2013 RexxLA International Rexx Language Symposium Proceedings

René Vincent Jansen (ed.)

THE REXX LANGUAGE ASSOCIATION
RexxLA Symposium Proceedings Series
ISSN 1534-8954

Publication Data

A publication of **RexxLA Press**

The responsible publisher of this edition is identified as *IBizz IT Services and Consultancy*, Amsteldijk 14, 1074 HR Amsterdam, a registered company governed by the laws of the Kingdom of The Netherlands.

The RexxLA Symposium Series is registered under ISSN 1534-8954
The 2013 edition is registered under ISBN 978-94-037-3322-7

2024-01-07 First printing

Introduction

History of the International Rexx Language Symposium

In 1990, Cathie Dager of SLAC[1] convened the organizing committee for the first independent Rexx[2] Symposium for Developers and Users. SLAC continued to organize this annual event until the middle of the 1990's when the RexxLA took over that responsibility. Symposia have been held annually since 1990.

About RexxLA

During the 1993 Symposium in La Jolla, California, plans for a Rexx User Group materialized. The Rexx Language Association (RexxLA), as it was called, is an independent, non-profit organization dedicated to promoting the use and understanding of the Rexx programming language. RexxLA manages several open source implementations of Rexx.

The selection procedure

Presentation proposals are solicited yearly using a CFP[3] procedure, after which the RexxLA symposium comittee reviews them and votes which presentations are selected for the symposium. The presentations are peer reviewed before being presented. Presenters are not compensated for their presentations.

Location

The 2013 symposium was held in Raleigh-Durham area, North Carolina, USA from 5 May 2013 to 8 May 2013.

[1] Stanford Linear Accelerator Center, since 2008 SLAC National Accelerator Laboratory
[2] Cowlishaw, M. F., **The REXX Language** (second edition), ISBN 0-13-780651-5, Prentice-Hall, 1990.
[3] Call For Papers.

Contents

1

Embedding Assets in REXX code – Frank Clarke

Date and Time

6 May 2013, 13:30:00 CET

Presenter

Frank Clarke

Presenter Details

Retired IBM PL/I programmer learned CLIST in 1973 and Rexx in 1988. He has been building utility software for programmers since 1973.

Embedding Assets into REXX Code

Frank Clarke
2013

What?

- Panels and skeletons (among other things) are permanently welded to the source code
- ...so that all* the elements are in one place

(* some limitations apply)

Why?

- HUGE savings in I/O
- Ease of maintenance
- Simpler distribute/install

How?

The embedded elements are 'hidden' in a REXX comment at the back of the source:

```
/*     Embedded elements follow the source as a comment:
)))   PLIB  DISPL01    identifies this as a panel named 'DISPL01'
)ATTR
...
)))   SLIB  LOADJOB   identifies this as a skeleton named 'LOADJOB'
//&uid.A JOB (&zacctnum.),'DEFAULT JOBCARDS',
...
*/
```

Extracting the Element – pt 1
Transfer the data to the queue

```
do while sourceline(currln) <> "/*"

   text = sourceline(currln)        /* save with a short name     */

   if Left(text,3) = ")))" then do  /* package the queue          */
      . . .
      end                           /* package the queue          */

   else push text                   /* onto the top of the stack  */

   currln = currln - 1              /* previous line              */

end                                 /* while                      */
```

Extracting the Element – pt 2
If this is a new DDName…

```
 if Left(text,3) = ")))" then do  /* package the queue          */

    parse var text ")))" ddn mbr .   /* PLIB PANL001  maybe     */
    if Pos(ddn,ddnlist) = 0 then do  /* doesn't exist           */

       ddnlist = ddnlist ddn      /* keep track                 */
       $ddn = ddn || Random(999)
       $ddn.ddn = $ddn            /* PLIB247, maybe             */

       address TSO "ALLOC FI("$ddn")" fb80po.0

       "LMINIT DATAID(DAID) DDNAME("$ddn")"
       daid.ddn = daid

       end
    . . .
    end
```

Extracting the Element – pt 3
Create a new member

```
daid = daid.ddn

"LMOPEN DATAID("daid") OPTION(OUTPUT)"

do queued()

   parse pull line
   "LMPUT DATAID("daid") MODE(INVAR) DATALOC(LINE) DATALEN(80)"

end

"LMMADD DATAID("daid") MEMBER("mbr")"

"LMCLOSE DATAID("daid")"
```

