HEY" with scope "TEST"
     4 *-* t2~ho
       >I> Method "HO" with scope "TEST"
    15 *-* say 'ho (unguarded)'
ho (unguarded)
       <I< Method "HO" with scope "TEST"
     5 *-* say "-"~copies(50)
--------------------------------------------------
     6 *-* t2~start('hey')   -- async
     7 *-* t1~start('ho')    -- async
     8 *-* say "-"~copies(50)
--------------------------------------------------
       >I> Method "HEY" with scope "TEST"
    12 *-* say 'hey (guarded)'
hey (guarded)
       >I> Method "HO" with scope "TEST"
    15 *-* say 'ho (unguarded)'
ho (unguarded)
       <I< Method "HEY" with scope "TEST"
       <I< Method "HO" with scope "TEST"
```

ooRexx 5.1.0

.TraceObject~option="Thread"

```
t1=.test~new       -- create an instance
t2=.test~new       -- create another instance
t1~hey
t2~ho
say "-"~copies(50)
t2~start('hey')
t1~start('ho')
say "-"~copies(50)

::class test       -- some Rexx class
::method hey       -- by default: guarded
  say 'hey (guarded)'

::method ho unguarded    -- unguarded
  say 'ho (unguarded)'

::options trace all
```

Output:

```
     1 *-2* t1=.test~new      -- instance
     2 *-2* t2=.test~new      -- instance
     3 *-2* t1~hey
       >I2> Method "HEY" with scope "TEST"
    12 *-2* say 'hey (guarded)'
hey (guarded)
       <I2< Method "HEY" with scope "TEST"
     4 *-2* t2~ho
       >I2> Method "HO" with scope "TEST"
    15 *-2* say 'ho (unguarded)'
ho (unguarded)
       <I2< Method "HO" with scope "TEST"
     5 *-2* say "-"~copies(50)
--------------------------------------------------
     6 *-2* t2~start('hey')   -- async
     7 *-2* t1~start('ho')    -- async
     8 *-2* say "-"~copies(50)
--------------------------------------------------
       >I3> Method "HEY" with scope "TEST"
    12 *-3* say 'hey (guarded)'
hey (guarded)
       >I4> Method "HO" with scope "TEST"
    15 *-4* say 'ho (unguarded)'
ho (unguarded)
       <I3< Method "HEY" with scope "TEST"
       <I4< Method "HO" with scope "TEST"
```

ooRexx 5.1.0

.TraceObject~option="Standard"

```
t1=.test~new       -- create an instance
t2=.test~new       -- create another instance
t1~hey
t2~ho
say "-"~copies(50)
t2~start('hey')
t1~start('ho')
say "-"~copies(50)

::class test       -- some Rexx class
::method hey       -- by default: guarded
  say 'hey (guarded)'

::method ho unguarded    -- unguarded
  say 'ho (unguarded)'

::options trace all
```

Output:

```
[T2 I2]                       1 *-* t1=.test~new      -- instance
[T2 I2]                       2 *-* t2=.test~new      -- instance
[T2 I2]                       3 *-* t1~hey
[T2 I3 Gu A1 L0   ]             >I> Method "HEY" with scope "TEST"
[T2 I3 G  A1 L1 * ]          12 *-* say 'hey (guarded)'
hey (guarded)
[T2 I3 Gu A1 L0   ]             <I< Method "HEY" with scope "TEST"
[T2 I2]                       4 *-* t2~ho
[T2 I4 U  A2 L0   ]             >I> Method "HO" with scope "TEST"
[T2 I4 U  A2 L0   ]          15 *-* say 'ho (unguarded)'
ho (unguarded)
[T2 I4 U  A2 L0   ]             <I< Method "HO" with scope "TEST"
[T2 I2]                       5 *-* say "-"~copies(50)
--------------------------------------------------
[T2 I2]                       6 *-* t2~start('hey')   -- async
[T2 I2]                       7 *-* t1~start('ho')    -- async
[T2 I2]                       8 *-* say "-"~copies(50)
--------------------------------------------------
[T3 I5 Gu A2 L0   ]             >I> Method "HEY" with scope "TEST"
[T3 I5 G  A2 L1 * ]          12 *-* say 'hey (guarded)'
hey (guarded)
[T4 I6 U  A1 L0   ]             >I> Method "HO" with scope "TEST"
[T4 I6 U  A1 L0   ]          15 *-* say 'ho (unguarded)'
ho (unguarded)
[T3 I5 Gu A2 L0   ]             <I< Method "HEY" with scope "TEST"
[T4 I6 U  A1 L0   ]             <I< Method "HO" with scope "TEST"
```

ooRexx 5.1.0

.TraceObject~option="Full"

