

The New Rexx Debugger



2025 International Rexx Language Association Symposium,

University of Vienna, Vienna,

6th May 2025

By Dominic Wise

Dominic Wise

The New Rexx Debugger



Presentation overview

- What is “the New Rexx Debugger”
- Why create this at all
- What interpreter features does it leverage / why ooRexx ?
- Cross platform considerations
- Features / demonstration
- Links and Questions

The New Rexx Debugger



What is it

- A dialog based graphical debugger for Rexx programs
 - Source code, stack and variable views
 - Single step and run to breakpoint features
 - Console for interacting with the program and debugger
 - Standalone and embedded program can be debugged
 - Two modes – direct / embedded launch or debugger launch
- Written in Rexx – no compiled code
 - Runs in-interpreter on ooRexx 4.2 or later
- Cross platform - Windows, Linux and MacOS

The New Rexx Debugger



What is it – Windows

The screenshot shows the ooRexx Debugger interface with the following components:

- Source code:** A window titled 'tutorial.rex' containing REXX code. Line 79 is highlighted: `79 SAY 'In the newly added watch window for DIR, double click c`.
- Call stack:** A window showing the current execution point: `79 *- * SAY 'In the newly added watch window for DIR, double c`.
- Display console:** A window with instructions: 'The start of each line shows the index of the start of that block in the string', 'A + sign at the start of a watch variable means it is a collection that can be expanded.', 'In the Watch window, double click on STEMVAR to see its components', 'Arrays are collections so can be expanded, including multi-dimensional arrays', 'Click Run if you want to skip array set up', 'In the Watch window, double click on MULTIDIMARRAY', 'In the Watch window, double click on "DIR"', 'Recursive watch drilldown is possible for all collection types', 'In the newly added watch window for DIR, double click on "ArrayThing"'. Buttons for Next, Run, Exit, Watch, Help, Open, and Exec are visible.
- Input console:** An empty window for user input.

Source code →

Call stack →

Display console →

Input console →

Watch windows to display top level variables and expanded collection objects

The New Rexx Debugger



What is it – Linux

The screenshot displays the ooRexx Debugger Version 1.36 interface. The main window shows the source code for tutorial.rex, with line 52 highlighted: `* 52 z = 'Here is'.endofline'a multiple'.endofline'line string'`. Below the code is a console window showing the execution progress, including the message "Loop finished" and the current state `z2 = z`. To the right, three watch windows are open: "Watch" showing the current values of variables I, X, Y, Z, and Z2; "Watch Z" showing the string value of Z; and "Watch Z2" showing the hexadecimal byte representation of the string Z.

Different ways
of viewing string
variables

The New Rexx Debugger



What is it – MacOS

The screenshot shows the ooRexx Debugger Version 1.35 interface. The main window displays a REXX script named 'tutorial.rex' with the following code:

```
23 CALL SAY 'With this set the breakpoint will be ignored until the loop variable i has the value 5'
24 CALL SAY 'you should see 1-5 printed out before the breakpoint is hit, after which it won't be hit again'
25 CALL SAY 'Hit the Run button twice to see the conditional breakpoint in action'
26 CALL SAY 'Running loop 8 times'
27 do i = 1 to 8
28   CALL SAY i
29   /**/ NOP
30 end
31 CALL SAY 'Loop finished'
32
33 /**/CALL SAY 'Enter the loop variable i'
34 CALL SAY 'The loop variable i is ' || i
35 CALL SAY 'Enter the loop variable i'
36 SAY 'Enter "CAPTURE" to capture the loop variable i'
37 SAY 'Type "CAPTURE" to capture the loop variable i'
38 SAY 'Trace output to the console'
39 SAY 'Type "SAY 3" to say the loop variable i'
40 SAY 'Press the Enter key to continue'
41 SAY 'Clear the console'
42 SAY 'Play around'
43 NOP
```

A 'Breakpoint Hit' dialog box is overlaid on the code editor, showing the following options:

- Always
- When

The 'When' option is selected, and the condition 'i=5' is entered in the text field below. The dialog box has 'Ok' and 'Cancel' buttons.

