History

- IBM developed Object Rexx (ORexx) for the OS/2 platform
- IBM transfers ownership to RexxLA as open source
  - Change name to Open Object Rexx (ooRexx)
  - License allows users maximum freedom, Common Public License
  - Did not transfer OS/2 source code nor Windows Developer Edition
- Available on several platforms: Windows, Linux, AIX, Solaris
What is Object REXX

- A programming language
- A scripting language
- An object oriented programming language
- Design principles same as for REXX
- REXX code will run under Object REXX
- Object REXX offers full object oriented capabilities:
  - Objects, classes, and methods
  - Inheritance
How Customers Use Object REXX

- To teach the fundamentals of procedural and object-oriented programming at a university
- To integrate applications
- To create utilities to support the Windows environment and infrastructure
Support for ooRexx

- **SourceForge** – site for many open source projects
  - [http://sourceforge.net/projects/oorexx/](http://sourceforge.net/projects/oorexx/)
  - Report bugs
  - Request features
  - Request support

- **ooRexx mailing lists**
  - [http://sourceforge.net/mail/?group_id=119701](http://sourceforge.net/mail/?group_id=119701)
  - Ask questions
  - General discussion

- **RexxLA mailing list** – must be a member to subscribe
  - [http://www.rexxl.a.org](http://www.rexxl.a.org)

- **Newsgroup for general REXX information**
  - comp.lang.rexx
Where to Get ooRexx

http://www.rexxla.org
User group for REXX, Object REXX, and ooRexx
Annual 4-day symposium:
Apr 30-May 3 Tampa Bay

http://www.oorexx.org
RexxLA’s site dedicated to Open Object Rexx
Main attractions ooRexx

- **Object oriented programming**
  - Object oriented extensions added to the language
  - Traditional Rexx still works
    - Percent = 100*right/questions  --express as a percent
  - Added classes, objects, methods
  - Added messaging, polymorphism
  - Added inheritance and multiple inheritance
  - Supplies user with base set of classes
Main attractions ooRexx

- An English like language
- Cross platform versatility
  - Windows (95, 98, NT, 2000, XP)
  - Linux (Intel and S/390)
  - AIX
- Fewer rules
  - No column rules, no line numbering, no semi-colons
  - Line spanning
  - No upper/lower case problems.
Main attractions ooRexx

- **Interpreted, not compiled**
  - Faster and easier to write code
  - No need to compile, link, bind
  - Lack code path checks
  - Open source code

- **Built-in functions and methods**

- **Type-less variables**
Main attractions ooRexx

- **String handling**
  - Traditional Rexx variables are String class
  - Flexible read, write, manipulate, compare
  - Perform arithmetic on String data

- **Clear messages**

- **Trace debugging**
Main attractions ooRexx

- **Development tools**
  - Windows Rexx API
  - Interact with C/C++, Cobol, OLE/ActiveX
  - Mathematical function package
  - File encryption package for Windows 2000 file system
  - Java interaction package available

- **Use Rexx as a Scripting language with the operating system**
Continue to use traditional Rexx code

- Almost all existing Rexx code will run
- Allows you to run both object oriented and procedure code at the same time

\[
s = \text{substr}(\text{name}, 2, 3) \quad /*\text{builtin procedure call}\*/ \\
s = \text{name}~\text{substr}(2, 3) \quad /*\text{oo String method substr}\*/
\]

- ooRexx adds reference objects to new types (arrays, queues, streams, etc).

\[
\text{my} = .\text{array}~\text{new}(5) \quad /*\text{create 5 element array}\*/
\]
Quick tour of traditional Rexx

/* REXX - Greeting.rex */
say `Please enter your name`
pull name
say `Hello’ name
EXIT 0

C:\> rexx greetings.rex
Quick tour of traditional Rexx

/* REXX - call1.rex */
i=10
call MyRoutine  /* call a routine */
say I
say plus10(i)  /* call as a function, returns value */
EXIT
/*------*/
myroutine:
i=i+10
return
/*------*/
Plus10:
return(i+10)
Quick tour of traditional Rexx

/* REXX - coin.rex */
count.1=0; count.2=0
do i=1 to 100
   call CoinToss
   say 'Flip =' result 'total H='||count.1 'T='||count.2
end/*i*/
EXIT 0