```
t1=.test~new        -- create an instance
t2=.test~new        -- create another instance
t1~hey
t2~ho
say "-"~copies(50)
t2~start('hey')
t1~start('ho')
say "-"~copies(50)

::class test        -- some Rexx class
::method hey        -- by default: guarded
  say 'hey (guarded)'

::method ho unguarded    -- unguarded
  say 'ho (unguarded)'

::options trace all
```

Output:

```
[R1 T2 I2]                     1 *-* t1=.test~new      -- instance
[R1 T2 I2]                     2 *-* t2=.test~new      -- instance
[R1 T2 I2]                     3 *-* t1~hey
[R1 T2 I3 Gu A1 L0   ]           >I> Method "HEY" with scope "TEST"
[R1 T2 I3 G  A1 L1 * ]        12 *-* say 'hey (guarded)'
hey (guarded)
[R1 T2 I3 Gu A1 L0   ]           <I< Method "HEY" with scope "TEST"
[R1 T2 I2]                     4 *-* t2~ho
[R1 T2 I4 U  A2 L0   ]           >I> Method "HO" with scope "TEST"
[R1 T2 I4 U  A2 L0   ]        15 *-* say 'ho (unguarded)'
ho (unguarded)
[R1 T2 I4 U  A2 L0   ]           <I< Method "HO" with scope "TEST"
[R1 T2 I2]                     5 *-* say "-"~copies(50)
--------------------------------------------------
[R1 T2 I2]                     6 *-* t2~start('hey')   -- async
[R1 T2 I2]                     7 *-* t1~start('ho')    -- async
[R1 T2 I2]                     8 *-* say "-"~copies(50)
--------------------------------------------------
[R1 T3 I5 Gu A2 L0   ]           >I> Method "HEY" with scope "TEST"
[R1 T3 I5 G  A2 L1 * ]        12 *-* say 'hey (guarded)'
hey (guarded)
[R1 T4 I6 U  A1 L0   ]           >I> Method "HO" with scope "TEST"
[R1 T4 I6 U  A1 L0   ]        15 *-* say 'ho (unguarded)'
ho (unguarded)
[R1 T3 I5 Gu A2 L0   ]           <I< Method "HEY" with scope "TEST"
[R1 T4 I6 U  A1 L0   ]           <I< Method "HO" with scope "TEST"
```

TraceTool, 1

- tracetool.rex (front end) using traceutil.cls (an ooRexx package)
- Takes advantage of the new TraceObject class in ooRexx 5.1.0
  - All traces of a Rexx and ooRexx program get collected
  - The traced Rexx/ooRexx program gets executed on a separate thread
- The collected trace information gets saved as a *tracelog* text file
  - Rendered as XML, JSON or CSV
  - A *tracelog* can be converted from one encoding to the other
- The *tracelog* can be replayed in different trace formats and orderings
  - Allows for analyzing the execution of Rexx programs
  - Even the most complex multi-threaded ooRexx programs

TraceTool, 2

- The *tracelog* can be used for
  - Creating a profile of the traced Rexx program and optionally
  - Creating a SQL batch file from the profile data for SQL RDBMS like SQLite
    - This includes DDL statements
    - By default SQLite is supported, one can therefore use SQLite or ooRexx' ooSQLite
- Analyzing the execution of complex Rexx and ooRexx programs, e.g.
  - Deadlocked multi-threaded ooRexx programs
  - The dynamics of a multi-threaded ooRexx program
  - ... and much more
- TraceTool allows for adding or removing global trace settings
  - ::options trace xyz

TraceTool, 3

- tracetool.rex is a command line tool
  - To get all available options just enter "tracetool.rex"
  - Main options
    - **-t** ... run a Rexx/ooRexx program and create a tracelog
      - One can supply optionally a global trace option
    - **-s** ... shows/replays the tracelog
      - One can supply optionally a desired formatting option
    - **-c** ... convert a tracelog file to xml, json or csv
    - **-p** ... analyze a tracelog and give overview profiling information
      - One can optionally have a SQL script created to allow profiling and analyzing off a RDBMS
    - **-m** ... manage Rexx/ooRexx programs (allow to add or remove global trace options)

TraceTool Example

- The Rexx program "D:\tmp\anyprogram.rex" (could be any of your programs)
  - Simple program to keep number of slides low
  - Invokes two routines
    - An internal routine (represented by a Rexx label)
    - A directive routine (represented by the ::ROUTINE keyword)
  - Uses the ooRexx environment symbols
    - .line ... the line of the currently traced statement
    - .context~name ... queries the name of the current executable

