Restructured Extended Executor

History Abridged

- Started by Mike Cowlishaw in 1979 at IBM England
- First demoed in 1981
- Shipped as an IBM mainframe product in 1982
- Ported to the PC in 1984/1985 (not by IBM)
- Native scripting language of OS/2
- Object Rexx source code released in 2004
- Rebranded as Open Object Rexx and released in 2005
- Considered by some to be a precursor to Python
- One of the older scripting languages – older than Perl, Python, and Ruby

Portability

- Runs on just about every platform
- To name a few: AIX, Linux, MS-DOS, all versions of Windows, Mac OS, MVS, VM, OS/400, OpenVMS, OS/2, NetWare, Amiga OS, Palm OS, and so on...
- Rexx uses decimal arithmetic – calculations are guaranteed across all architectures and OSes
- Rexx can invoke system commands which potentially hurt portability (cls vs. clear), but this can be avoided by using RexxUtil in many cases

Implementations

- Many free Rexx interpreters exist
- Regina and Open Object Rexx are most popular
- Others include Reginald, r4, roo!, NetRexx (Java)
- All conform to ANSI 1996 Rexx standard except NetRexx

The Language

- Only one data type (character string)
- Case insensitive
- Variables are automatically created as used
- Only 23 instructions
- Simple I/O
- Text oriented with excellent string manipulation, not meant for intensive math operations
- Straight forward - no required semicolons, indentation, brackets, or other superfluous syntax
- No compiling or linking results in faster coding and debugging
- Optionally object oriented with some interpreters
- Interpreted – modern computers can execute over 1,000,000 Rexx clauses per second so it's not a big deal...
- A suitable substitute for shell scripts and most other scripting languages

Hello World!

- Hello World can be written with a single command

```
say 'Hello World'
```

- This is a complete Rexx program
- No importing or main functions required

Where's the Interpreter?

- Neither Regina nor Open Object Rexx come with built-in interpreters
- So what do you do?

Write Your Own!

- A basic interpreter can be implemented with only four lines of code

```
do forever
        parse pull line
        interpret line
end
```

- A robust interpreter can be made with only a few more lines of code (mainly error trapping)

Text Processing

- Rexx has excellent text processing facilities
- One of the most convenient ways is with `parse`
- Example Data: `data = 'Doe,John,M,2/10/65'`
- A typical comma delimited string representing last name, first name, gender, and birth date

```
parse value data with lastname ',' firstname
  ',' gender ',' birthdate
```

- Splits the data at the commas and stores it into variables
- This simple example can be expanded to read many records from a text file and process the data
- No ugly string manipulation or regular expression syntax to remember

Arrays

- Rexx's array implementation provides a very flexible data type that can be used to implement other data types (like hash tables)

```
array.0 = 'Hello'
array.99 = 'World'
say array.0 array.99 /* prints Hello World */
```

- Rexx arrays can be sparse and don't have to start at 0.

Creative Use of Arrays

```
classes.cs231.location = ‘1404 Siebel’
classes.cs231.credits = 3
classes.phys211.location = ‘141 Loomis’
classes.phys211.credits = 4
say ‘Enter course number’
pull course
say ‘Your class is in’ classes.course.location ‘and
  is worth’ classes.course.credits ‘credits.’
```

Output:

```
Enter course number
cs231
Your class is in 1404 Siebel and is worth 3 credits.
```

System Commands

- Anything that isn't a Rexx command is passed to the system

```
say 'Here''s some text.'
say 'Press enter to clear the screen.'
pull txt
'clear' /* Linux command to clear screen */
say 'The screen was cleared.'
```

File I/O

- Rexx greatly simplifies file I/O
- No explicit file opening or closing is required
- No dealing with file handles
- If you need tight file I/O it can be done
- Rexx supports character and line I/O

Printing a File to Screen

```
file = ‘data.txt’
do while lines(file) > 0
  say linein(file)
end
```

Numeric Operations

- Rexx isn't meant for number crunching
- It does provide extremely useful number processing capabilities
- Precision limited only by memory
- Default precision is 9 digits but can be changed

```
numeric digits 150 /* 150 digits of precision */
say 435/756
/* Output:
0.57539682539682539682539682539682539682539682
  5396825396825396825396825396825396825396825396825 3
  9682539682539682539682539682539682539682539682539 7
*/
```

Other Interfaces

- Rexx interfaces have been created for other popular libraries and data sources
- Rexx/SQL
- Rexx/Curses
- Rexx/gd
- Rexx/CURL
- Many others...

Further Reading

- *Rexx Programmers Reference* by Howard Fosdick (2005) – very up to date
- *The Rexx Language 2nd Edition* (TRL-2) by Mike Cowlishaw (1996) – hard to find
- The documentation provided with the interpreters is usually excellent and will describe nuances of the particular interpreter