/*-------*/
CoinToss: procedure expose count.
i = random(1,2)
count.i = count.i + 1
if i=1 then return 'Heads'
return 'Tails'
The Object oriented world

- **Procedural code**
  - code: routines or procedures, are the primary focus
  - data: is passed to routines to act upon
  - code and data often independent of each other, code tends to be general purpose

- **Object oriented code**
  - data: (objects) are the primary focus
  - code: (methods) passed messages to execute
  - code and data are more closely tied together
The Object oriented world

- Objects receive messages to perform action, called methods
  - object~method(parameters)
  - “!iH”~reverse returns “Hi!”
  - Tradition Rexx built-in function reverse(“!iH”)

- Polymorphism. Same method in different classes, each with it’s own meaning.
  - pen~reverse, ball~reverse, “!iH”~reverse
The Object oriented world

- Classes are templates defining methods and variables.

- An object created from a class is called an instance of a class.

- ooRexx supports super-classes and sub-classes.
  - Create a BankAccount class.
  - Sub-class SavingsAccount and CheckingAccount.
Sub-class

- Scientific classification of organisms: Kingdom, Phylum, Class, Order, Family, Genus, Species
- Phylum subclass Kingdom, Class subclass Phylum, etc…
- Each level down inherits all the characteristics of the higher levels
Sub-class

/* rexx - acct.rex */
sav = .savings~new /*oo: create a new instance */
say sav~type /* prints A savings account */
sav~name = 'George'
exit
::class Account /*oo: start of class definition*/
::method type
   return 'An account'
::method 'NAME=`
   expose name
   use arg name
::class Savings subclass Account
::method type
   return 'A savings account'
ooRexx basic classes

- Alarm
- Message
- Method
- Monitor
- Stem
- Stream
- String
- Supplier
- Object (root of hierarchy)

Collection classes
- Array
- List
- Queue
- Table
- Set
- Directory
- Relation
- bag
### ooRexx supplied classes

**OBJECT class methods**

- new
- =
- ==
- <=
- <>
- <==
- class
- copy
- defaultname
- hasmethod
- init
- objectname
- objectname=
- request
- run
- setmethod
- start
- string
- unsetmethod

**ALARM class methods**

- cancel
- init
ooRexx supplied classes

CLASS class methods

  baseclass  defaultclass  define  delete
  enhanced   id           inherit  init
  metaclass  method       methods  mixinclass
  new        querymixinclass  subclass
  subclasses  superclasses  uninherit
ooRexx supplied classes

ARRAY class methods

new of [ ] [ ]= at
dimension first hasindex items last
makearray next previous put remove
section size supplier

LIST class methods

new [ ] [ ]= at first
firstitem hasindex insert items last
lastitem makearray next previous put
remove sectionsupplier
ooRexx supplied classes

QUEUE class methods

[ ]   [ ]=   at   hasindex   items
makearray   peek   pull   put
queue   remove   supplier

TABLE class methods

[ ]   [ ]=   at   difference   hasindex
intersection items   makearray   put
remove   subset   supplier
union   xor
ooRexx supplied classes

DIRECTORY
RELATION
MESSAGE
METHOD
SET
BAG
MONITOR
MUTABLEBUFFER
STEM
STREAM
SUPPLIER
ooRexx supplied classes

STRING class methods

new """" (abbuttal) " """ (blank)

comparison: = \= < < > > >= \> < <= \< == \== >> \>> >>= << \<< <<= ...

>> \>> >>= << \<< <<= concatenation: ||

arithmetic: + - * / % // ** logical: & && | \n
abbrev abs bitand bitor bitxor

b2x center changestr compare

copies countstr c2d c2x datatype
ooRexx supplied classes

STRING class methods

delstr  delword  d2x  format
insert  left     lastpos  length
makestring max  min  overlay  pos
reverse  right  sign  space  string
strip  substr  subword  translate
truncverify  word  wordindex
wordlength  wordpos  words  x2b  x2c  x2d
ooRexx supplied classes (Windows only)

- OODialog
- WindowsProgramManager
- WindowsRegistry
- WindowsEventLog
- WindowsManager
- WindowsObject
- MenuObject
- WindowsClipboard
- OLEObject
Creating Rexx classes

- Rexx directives
  - ::CLASS
  - ::METHOD
  - ::ROUTINE
  - ::REQUIRES

- Place directives after program code.
- Encountering a directive implies the end to code or prior directives.
::class