```
say .line .context~name 'hello, world!'
call internalTestRoutine
call testRoutine
exit
internalTestRoutine:
  say .line .context~name
  return
::routine testRoutine
  say .line .context~name
  return
```

```
 1 D:\tmp\anyprogram.rex hello, world!
 6 INTERNALTESTROUTINE
 9 TESTROUTINE
```

Create a Tracelog, 1

- Main tracetool option is **-t** (create tracelog)
  - Can be optionally followed by the letter **a** (**a**ll), **i** (**i**ntermediate), **l** (**l**abels), **n** (**n**ormal, default), **r** (**r**esults)
- The example tracelog should be created with trace option **a**ll in effect for the entire program, hence
  - Add the trace option to the main option: **-ta**
  - By default the tracelog will be
    - In **xml** hence any xml tools can be used to process the tracelog including xslt
    - The default name will be the name of the program with the string "_trace.xml" appended, hence "anyprogram.rex_trace.xml"
  - Command: tracetool -ta anyprogram.rex

Create a Tracelog, 2

Command "tracetool -ta anyprogram.rex"

- Created tracelog: "D:\tmp\anyprogram.rex_trace.xml" (excerpt)

```
<traceLog>
   <traceObject>
      <option>P</option>
      <number>1</number>
      <timestamp>2025-05-03T15:46:52.442000</timestamp>
      <interpreter>0</interpreter>
      <thread>0</thread>
      <invocation>0</invocation>
      <lineNr>398</lineNr>
      <stackFrame>
         <arguments></arguments>
         <executableId>FFFFFE81_C03BB9AF</executableId>
         <executablePackage>D:\tmp\tracetool.rex</executablePackage>
         <invocation>2</invocation>
         <line>398</line>
         <name>traceLogStart</name>
         <package>tracetool.rex</package>
         <target>.nil</target>
         <thread>1</thread>
         <traceLine>.nil</traceLine>
         <type>ROUTINE</type>
      </stackFrame>
      <traceLine>       +++ tracetool.rex for [anyprogram.rex] (start collecting)</traceLine>
   </traceObject>
   <traceObject>
      <option>P</option>
      <number>2</number>
      <timestamp>2025-05-03T15:46:52.442000</timestamp>
… cut …
```

Show/Replay a Tracelog, 1

- Main tracetool option is **-s** (show/replay tracelog)
- By default a plain ("**n**ormal") trace will be displayed
- Using the format suboption **-f** followed by one of the letters **n** (**n**ormal, default), **t** (**t**hread), **s** (**s**tandard), **f** (**f**ull), **m1** (extensive information), **m2** (full formatting with ID widths one character), **m3** (full formatting with ID widths two characters wide) determines the formatting to use
- Using the order suboption **-o** followed by one of **n** (**n**ormal, default), **an** (**a**ttributepoolID, **n**umber), **ain** (**a**ttributepoolID, **i**nvocationID, **n**umber), **atn** (**a**ttributepoolID, **t**hreadID, **n**umber), **atin** (**a**ttributepoolID, **t**hreadID, **i**nvocationID, **n**umber), **in** (**i**nvocationID, **n**umber), **rtin** (**r**exxInterpreterID, **t**hreadID, **i**nvocationID, **n**umber)
- Command: `tracetool -s anyprogram.rex_trace.xml`

Show/Replay a Tracelog, 2
Command "tracetool -s anyprogram.rex_trace.xml"

- Tracelog "D:\tmp\anyprogram.rex_trace.xml" (default: normal format)

```
       +++ tracetool.rex for [anyprogram.rex] (start collecting)
       >I> Routine "anyprogram.rex" in package "anyprogram.rex".
     1 *-* say .line .context~name 'hello, world!'
     2 *-* call internalTestRoutine
     5 *-*   internalTestRoutine:
     6 *-*   say .line .context~name
     7 *-*   return
     3 *-* call testRoutine
       >I> Routine "TESTROUTINE" in package "anyprogram.rex".
     9 *-* say .line .context~name
    10 *-* return
       <I< Routine "TESTROUTINE" in package "anyprogram.rex".
     4 *-* exit
       <I< Routine "anyprogram.rex" in package "anyprogram.rex".
       +++ tracetool.rex for [anyprogram.rex] (end collecting)
```

Show/Replay a Tracelog, 3

Command "tracetool -s -fs anyprogram.rex_trace.xml"

- Tracelog "D:\tmp\anyprogram.rex_trace.xml" (**-fs** causes **s**tandard format)