Below the code editor, there is a status bar showing '27 *-* do i = 1 to 8'. At the bottom of the debugger window, there is a help text area with the following instructions:

An empty comment at the start of a traceable line will cause the debugger to insert a breakpoint
Double clicking on the following line will insert a ? breakpoint marker because the line isn't traceable
Double click on any breakpoint line above to remove the breakpoint
Breakpoints can have conditions set on them via a right-click action when they are selected
Click on line 29 then use the right-click action to set the condition i=5 for the breakpoint
With this set the breakpoint will be ignored until the loop variable i has the value 5 so when running you should see 1-5 printed out before the breakpoint is hit, after which it won't be hit again
Hit the Run button twice to see the conditional breakpoint in action
Running loop 8 times

On the right side of the debugger window, there are several buttons: Next, Run, Exit, Watch, Help, Open, and Exec.

Breakpoint settings dialog

The New Rexx Debugger



Why was it created

- Rexx has powerful debugging features already
 - TRACE
 - Report running program status
 - Interactive (after each instruction)
 - View & update variables, skip instructions
- I wanted a GUI debugger for Rexx and couldn't find one
 - Mostly a Windows developer with C/C++ and I like GUI debuggers
 - I work with (oo)Rexx as an embedded application interfacing engine
 - History, going back to OS/2 is mainly with a focus on ANSI-like Rexx features
 - The initial target was embedded use on Windows (my use case)

The New Rexx Debugger



Why was it created - continued

- Advent of code 2023
 - Set of coding puzzles over December
 - Attempted this in ooRexx
 - Didn't get especially far
 - Further increased my desire for a GUI debugger
 - Discovered and started to use ooRexx features
 - Became aware of language and platform features that might make an ooRexx debugger possible

The New Rexx Debugger



What interpreter features – why ooRexx

- Functionality a GUI debugger needs from the interpreter
 - Ability to take control of a running program
 - Ability to view and manage program state
 - Non-intrusive – isolation of debugger program state
 - A UI library
 - Some form of multi-threading support (generally, for UI)

The New Rexx Debugger



Required features - Taking control of the program

- Option: Use of Rexx system exit interfaces
 - Part of the standard Rexx programming API
 - Registered by the program running the interpreter
 - Can halt after every instruction
 - Can intercept interactive trace input responses along with program and trace output
- An interpreter wrapper program needs to be built
- Requires modification of embedded engines which may not be possible

The New Rexx Debugger



Required features - Taking control of the program

- Option: Leverage language hooks to intercept tracing
- Interpreter effectively calls PULL on an input stream
 - In ooRexx, this can be intercepted to call ooRexx code
 - This uses the `.DEBUGINPUT` global object
- How ? Intercept standard ooRexx I/O
 - Standard stream objects `.STDIO`, `.STDIN`, `STDERR` etc exist
Have methods like `LINEOUT`, `CHAROUT`, `LINEIN` etc

Instead of SAY: `.STDOUT~LINEOUT` string

Instead of PULL: `str = .STDIN~LINEIN`

Dominic Wise

The New Rexx Debugger



Required features - Taking control of the program

- An extra level, the IO stream can be accessed “Monitors”
 - Monitors are literally placeholders for other objects
 - A target object is assigned to the monitor
 - The target object can be queried and modified
 - All method calls are forwarded to the current target object
 - The set of IO monitors
 - `.INPUT` redirects to the `.STDIN` object
 - `.OUTPUT` redirects to `.STDOUT` object
 - `.ERROR` redirects to `.STDERR`
 - `.TRACEOUTPUT` (trace output) redirects to `.ERROR`
 - **`.DEBUGINPUT` (interactive trace input)**

The New Rexx Debugger



Required features - Taking control of the program

For normal(*) IO processing ooRexx will use the monitor objects and not the underlying IO objects so:

Instead of SAY: `. OUTPUT~LINEOUT string`

Instead of PULL: `str = . INPUT~LINEIN`

(*) The interpreter wrapper may have set up API system exits which tell ooRexx NOT to call these – can be an issue when embedding