- ::CLASS classname
  [METACLASS metaclass]
  [SUBCLASS object|class]
  [MIXINCLASS mclass]
  [PUBLIC]
  [INHERIT iclass]
  ;
::method

- ::METHOD methodname

  [CLASS]
  [ATTRIBUTE]
  [PRIVATE]
  [GUARDED| UNGUARDED]
  [PROTECTED]
::routine

- ::ROUTINE routinename [PUBLIC]
::requires

- ::REQUIRES ‘programname’
Sample program

/* rexx - dino.rex */
dino = .dinosaur~new  /*create a instance*/
dino~diet  /*run the DIET method*/
::class dinosaur
::method init  /*constructor*/
    expose type
    say "Enter a type of dinosaur"
pull type
::method diet
    expose type
    select
    when type="T-REX" then string="Meat-eater"
    when type="RAPTOR" then string="Meat-eater"
    when type="BRONTOSAUR" then string="Plant-eater"
    otherwise string="Type of dinosaur or diet unknown"
end/*select*/
say string
Constructor/destructor

- Constructor, method “init”
- Destructor, method “uninit”

/* REXX - world.rex */
a = .world~new
DROP a
exit
::CLASS world
::method init
    say "constructor"
::method uninit
    say "destructor"
Special variables

- RC
- RESULT
- SIGL

- SELF – the instance of the class invoking the method
- SUPER – the super-class of the instance invoking the method
Types of classes

- **Object class**
  Creates instances and provides methods that these instances can use.

- **Abstract class**
  Defines methods its subclasses can inherit, but typically no instances. Used to organize classes in hierarchy.

- **Mixin class**
  Lets you add a set of instance and class methods to one or more other classes using inheritance.
Metaclasses

- ooRexx provides the CLASS class.
- It is a class factory for creating other classes.
/* rexx meta.rex */ /* output */
Created point instance ( 1 , 1 )
Created point instance ( 2 , 2 )
Created point instance ( 2 , 2 )
The point class has created 3 instances.*/

a = .point-new(1,1)  /*create point instances */
say "Created point instance" a
b = .point-new(2,2)
say "Created point instance" b
c = .point-new(2,2)
say "Created point instance" c

say "The point class has created" .point-instances "instances."  /* ask point class how many instances */

::class InstanceCounter subclass class
::method init
   expose InstanceCount
   InstanceCount = 0
   .message-new(self,.array-of("INIT",super),"a",arg(1,"A"))~send
::method new
   expose InstanceCount
   InstanceCount = InstanceCount + 1
   return .message-new(self,.array-of("NEW",super),"a",arg(1,"A"))~send
::method instances
   expose InstanceCount
   return InstanceCount
::class point public metaclass InstanceCounter
::method init
   expose xVal yVal
   USE ARG xVal,yVal
::method string
   expose xVal yVal
   return "(" xVal "," yVal ")"
Including classes defined in other files

/* REXX part.rex */
::class part public
::method init
  expose name description number
  use arg name, description, number
::method string
  expose name
  return "Part name:" name

/*REXX - usepart.rex */
a = .part~new("Widget", "A small widge", 12345)
b = .part~new("Framistat", "device to control frams", 899)
say a
say b
exit 0
::requires part.rex
Any procedural operation can be made OO

- **Procedural**
  - `say 3+7`
- **OO**
  - `say 3~’+’(7)`
Windows has powerful classes available

```rexx
/* REXX word.rex */
MyWord = .OLEObject~new("Word.Application")  --create object
MyWord~visible = .true
MyDocument = MyWord~documents~add    --get document object
MySelection = MyWord~selection       --get selection object
MySelection~TypeText("Text entered through ooRexx.")
MySelection~TypeParagraph
MyDay = date(weekday)
MySelection~TypeText("Today is " MyDay)

::REQUIRES "OREXXOLE.CLS"
```
Sample program

/* REXX stream.rex */
WordCounter = 0
InFile = .stream~new("STREAM.REX")
do until InFile~lines = 0
    InLine = InFile~linein
    NumberWords = InLine~words
    WordCounter = WordCounter + NumberWords
end

say "I counted the words of this file."
say "There are" WordCounter "words."
EXIT
Sample program