```
[T0  I0 ]                      +++ tracetool.rex for [anyprogram.rex] (start collecting)
[T2  I3 ]                      >I> Routine "anyprogram.rex" in package "anyprogram.rex".
[T2  I3 ]                    1 *-* say .line .context~name 'hello, world!'
[T2  I3 ]                    2 *-* call internalTestRoutine
[T2  I4 ]                    5 *-*   internalTestRoutine:
[T2  I4 ]                    6 *-*   say .line .context~name
[T2  I4 ]                    7 *-*   return
[T2  I3 ]                    3 *-* call testRoutine
[T2  I5 ]                      >I> Routine "TESTROUTINE" in package "anyprogram.rex".
[T2  I5 ]                    9 *-* say .line .context~name
[T2  I5 ]                   10 *-* return
[T2  I5 ]                      <I< Routine "TESTROUTINE" in package "anyprogram.rex".
[T2  I3 ]                    4 *-* exit
[T2  I3 ]                      <I< Routine "anyprogram.rex" in package "anyprogram.rex".
[T0  I0 ]                      +++ tracetool.rex for [anyprogram.rex] (end collecting)
```

Convert Tracelog, 1

- Main tracetool option is **-c** (convert tracelog)
  - Followed by the suboption **-fx** (xml), **-fj** (json), or **-fc** (csv)
- The example tracelog should be converted to json
- Command: tracetool -c -fj anyprogram.rex_trace.xml

Convert Tracelog, 2

Command "tracetool -c -fj anyprogram.rex_trace.xml"

- Created tracelog: "D:\tmp\anyprogram.rex_trace.xml.converted.json" (excerpt)

```
[
   {
      "option": "P",
      "number": 1,
      "timestamp": "2025-05-03T15:46:52.442000",
      "interpreter": 0,
      "thread": 0,
      "invocation": 0,
      "lineNr": 398,
      "stackFrame": {
                "arguments": "",
                "executableId": "FFFFFE81_C03BB9AF",
                "executablePackage": "D:\\tmp\\tracetool.rex",
                "invocation": 2,
                "line": 398,
                "name": "traceLogStart",
                "package": "tracetool.rex",
                "target": null,
                "thread": 1,
                "traceLine": null,
                "type": "ROUTINE"
                },
      "traceLine": "       +++ tracetool.rex for [anyprogram.rex] (start collecting)"
   },
   {
      "option": "P",
      "number": 2,
      "timestamp": "2025-05-03T15:46:52.442000",

… cut …
```

Profile Tracelog, 1

- Main tracetool option is **-p** (profile tracelog)
- By default displays the call hierarchy to the default depth 7, and the relative percentage referring the total duration and the group's duration
- Option **-s** (sql) creates a commented SQL script that creates tables and views, the tracelog's INSERT statements, and example SELECT statements
  - Option **-sl** creates the SQL script specifically for SQLite which can be processed by sqlite3 or oosqlite (ooRexx' sqlite3 library)
- Command: `tracetool.rex -p anyprogram.rex_trace.xml`
- Command: `tracetool.rex -p -sl anyprogram.rex_trace.xml`
  - Creates a SQLite SQL script and shows how to use the generated SQL script

Profile Tracelog, 2

Command "tracetool.rex -p anyprogram.rex_trace.xml"

- Output

```
profiling needs to analyze 15 traceObjects ...
>>> ordered descendingly by duration of executables (% of total duration):
-------------------------------------------------------------------------
  0.00% 00:00:00.000000 called      1 times/calling   1 execs [anyprogram.rex (ROUTINE L# 1 in anyprogram.rex)]
  0.00% 00:00:00.000000 called      1 times/calling   0 execs [anyprogram.rex/INTERNALTESTROUTINE (INTERNALCALL L# 5 in anyprogram.rex)]
  0.00% 00:00:00.000000 called      1 times/calling   0 execs [TESTROUTINE (ROUTINE L# 9 in anyprogram.rex)]

>>> aggregated duration call tree (% of global total duration):
---------------------------------------------------------------
  0.00% 00:00:00.000000 called      1 times/calling   1 execs [anyprogram.rex (ROUTINE L# 1 in anyprogram.rex)]
      0.00% 00:00:00.000000 called      1 times/calling   0 execs [TESTROUTINE (ROUTINE L# 9 in anyprogram.rex)]