This is true for stdout and trace output for my organisation's application

The New Rexx Debugger



Required features - Taking control of the program

During interactive tracing, `.DEBUGINPUT~LINEIN` will be called to get user response (unless blocked by a registered API system exit). The following is a customized interactive debugger utilising this:

```
.DEBUGINPUT~destination(.debuginputinterceptor~new)
trace ?A /* Trace after every instruction */
do i = 1 to 10
  say i
end

::class debuginputinterceptor
::method linein
say 'Press enter to continue or type code to run'
parse pull data
return data
```

The New Rexx Debugger



Required features –Managing program state

- Program state requirements
 - Current location
 - Source file (to load the correct file)
 - Current line for tracking and breakpoints
 - Stack trace
 - Program variables e.g. for debugger Watch windows
 - Error state
- All accessible from the dynamically created .context object executable, line, variables , stackframes, condition

The New Rexx Debugger



Required features – Managing program state

- Communication via `.local~debug.channels` object
 - Per-thread debug state (one channel per thread)
 - Accessible to both program and debugger
 - Elements for the program to store state information
 - A “next action” field for the debugger
 - Multiple round trips to the debugger following the initial execution of a new clause
 - The interactive trace response from the debugger tells the program what information to provide next
 - If the debugger decides to stop the program, there is no response until one is provided by the debugger window

The New Rexx Debugger



Required features – A UI library

- Doesn't need to be especially complex
- Multiple dialogs are used e.g. main dialog and watch window dialogs
- Basic dialog elements are sufficient, lists, buttons, entry fields
- Windows has ooDialog which has the above requirements
- There is no direct equivalent on other platforms so the initial releases were only on this platform

The New Rexx Debugger



Required features – Cross platform consideration

- After I posted details of my work a Linux version was requested
- There is no native ooRexx UI library for non-Windows platforms
- Lots of helpful discussions on the mailing list

- BSF4ooRexx provides full access to a Java JRE /JDK from ooRexx
- Java Swing /AWT libraries offer well tested basic UI elements in Java 8 and these are core components so always available
- Using Java extended support to Linux, Mac and maybe others

The New Rexx Debugger



Required features – Cross platform considerations

- The debugger comprises a core low-level debugger module `RexxDebugger.rex` (this also contains some cross-UI shared code) and separate UI modules
- The two UI modules are implemented as plugins in:
 - `RexxDebuggerWinUI.rex` for `ooDialog`
 - `RexxDebuggerBSFUI.rex` for `Java /BSF4ooRexx`
- Both modules implement the same UI functionality
- A thin (common) API is exposed for the core debugger to use
- One of the UI modules is loaded by the debugger at launch
- The Java UI can be used on Windows if desired

The New Rexx Debugger



Required features – Cross platform considerations

- On Windows, the debugger files can be anywhere in the path, running RexxDebugger.rex will launch the debugger
- On Unix platforms a shell script (“rexxdebugger” is provided to run the debugger with required environment options
- No “install.sh” but debugger files can be copied to e.g. /usr/local/bin and rexxdebugger made executable
- Deployment instructions are available in the README.md

The New Rexx Debugger



Direct launch - Tutorial Walkthrough

- Direct launch is running the program directly from the command line or in an application which hosts an embedded ooRexx interpreter to run the program
- The main dialog has source, stack and console views with console input and push buttons for control
- Program and trace output is not captured to the debugger console by default but debugger commands can be used to modify this behaviour
- The debugger pauses **after** executing an instruction as this is when Rexx requests trace input
- Breakpoints can be added and removed via double clicking and conditions set by right-clicking on them. Empty `/**/` comments in the code can be used to indicate that the debugger should insert a breakpoint when loading the program
- Rexx statements can be typed in the console prompt to view /modify program state



Direct launch - Tutorial Walkthrough (continued)

