But I don't use objects, or do I?

Using Open Object Rexx to solve Classic Rexx problems

An altogether too common statement:

- “these needs arise from *trying not to use the oo features of oorexx* since i'm creating a way for some users who know no program-ming language to use *the minimal features of rexx.”*
  - Recent comment on the REXXLA mailing list (emphasis added)

This frequently results in rejecting the easiest solution

- The discussion from the previous statement ended up as a discussion of whether interpret or value() provided the better solution.
  - did not meet *the minimal features of rexx* goal
  - ooRexx solution would have been much smaller and easier for the target users to understand

Goals of Object Rexx Features

- Features were added with an eye toward providing easier ways to solve problems that users frequently asked about.
- Mike Cowlishaw's “top ten” list.
- Object orientation in many cases was the solution, not the end goal of the design.

Typical Questions

- How do I pass/return a stem to/from a procedure
- How do I expose a variable without having to expose through all call levels
- How do I drop a sub-stem
- How do I copy a sub-stem
- How do I reuse more of my code
- How do I get stem.0 to be automatically set
- How do I implement callbacks within my program

A simple example

```
emp.i.name = “Rick McGuire”
emp.i.location = “Sandy Hook”
....
call print_employees
....
print_employees:  procedure expose emp. empcount

do i = 1 to empcount
  ....
end
```

Common problems with using the classic approach

- The “accidental simple variable” problem.
- Writing code to deal with multiple collections.
- The external function variable scope.
- The embedded “.” problem
- Some problem solutions require use of interpret or value().

But wait...

- Structured data...
- A series of functions that operate on that data....

SOUNDS LIKE AN OBJECT TO ME!

An ooRexx equivalent

```
::class employee public
::method name attribute
::method location attribute
::method print
   say self~string
::method string
   expose name, location
   return name “at” location
```

An ooRexx equivalent

```
employees = .array~new
....
employee = .employee~new
employee~name = “Rick”
employee~location = “Rick”
employees[i] = employee
....
do employee over employees
    employee~print
end
```

Key differences

- Separation of the “object” from the “collection of objects”
- Not dependent upon exposing callers variables through multiple levels of call.
- Code is easily reused in other programs.
- Immune to the “constant tail element” problem.
- Error reporting for mistyped names.
- No interpret or value() required.

Building beyond stems and strings

Adding more structure to your programs:

```
::method init
expose managed
managed = .set~new

::method addManaged
expose managed
use arg employee
managed~put(employee)

::method getManaged
expose managed
return managed
```

All we are saying, is give peace a chance...

- Allow the ooRexx language to help you with what you're already trying to do!
- Using ooRexx features doesn't require a complete reshaping of your mind set...immediately rejecting these features frequently means you're working too hard!