>>> averaged duration call tree (% of global total duration):
-------------------------------------------------------------
  0.00% 00:00:00.000000 called      1 times 00:00:00.000000/calling   1 execs [anyprogram.rex (ROUTINE L# 1 in anyprogram.rex)]
      0.00% 00:00:00.000000 called      1 times 00:00:00.000000/calling   0 execs [TESTROUTINE (ROUTINE L# 9 in anyprogram.rex)]

>>> aggregated duration call tree (% of group's total duration):
----------------------------------------------------------------
  0.00% 00:00:00.000000 called      1 times/calling   1 execs [anyprogram.rex (ROUTINE L# 1 in anyprogram.rex)]
      0.00% 00:00:00.000000 called      1 times/calling   0 execs [TESTROUTINE (ROUTINE L# 9 in anyprogram.rex)]

>>> averaged duration call tree (% of group's total duration):
--------------------------------------------------------------
  0.00% 00:00:00.000000 called      1 times 00:00:00.000000/calling   1 execs [anyprogram.rex (ROUTINE L# 1 in anyprogram.rex)]
      0.00% 00:00:00.000000 called      1 times 00:00:00.000000/calling   0 execs [TESTROUTINE (ROUTINE L# 9 in anyprogram.rex)]

… cut …
```

Profile Tracelog, 3

Command "tracetool.rex -p -sl anyprogram.rex_trace.xml"

- Output

```
profiling needs to analyze 15 traceObjects ...
>>> ordered descendingly by duration of executables (% of total duration):
-------------------------------------------------------------------------
  0.00% 00:00:00.000000 called      1 times/calling   1 execs [anyprogram.rex (ROUTINE L# 1 in anyprogram.rex)]
  0.00% 00:00:00.000000 called      1 times/calling   0 execs [anyprogram.rex/INTERNALTESTROUTINE (INTERNALCALL L# 5 in anyprogram.rex)]
  0.00% 00:00:00.000000 called      1 times/calling   0 execs [TESTROUTINE (ROUTINE L# 9 in anyprogram.rex)]

>>> aggregated duration call tree (% of global total duration):
---------------------------------------------------------------
  0.00% 00:00:00.000000 called      1 times/calling   1 execs [anyprogram.rex (ROUTINE L# 1 in anyprogram.rex)]
      0.00% 00:00:00.000000 called      1 times/calling   0 execs [TESTROUTINE (ROUTINE L# 9 in anyprogram.rex)]

>>> averaged duration call tree (% of global total duration):
-------------------------------------------------------------
  0.00% 00:00:00.000000 called      1 times 00:00:00.000000/calling   1 execs [anyprogram.rex (ROUTINE L# 1 in anyprogram.rex)]
      0.00% 00:00:00.000000 called      1 times 00:00:00.000000/calling   0 execs [TESTROUTINE (ROUTINE L# 9 in anyprogram.rex)]

>>> aggregated duration call tree (% of group's total duration):
----------------------------------------------------------------
  0.00% 00:00:00.000000 called      1 times/calling   1 execs [anyprogram.rex (ROUTINE L# 1 in anyprogram.rex)]
      0.00% 00:00:00.000000 called      1 times/calling   0 execs [TESTROUTINE (ROUTINE L# 9 in anyprogram.rex)]
… cut …
3181: CREATESQL - depending on the size of the tracelog (15 items) this may take a while ...