- The Watch button opens a variable listing aka “watch window”
- Watch windows update after running part of the program or executing a console statement
- Strings show on a single line by default, but double-clicking opens another watch window where the string can be viewed in multi-line and hexadecimal (byte) formats
- All collection objects (stems, arrays etc) in a watch window can be double clicked on to show their items in a new watch window. A (+) sign indicates an expandable collection
- In the top level watch window, .LOCAL and .ENVIRONMENT directories can be selected for viewing with a right-click menu action
- When in a ROUTINE, METHOD or procedure, watch windows will update to show the variables for that scope, greying out if the watch window is not valid for that scope
- For direct launch the program needs to include instructions / directives for activating interactive trace and loading the debugger, at the end of the tutorial.rex file in this instance.

The New Rexx Debugger



Launching from the debugger

- The debugger is run and loads the Rexx program itself
- The same UI is presented as for direct launch but with the addition of an Open button
- In this mode the default is for program output to be captured to the debugger console with trace output (apart from errors) silently dropped
- The program to run can be specified as a command line option e.g. `rexdebugger myprog.rex`, or loaded via the Open button
- Command line options are available to customize the debug session
- No modification of the program is required

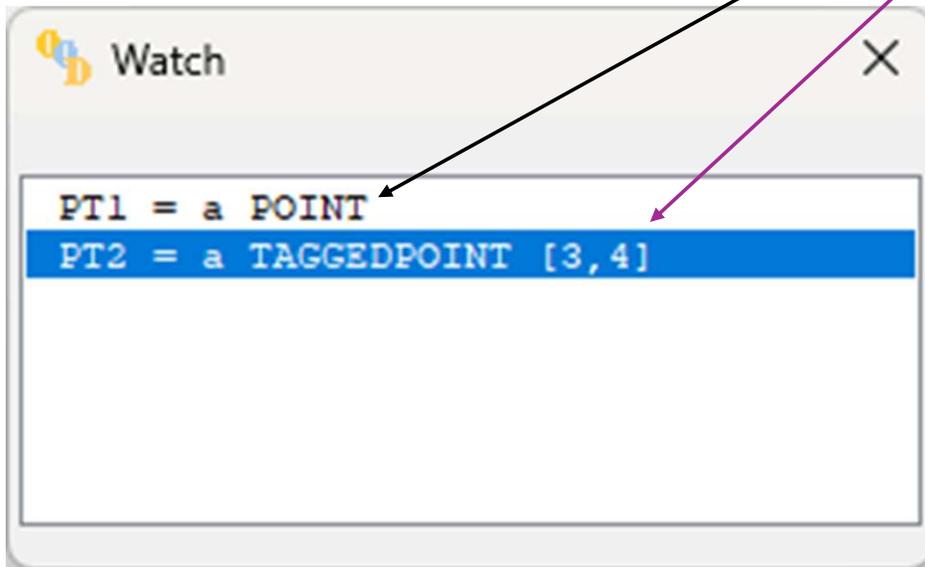
Sample TaggedClass.rex – extending a class to display object detail

- Object attributes for custom classes do not display in watch windows
- Implement an unguarded method “`makedebuggerstring`” to return display information
- The return value is displayed alongside the object type in watch windows
- Illustrated on the next slide

The New Rexx Debugger



TaggedClass.rex sample



```
NOP
pt1 = .point~new(1,2)
pt2 = .taggedpoint~new(3,4)
NOP
```

```
::class point
::attribute x
::attribute y
```

```
::method init
expose x y
use arg x,y
```

```
::class taggedpoint
::method init
expose x y
use arg x,y
```

```
::attribute x
::attribute y
```

```
::method makedebuggerstring unguarded
expose x y
return x','y
```

The New Rexx Debugger



Questions ?

The New Rexx Debugger



Links

ooRexxDebugger

<https://sourceforge.net/projects/oorexdebugger/>

ooRexx (4.2 or later required)

<https://sourceforge.net/projects/oorex/>

Java UI - BSF4ooRexx (641 or later required)

<https://sourceforge.net/projects/bsf4oorex/>

Java UI - Java 8 JRE (Minimum Java version tested)

<https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html>