Open Object Rexx

Internet Utilities

W. David Ashley

Rexx Language Symposium

April 2008

What Are the Internet Classes?

The ooRexx Internet Classes are a collection of useful classes, methods and scripts to perform Internet related tasks. All of the classes are something you will need eventually or are something you will find useful in you own projects.

Internet Classes

- mime.cls
  - This class encapsulates mime types
- socket.cls
  - This class encalsulates the rxsock extension library
- streamsocket.cls
  - This class extends the socket.cls and subclasses the ooRexx standard InputOutput Stream class

Internet Classes cont.

- smtp.cls
  - SMTP mailer class. Uses the streamsocket.cls and mime cls
- memcache.cls
  - Encalsulates the Linux memcache library

Using the Mime Class

```
part1 = .mimepart~new()

part1~description = 'just some text'

part1~addContent('Part 1' || '0D0A'x)

part1~addContent('This is line 2' || '0D0A'x)

part1~addContent('This is line 3' || '0D0A'x)

part1~addContent('This is line 4' || '0D0A'x)

part2 = .mimepart~new()

part2~addContent('Part 2' || '0D0A'x)

multip =.mimemultipart~new()

multip~addPart(part1)

multip~addPart(part2)

say multip

return

::requires 'mime.cls'
```

Using the Socket Class

```
host = '127.0.0.1'

port = 8080

srvr = .server~new(host, port)

call syssleep(1)


call client host, port, 'This is test 1'

call client host, port, 'This is test 2'

call client host, port, 'stop'

return

::requires 'socket.cls'
```

Using the Socket Class (cont.)

```
::routine client

use strict arg host, port, message

s = .socket~new()

addr = .inetaddress~new(host, port)

retc = s~connect(addr)

if retc <> 0 then do

   say 'Error' s~errno() 'connecting to server socket.'

   return

   end

retc = s~send(message)

say s~recv(4096)

s~close()

return
```

Using the Socket Class (cont.)

```
::class server

::method init

use strict arg host, port

s = .socket~new()

if s = -1 then do

   say 'Error' s~errno() 'creating server socket'; return

   end

retc = s~setoption('SO_REUSEADDR', 1)

if retc = -1 then do

   say 'Error' s~errno() 'setting socket option'; return

   end
```

Using the Socket Class (cont.)

```
addr = .inetaddress~new(host, port)

retc = s~bind(addr)

if retc = -1 then do

   say 'Error' s~errno() 'binding socket'; return

   end

retc = s~listen(3)

if retc = -1 then do

   say 'Error' s~errno() 'making the socket a listening socket'

   return

   end

reply

stop = .false
```

Using the StreamSocket Class

```
host = '127.0.0.1'

port = 8080

srvr = .server~new(host, port)

call syssleep(1)


call client host, port, 'This is test 1'

call client host, port, 'This is test 2'

call client host, port, 'stop'

return



::requires 'streamsocket.cls'
```

Using the StreamSocket Class (cont.)

```
::routine client

use strict arg host, port, message

s = .streamsocket~new(host, port)

retc = s~open()

if retc <> 'READY:'then do

   say 'Error' retc 'connecting to server stream.'

   return

   end

retc = s~lineout(message)

say s~linein()

s~close()

return
```

Using the StreamSocket Class (cont.)

```
::class server

::method init

use strict arg host, port

s = .socket~new()

if s = -1 then do

   say 'Error' s~errno() 'creating server socket'

   return

   end

retc = s~setoption('SO_REUSEADDR', 1)

if retc = -1 then do

   say 'Error' s~errno() 'setting socket option'

   return

   end
```

Using the StreamSocket Class (cont.)

```
addr = .inetaddress~new(host, port)

retc = s~bind(addr)

if retc = -1 then do

   say 'Error' s~errno() 'binding socket'

   return

   end

retc = s~listen(3)

if retc = -1 then do

   say 'Error' s~errno() 'making the socket a listening socket'

   return

   end

say 'Server starting'

reply

stop = .false
```

Using the StreamSocket Class (cont.)

```
do while \stop
   cs = s~accept()
   if cs = .nil then do
      say 'Error accepting new socket'
      iterate
      end
   css = .StreamSocket~new(cs)
   cmd = css~linein()
   css~lineout(cmd)
   css~close()
   if cmd~upper() = 'STOP' then stop = .true
   end
s~close()
return
```

Using the Smtp Class

```
mime1 = .mimepart~new

mime1~addContent('This is a test.' || '0D0A'x)

msg = .smtpmsg~new

msg~From = 'dashley@holmes4.com'

msg~addRecipient('dashley@us.ibm.com')

msg~Subject = 'Test SMTP Msg From ooRexx'

msg~Content = mime1

smtpconx = .smtp~new

retc = smtpconx~connect('holmes4.com', 'dashley@holmes4.com', 'xxx')

retc = smtpconx~send(msg)

retc = smtpconx~logoff

return

::requires 'smtp.cls'
```

Using the Smtp Class – Example 2

```
mime1 = .mimepart~new

mime1~addContent('This is a test.' || '0D0A'x)

mime2 = .mimepart~new

mime2~addContent('Another test.' || '0D0A'x)

mimemp = .mimemultipart~new

mimemp~addPart(mime1)

mimemp~addPart(mime2)

msg = .smtpmsg~new

msg~From = 'dashley@holmes4.com'

msg~addRecipient('dashley@us.ibm.com')

msg~Subject = 'Test SMTP Msg From ooRexx'

msg~Content = mimemp
```

Using the Smtp Class – Example 2 (cont.)

```
smtpconx = .smtp~new

retc = smtpconx~connect('holmes4.com', 'dashley@holmes4.com', 'xxx')

if retc = -1 then return

retc = smtpconx~send(msg)

if retc = -1 then return

retc = smtpconx~logoff

return

::requires 'smtp.cls'
```