created sql script [anyprogram_rex_trace_20250503_154652_db.sql] for database [anyprogram_rex_trace_20250503_154652_db] based on
[anyprogram.rex_trace.xml]
    sqlite3 anyprogram_rex_trace_20250503_154652_db.db < anyprogram_rex_trace_20250503_154652_db.sql
```

Profile Tracelog, 4

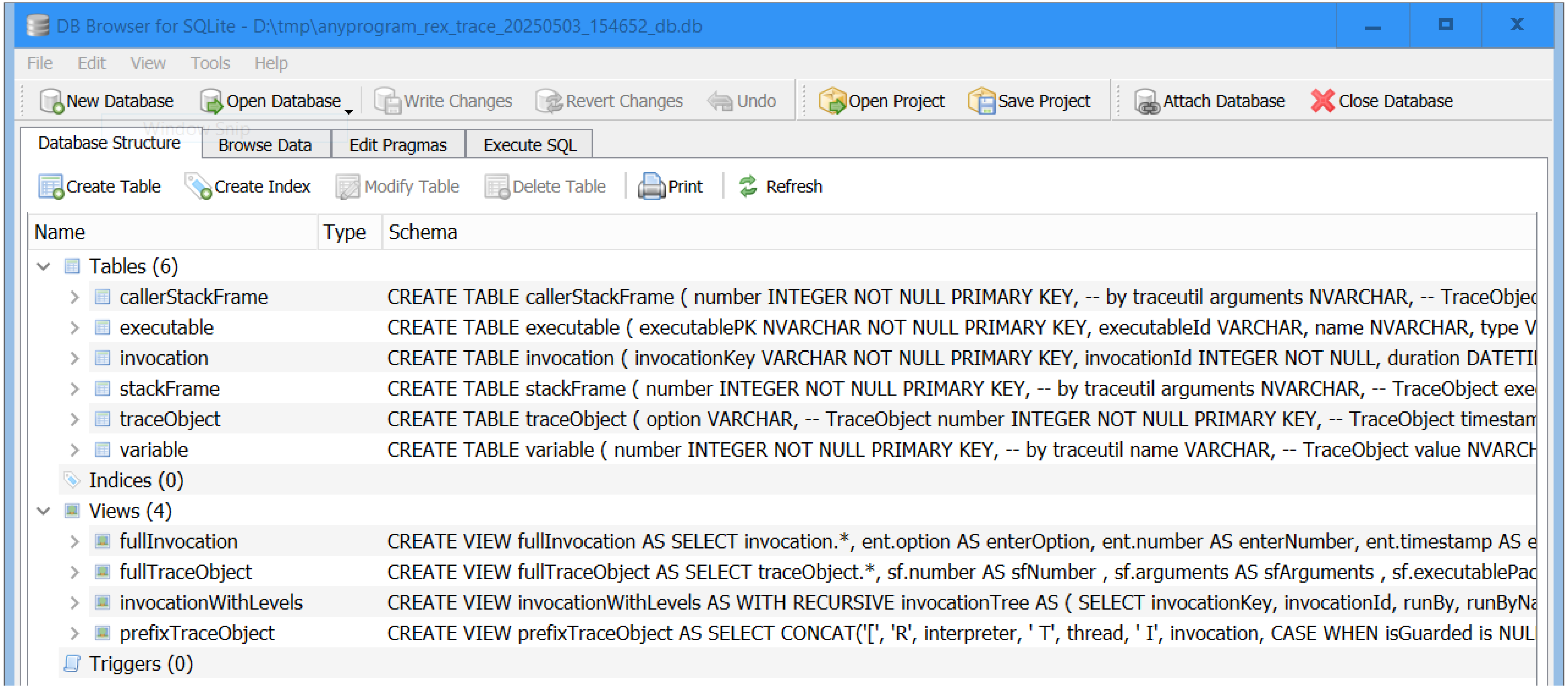
Employing sqlite3 to Create And Use a RDBMS From the Tracelog

- Output

```
D:\tmp>sqlite3 anyprogram_rex_trace_20250503_154652_db.db < anyprogram_rex_trace_20250503_154652_db.sql
.mode box --wrap 120

/* ------------------------------------------------------------------------- */
-- show the first 10 traceLines

SELECT traceLine
FROM   traceObject
ORDER BY number
LIMIT 10;
┌─────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐
│                            traceLine                                │
├─────────────────────────────────────────────────────────────────────────────────────────────────────────────────┤
│         +++ tracetool.rex for [anyprogram.rex] (start collecting)  │
│         >I> Routine "anyprogram.rex" in package "anyprogram.rex".  │
│       1 *-* say .line .context~name 'hello, world!'                │
│       2 *-* call internalTestRoutine                               │
│       5 *-*   internalTestRoutine:                                 │
│       6 *-*   say .line .context~name                              │
│       7 *-*   return                                               │
│       3 *-* call testRoutine                                       │
│         >I> Routine "TESTROUTINE" in package "anyprogram.rex".     │
│       9 *-* say .line .context~name                                │
└─────────────────────────────────────────────────────────────────────────────────────────────────────────────────┘