/* REXX array.rex */
MyArray = .array~new
MyArray[1,1] = "1.1"
MyArray[1,2] = "1.2"
MyArray[1,3] = "1.3"
MyArray~put("content 2.1",2,1)  --assign value
say MyArray[1,1] MyArray[2,1]
say "Num of items in MyArray" MyArray~items
EXIT
Sample program

/* REXX list.rex */
MyList = .list~of("first line", "second line", "third")
LastIndex = MyList~last  --get the last index
MyList~insert("fourth line", LastIndex)
do ListLines over MyList
   say ListLines
end
EXIT
Sample program

/* REXX calc.rex */
MyCalc = .calc~new
MySub = .calcSub~new
MySub~add
MySub~diff
EXIT      --exit not needed, but clear meaning
::CLASS calc
::METHOD add
    say "add called"
::METHOD diff
    say "subtract called"
::CLASS calcSub subclass calc
::METHOD mult
    say "multiply called"
::METHOD diff
    say "better subtract called"
Sample program

/* REXX poly.rex */
MyRect = .rectangle~new
MyTri  = .triangle~new
say "Area of my rectangle is" MyRect~area(4,3)
say "Area of my triangle is" MyTri~area(4,3)
exit
::CLASS rectangle
::METHOD area
  use arg width, height
  return(width*height)
::CLASS triangle
::METHOD area
  use arg base, height
  return(base*height/2)
Windows sample

/* REXX name.rex */

myDialog = .InputBox~new("Please enter your name:",
    "Input box sample")

name = myDialog~execute
say "Your name is" name
exit
exit
:::REQUIRES "OODIALOG.CLS"
Windows sample

/*REXX BB.rex */
myStem = .stem~new
myStem.1 = "New York Yankees"; myStem.2 = "Boston Red Sox"
myStem.3 = "Toronto Blue Jays"; myStem.4 = "Baltimore Orioles"
myStem.5 = "Tampa Bay Devil Rays"
myDialog = .SingleSelection~new("Who wins AL east?",,
                                  "Baseball poll", myStem.)
selected = myDialog~execute
if selected=0
   then say "You pressed cancel"
else do
   say "Your choice is" selected
   say "content" myStem.selected
end
::REQUIRES "oodialog.cls"
Windows sample

/* rexx multi.rex */

myStem = .stem~new

myStem.1 = "one"; myStem.4 = "four"
myStem.2 = "two"; myStem.5 = "five"
myStem.3 = "three"

myDialog = .multiListChoice~new("head line",,
                            "box title", myStem.)

selected = myDialog~execute

if selected=""
then say "you pressed cancel"
else say "you selected" selected

::requires oodialog.cls
/ * REXX IE.rex */

myIE = .OLEobject~new("InternetExplorer.Application")
myIE~width = 500; myIE~height = 700
say "dimensions set to" myIE~width "by" myIE~height
myIE~visible = .true
myIE~navigate("c:\www\baltimore.htm")
call SysSleep 5    /* wait 5 seconds */
myIE~quit
::requires "ORexxOLE.cls"
Windows sample

```html
<HTML>
<!-- wsh1.htm -->

<HEAD><TITLE>ooRexx as a Windows Scripting</TITLE>
</HEAD>

<BODY>

<H1>GLF text of body</H1>

<SCRIPT LANGUAGE="Object Rexx">

/*This is object Rexx */

/* 'document' implicite available object of MSIE*/

/* and 'writeln' is a method of this object*/

numeric digits 30 --rexx statement

document~writeln("<P>written with ooRexx... hello</P>")

document~writeln("<P>one divided by three is" 1/3 "</P>")

</SCRIPT>

</BODY>
</HTML>
```
Windows sample

```
<HTML><HEAD><TITLE>mixed sample</TITLE></HEAD><BODY>
<SCRIPT LANGUAGE="Object Rexx">
    document~writeln("<p>hello from object rexx</p>"

</SCRIPT>
<SCRIPT LANGUAGE="VBScript">
    document.writeln("<p>hello from VBScript</p>"

</SCRIPT>
<SCRIPT LANGUAGE="JScript">
    document.writeln("<p>Hello from JScript</p>"

</SCRIPT>
</BODY></HTML><!-- wsh3.htm -->
Closing

- **Product web site**
  - http://www.rexxla.org
    - Publications
    - Pre-requisites
    - Announcements
    - Support

- **Email**: George@Fulk.name