Make the Libraries Active

```
dd = ""

do Words(ddnlist)                    /* each LIBDEF DD             */

   parse value ddnlist dd  with  dd ddnlist
   $ddn   = $ddn.dd                  /* PLIB322 <- PLIB            */
   "LIBDEF  ISP"dd "LIBRARY  ID("$ddn") STACK"

end

ddnlist = ddnlist dd
```

Other Examples

Many of the routines on my code examples web page: (home.roadrunner.com/~mvsrexx/REXX/) make use of this technique. Any of them displaying this warning:

```
|**-***-***-***-***-***-***-***-***-***-***-***-***-***-***-***-**|
|                                                                 |
|          WARNING: EMBEDDED COMPONENTS.                          |
|             See text following TOOLKIT_INIT                     |
|                                                                 |
|**-***-***-***-***-***-***-***-***-***-***-***-***-***-***-***-**|
```

use DEIMBED.

2

Defensive Programming with Rexx – Les Koehler

Date and Time

6 May 2013, 15:00:00 CET

Presenter

Les Koehler

Presenter Details

About the speaker: Les has been involved with Rexx since he received the initial distribution of Mike Cowlishaw's first Specification on 1 May 1979 at the IBM Research Triangle Park Lab just outside Raleigh NC, where they had a VM/370 mainframe and Les developed VM tools and applications.

24th International Rexx
Language Symposium
5-8 May 2013
Durham, NC USA
Sponsored by
Rexx Language Association

You're Not Paranoid If...
Defensive Programming In Rexx
A User Experience

Les Koehler
6 May 2013

Table of Contents

Abstract

I define "Defensive Programming" as the ability to preserve run time data so that problem determination in case of failure is straight forward.

Thus, I will present techniques I've learned to use in my code that make it easier to debug problems after the fact.

The environment and the problem

When I first started using The Hessling Editor (THE) with my Windows 2000 Gateway pc, I was puzzled by some behavior after I had made some (I thought) simple changes to its *profile.the*

First attempt - Using the MSG command

Initially, I used a *msg?* flag and the **msg** command:

if *msg?* then 'msg whatever'

However, it soon became apparent that what was really needed was a file that could be examined later.

Writing to a file

Here is the subroutine used to write to the file:

```
LOGIT: Procedure Expose sigl
  mysigl=sigl
  Parse Arg logargs
  If logargs='' Then logargs=Sourceline(mysigl+1)
  Parse Value Right(Space(Date(),0),9,0) Time('L') With ds ts
  logfile='C:\MyTHEstuff\msglog.log'
  .stream~new(logfile)~~lineout(ds ts logargs '@' mysigl)~close
  Return
```

An entry in the file would look like this:

```
19Mar2013 23:37:24.416000 -- Initial @ 53
```

when created by the following snippet of code:

```
If initial() Then Do
  If log? Then Call logit
-- Initial
```

An arbitrary string can also be passed to **LOGIT**:

```
  If log? Then Call logit 'PROFILE Starting.' ,
   Initial()='initial() 'ctr='ctr ,
   'fid='fid '@' thisline()
```

Here I was capturing the *initial()* flag that

indicates that this is the first execution of the **profile**, as well as an internal *ctr* variable to count the number of executions.

The same methodology was used in several place to record various information so I could find out what was wrong with the changes I had made.

Log Results

The file showed me:

```
PROFILE Starting. Initial()=1 ctr=1
USERPROF starting!: EDITV CTR= 1 Passed CTR= 1
          Passed initial?= 1 INITIAL()=1
PROFILE Starting. Initial()=1 ctr=2
USERPROF starting!: EDITV CTR= 2 Passed CTR= 2
          Passed initial?= 1 INITIAL()=1
USERPROF  ending!: EDITV CTR= 2 Passed CTR= 2
          Passed initial?= 1 INITIAL()=1
PROFILE Ending.  Initial()=1 ctr=2
USERPROF  ending!: EDITV CTR= 2 Passed CTR= 1
          Passed initial?= 1 INITIAL()=1
PROFILE Ending.  Initial()=1 ctr=1
```

clearly showing that *something* was causing the **profile** to recursively execute!

Zeroing In

I added some more calls to **LOGIT** which produced: