

RexxLA Symposium 2018

Redirecting I/O for Commands to an External Environment

Overview

- Definitions - What is I/O Redirection?
- The Development Journey
- The Package Implementation
- Summary

Definitions

- What are we talking about?
 - At a command prompt, you can e.g. send the output of a command to a file rather than the screen
 - In Windows you might issue:
 - `Dir *.rex > rexfiles.txt`
 - You can also redirect the input to a command:
 - `More < rexfiles.txt`
 - Error output can also be redirected
 - And the option to append to a file is available

Definitions(2)

- Can't we do that now?
 - Yes! Rexx will send any clauses that are just expressions to an External Environment as a character string (c.f. TRL2, Section 6)
 - So you can issue a command with I/O redirection from Rexx and then read the resulting file to process the command output
 - There is also the RXQUEUE filter that can make this easier
 - But it doesn't look much like Rexx!

Definitions(3)

- The ANSI Standard addresses this issue
 - Additional sub-keywords are added to the ADDRESS instruction to allow for I/O redirection in a more readable syntax; also allows for redirection to/from compound variables (stems) as well as files
 - This obviously makes processing a command's input/output much more convenient
 - REGINA has implemented this capability but ooRexx has not

Definitions(4)

- Overview of the syntax
 - The sub-keyword WITH is added following the command string (if present) and one or more “connection” definitions follow it
 - A “connection” defines one of the three I/O streams and specifies its redirection
 - Following the name of the stream – INPUT, OUTPUT or ERROR – is another sub-keyword that defines the type of the redirection target: STEM or STREAM

Definitions(5)

- Overview of the syntax (cont.)
 - For the INPUT stream, the last word is the “name” of the stream or stem
 - For the OUTPUT and ERROR streams, the “name” can be preceded by another sub-keyword, either REPLACE or APPEND
 - Whew!
- An example might help:
 - address cmd "curl -s" url with output stem s.

Development

- Goal
 - Eventually: develop and test the code needed to make this functionality available in ooRexx; i.e. implement RFE 4
 - Initially: to understand the mechanism used to execute external commands and then to determine how their I/O might be redirected

Development(2)

- Requirements
 - Become comfortable developing code in C++, specifically as used in the ooRexx interpreter
 - Be able to build a version of the ooRexx interpreter so that modifications can be tested

Development(3)

- Process

- Tried looking at the source code for the interpreter to see how the ADDRESS keyword instruction is implemented
 - Little success – there is no “roadmap” for the structure of the code; a high-level design document really should be written
 - Stumbled on a Windows API called CreateProcess which looked promising
- Read the MS documentation on CreateProcess
 - Found a link “Creating a Child Process with Redirected Input and Output”

Development(4)

- Process (cont.)
 - That link had a code example showing how to do the I/O redirection!
 - Began to design a “proof of concept” code implementation that would incorporate the technique shown in the example
- Structure of the design
 - The design should allow any combination of the three streams to be redirected (or none of them)

Development(5)

- Structure of the design (cont.)
 - The design should allow either stems or arrays to be specified as the “target” of the redirection
 - Processing was divided into 1) an ooRexx program that handled the input arguments and 2) a native (C++) routine that implemented the CreateProcess API invocation

Development(6)

- Structure of the design (cont.)
 - The interface to the ooRexx program consisted of up to 5 arguments: the environment name, the command string to be executed, and (optionally) the objects to be the “target” of the redirection in the order Input, Output and Error
 - The interface to the native routine consisted of exactly 3 arguments: the environment name, the command string to be executed, and an ooRexx directory
 - This avoided learning how to do “optional” args!

Development(7)

- Design Rationale
 - As the C++ code would be much more complex than anything I had previously written, keeping it limited to just what had to be done to access the Windows APIs seemed prudent
 - The Rexx program would handle the processing of the arguments and transforming the stream data from/to a common format – arrays
 - The ooRexx C++ APIs have good support for both arrays and directories

Implementation

- Develop the Rexx program first
 - Write a “stub” in Rexx to stand in for the C++ routine to be added later
 - Allow the first arg, the environment name, to be omitted and default to the value returned by the address() BIF
 - In Windows, this will be CMD on my system as I have no other environments defined
 - The second argument is simply a string to be passed to the CreateProcess API

Implementation(2)

- Develop the Rexx program first (cont)
 - The third, fourth and fifth arguments are optional and are the objects that are the targets for the redirection of the Input, Output and Error streams respectively
 - If the Input object is a stem, create an array and put the stem items in it in order
 - If the Output or Error object is specified, create an empty array to hold the resulting data

Implementation(3)

- Develop the Rexx program first (cont)
 - Create an empty directory and add entries named INPUT, OUTPUT and/or ERROR with the associated arrays if the corresponding argument was specified
 - Call the C++ routine (or stub) passing the three arguments
 - Process the Output and/or Error array data, converting it, if necessary, to the stem object(s) that were specified

Implementation(4)

- Develop the Rexx program first (cont)
 - Write a test program to run the package with various combinations of arguments
- Develop the native (C++) routine next
 - Use the same approach that I use when writing Rexx programs: a small bit at a time
 - Make use of the “iostream” class and the “cout” object to do the equivalent of Say in Rexx

Implementation(5)

- Main parts of the native routine
 - Determine which of the streams, if any, are to be redirected
 - Create the “pipes” that will connect to the new process
 - Create the new process that will execute the command
 - If the input stream is redirected, get the data from the Rexx array and write it to the pipe
 - Wait for the new process to complete

Implementation(6)

- Main parts of the native routine (cont)
 - If the output and/or error stream is redirected, read the data from the pipe(s) and put it into the Rexx array(s)
 - Make liberal use of “cout” statements to show what the routine is doing!
 - Make use of previously developed tools to make the “code-build-test” cycle easier and faster
 - Got it to run correctly without having to learn the C++ debugger!

Implementation(7)

- Review the ANSI standard
 - Ensure nothing I had done conflicted with what was specified
 - Realized I hadn't allowed for streams
 - Easy to add by using ArrayIn and ArrayOut
 - Only need to change the Rexx program
 - Decide to also allow the syntax in the standard that specified the “type” and “replace/append”
 - Argument(s) now became strings as opposed to object references

Implementation(8)

- Review the ANSI standard (cont)
 - Handling STREAM [REPLACE|APPEND] name wasn't too hard
 - Create a Stream object from the name
 - If Replace was specified (or defaulted to), send it the message `~~open(write replace)~close`
 - Handling STEM was more difficult
 - Needed to get a reference to the stem object from the name
 - The `GetContextVariable(name)` method will do that if name is a stem

Implementation(9)

- Review the ANSI standard (cont)
 - BUT the variable must be in the caller's scope
 - If I added a native routine to do this, it would have to be called directly from the invoking program, not from the Rexx program I had already written
 - Major redesign was required :-(
 - Divided the Rexx program into two Rexx routines: CheckArgs and RunCommand
 - Wrote another native (C++) routine that would be called in place of the original Rexx program

Implementation(10)

- Review the ANSI standard (cont)
 - Figure out how to do “optional” args
 - Not as difficult as I had expected
 - Have the CheckArgs routine return a directory for each stream that had a stem “name” specified with the “name” and a flag for “replace/append”
 - If any directories were returned, use GetContextVariable(name) to convert name to an object reference and send the message ~empty to it if the “replace” flag was set

Implementation(11)

- Review the ANSI standard (cont)
 - Pass the references to the RunCommand routine which would do the remainder of the processing
 - Write a lot of additional tests to make sure the original functionality still worked and various combinations of the new arguments did too
 - Debug, tweak and optimize ad infinitum

Implementation(12)

- Final package structure
 - One Requires file, ADDRWITH.REQ
 - One public external routine, ADDRWITH
 - One private external routine, ADD_WITH
 - Four private (ooRexx) routines:
 - checkArgs, which uses
 - resolve
 - runCommand, which uses
 - repackage

Implementation(13)

- Example
 - Earlier example of the ANSI standard:
address cmd "curl -s" url with output stem s.
 - Using the ADDRWITH package:
call addrwith cmd, "curl -s" url, , stem s.
 - Or
call addrwith , "curl -s url, , s.

Summary

- Proof of Concept complete
 - Implements the redirection functionality specified in the ANSI standard
 - Extends that functionality to include using objects as the targets of the redirection
 - Does NOT implement the maintenance of the redirection state
 - Not meant as a substitute for RFE 4

Addendum

- Testing under 4.2.0 revealed issues
 - Method ObjectToString sent to an an Array returns “an Array” instead of the contents
 - <iostream> causes compile errors
 - Method GetContextVariable for a new stem doesn't set the variable in the caller
- Contact information for comments and questions:
 - gil.b@windstream.net