…. cut …
```

Profile Tracelog, 5

Structure of the SQLite RDBMS

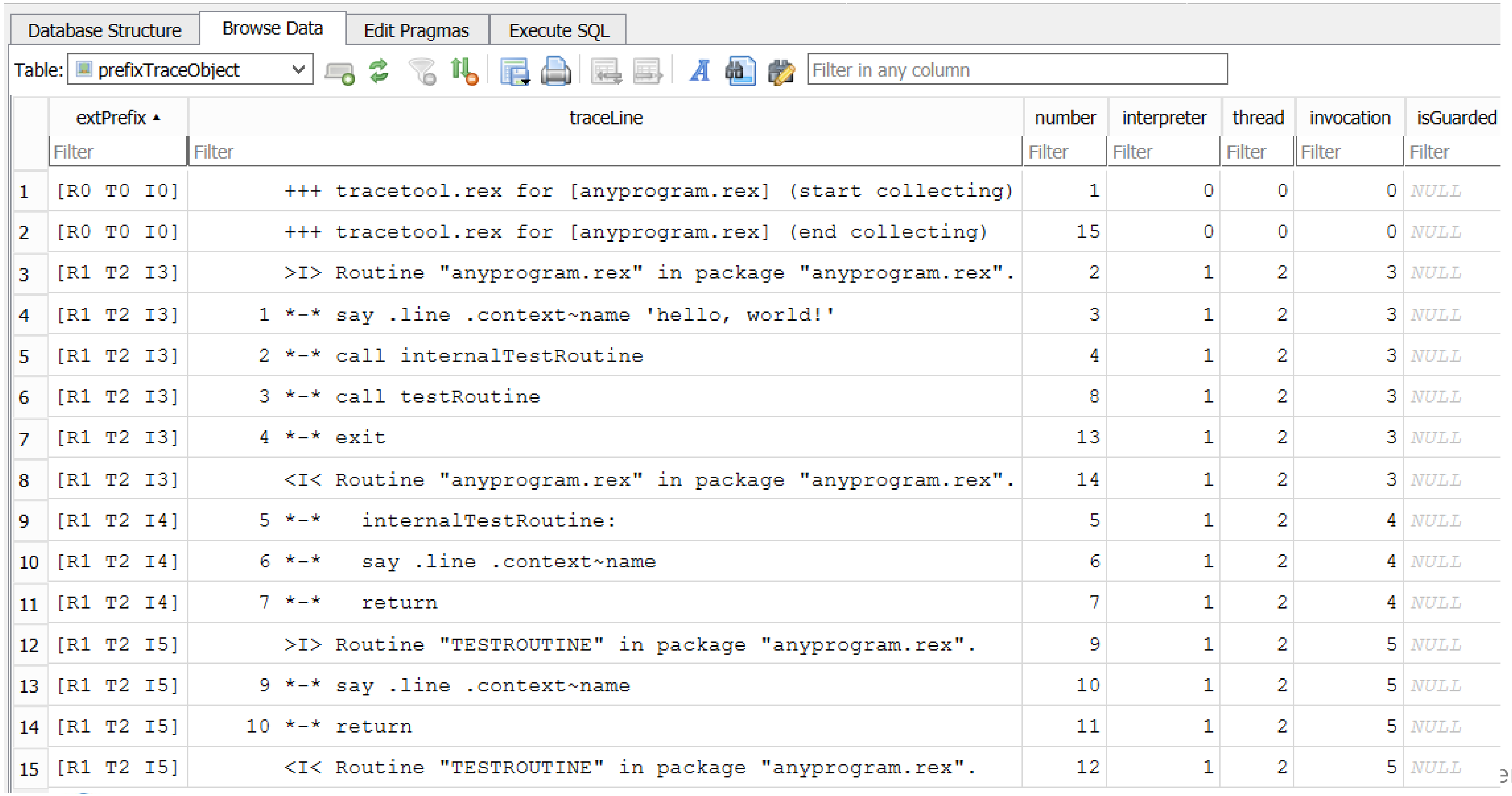


DB Browser for SQLite - D:\tmp\anyprogram_rex_trace_20250503_154652_db.db

Name	Type	Schema
Tables (6)		
callerStackFrame		CREATE TABLE callerStackFrame (number INTEGER NOT NULL PRIMARY KEY, -- by traceutil arguments NVARCHAR, -- TraceObjec
executable		CREATE TABLE executable (executablePK NVARCHAR NOT NULL PRIMARY KEY, executableId VARCHAR, name NVARCHAR, type V
invocation		CREATE TABLE invocation (invocationKey VARCHAR NOT NULL PRIMARY KEY, invocationId INTEGER NOT NULL, duration DATETI
stackFrame		CREATE TABLE stackFrame (number INTEGER NOT NULL PRIMARY KEY, -- by traceutil arguments NVARCHAR, -- TraceObject exe
traceObject		CREATE TABLE traceObject (option VARCHAR, -- TraceObject number INTEGER NOT NULL PRIMARY KEY, -- TraceObject timestan
variable		CREATE TABLE variable (number INTEGER NOT NULL PRIMARY KEY, -- by traceutil name VARCHAR, -- TraceObject value NVARCH
Indices (0)		
Views (4)		
fullInvocation		CREATE VIEW fullInvocation AS SELECT invocation.*, ent.option AS enterOption, ent.number AS enterNumber, ent.timestamp AS e
fullTraceObject		CREATE VIEW fullTraceObject AS SELECT traceObject.*, sf.number AS sfNumber , sf.arguments AS sfArguments , sf.executablePac
invocationWithLevels		CREATE VIEW invocationWithLevels AS WITH RECURSIVE invocationTree AS (SELECT invocationKey, invocationId, runBy, runByNa
prefixTraceObject		CREATE VIEW prefixTraceObject AS SELECT CONCAT('[', 'R', interpreter, ' T', thread, ' I', invocation, CASE WHEN isGuarded is NUL
Triggers (0)		

Profile Tracelog, 6

View "prefixTraceObject"



Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: prefixTraceObject

Filter in any column

	extPrefix	traceLine	number	interpreter	thread	invocation	isGuarded
1	[R0 T0 I0]	`+++ tracetool.rex for [anyprogram.rex] (start collecting)`	1	0	0	0	NULL
2	[R0 T0 I0]	`+++ tracetool.rex for [anyprogram.rex] (end collecting)`	15	0	0	0	NULL
3	[R1 T2 I3]	`>I> Routine "anyprogram.rex" in package "anyprogram.rex".`	2	1	2	3	NULL
4	[R1 T2 I3]	`1 *-* say .line .context~name 'hello, world!'`	3	1	2	3	NULL
5	[R1 T2 I3]	`2 *-* call internalTestRoutine`	4	1	2	3	NULL
6	[R1 T2 I3]	`3 *-* call testRoutine`	8	1	2	3	NULL
7	[R1 T2 I3]	`4 *-* exit`	13	1	2	3	NULL
8	[R1 T2 I3]	`<I< Routine "anyprogram.rex" in package "anyprogram.rex".`	14	1	2	3	NULL
9	[R1 T2 I4]	`5 *-*   internalTestRoutine:`	5	1	2	4	NULL
10	[R1 T2 I4]	`6 *-*   say .line .context~name`	6	1	2	4	NULL
11	[R1 T2 I4]	`7 *-*   return`	7	1	2	4	NULL
12	[R1 T2 I5]	`>I> Routine "TESTROUTINE" in package "anyprogram.rex".`	9	1	2	5	NULL
13	[R1 T2 I5]	`9 *-* say .line .context~name`	10	1	2	5	NULL
14	[R1 T2 I5]	`10 *-* return`	11	1	2	5	NULL
15	[R1 T2 I5]	`<I< Routine "TESTROUTINE" in package "anyprogram.rex".`	12	1	2	5	NULL

Manage Global Tracing

- Main tracetool option is **-m** (manage global tracing)
- Allows to add, query and remove global trace options to a group of Rexx programs

  -a or **-ar** (trace **r**esults), **-ai** (trace **i**ntermediates), **-al** (trace **l**abels), **-an** (trace **n**ormal)

  -q query whether files have a tracetool global trace option set

  -d delete any tracetool global trace options
- Optionally apply action recursively using the option **-r**
- Optional file pattern string, defaults to "*.rex *.cls *.frm *.rxj *.rxo"
- Command to add global trace results to all .rex and .cls files:

  ```
  tracetool -m -a "*.rex *.cls"
  ```

Roundup

- REXX
  - TRACE keyword instruction and built-in-function (BIF)
  - Trace options: "normal", "all", "results", "intermediates"
- ooRexx 5.1.0
  - Introduces the .TraceObject class as a subclass of .StringTable
    - Global configuration
    - Each traced instruction causes a trace object to be created with the trace information
    - Allows creating custom makeString methods for formatting the tracelines
  - Allows for gaining full insight into multithreaded execution and locking
    - For the first time possible to analyze hanging ooRexx programs in depth
      - Run the hanging program on a separate thread after configuring .TraceObject
      - After a predefined timeout, analyze the collected trace objects
  - Allows for creating trace logs for later analysis