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:

```
PROFILE Starting. Initial()=1 ctr=1 fid=C:\DIR DIR
PROFILE Starting. Initial()=1 ctr=2 fid=C:\DIR DIR
PROFILE: reprof on, defsort set.
                Initial()=1 ctr=2 fid=C:\DIR DIR
PROFILE Ending.  Initial()=1 ctr=2 fid=C:\DIR DIR
```

which clearly showed that there was a tie-in between the **reprof on** and **defsort** commands.

The fix was astonishingly simple: change the order of the commands so that **defsort** executed before **reprof on**!

Saved Again

Recently I wanted to make some improvements to the **smart_enter** macro, so I edited **defkeys**, which only executes when THE initially starts, and commented out the definition of **smart_enter**:

```
/* Define my keys */
'linend on #'
'define C-PLUS REFRESH'
--'define ENTER macro smart_enter'
"define A-R macro ringlist"
"define C-R macro ringlist"
"define C-T hit ~"
"define A-K cmdline top"
```

and manually reset it with *define enter* and got some of the work done. I then hibernated the pc and resumed work the next day, manually

redefining the *ENTER* key as needed while I did various tests and made imrovements to **smart_enter**.

Finally satisfied with my testing, I removed the comment, saved the **defkeys** file, closed THE and restarted it.

To my total surprise, THE went into a loop, and didn't present me with the opening view of its directory. So I closed it and started the thumb drive version to examine **defkeys** and it seemed fine so I Quit the file.

Next, I opened **profile** turned on the *log?* flag, saved the file and restarted the disk version of THE. As expected, it looped so I closed it and used the thumb drive version to examine the log file where I found:

```
PROFILE Starting. Initial()=1 ctr=2
  fid=C:\REXX $$$ @ 73 @ 73
'EDITV GETF PROFILE' @ 83
not userprof @ 93
PROFILE continues... Initial()=1 ctr=2
    fid=C:\REXX $$$ @ 94 @ 94
  'set msgline on +3 * overlay' @ 102
--  Call setpfkeys '_default shn s' @ 105
    'macro defkeys' @ 108
```

so it looked like **defkeys** was the problem, but it doesn't run any other macros, it just has *define*

statements!

Very puzzled, I then used the *'Cut the problem in half'* approach. That's when I determined that making the first line an *exit* statement made no difference, so it wasn't the code that was causing the loop.

That led me to re-examine the log file, where the implications of:

```
fid=C:\REXX $$$ @ 73 @ 73
```

hit me: That's the name of of the trace file that THE produces when *trace* is turned on!

I began to suspect a corrupted **defkeys** file, so I switched to the DIR.DIR file and deleted **defkeys**. Then I saved and quit the copy I was editing, closed THE and restarted it... problem solved!

Progresss messages and logging

With a little planning you can integrate *progress messages* with the logging facility to get twice the payback, which is what I've done with all the code I use to process Membership Applications

and Symposium Registrations.

For example:

```
Call msg 'PayPalDate and Transaction ID on one line'



MSG: Procedure Expose sme logfile log? msg? me
  trace o
  If msg? Then 'command msg' me Arg(1)
  If log? Then Call log Arg(1)
  Return
```

Which might produce a log file like this:

```
SYMREG<<<<< 24Feb2013 04:43:11 Started with:
SYMREG      24Feb2013 04:43:11  = NOUPDATE TEST PROGRESS NOQUIET DETAILS
LOG NOHELP NO/? NO--HELP NO?
SYMREG      24Feb2013 04:43:11 File: C:\Symposium_Registration_2013\Symposium
Registration for Les Koehler II (vmrexx@womewhere.com).eml.txt
SYMREG      24Feb2013 04:43:11 Parsing Registration
SYMREG      24Feb2013 04:43:11 Checking flags
SYMREG      24Feb2013 04:43:11 error?=0. Checking variables
SYMREG      24Feb2013 04:43:11 TEST MODE! Here's what *would* have happened:
SYMREG      24Feb2013 04:43:11 Saving Properties for vmrexx@somewhere.com
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Name,Les Koehler II)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Addr,8450 Programmer Lane )
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
City,TAMPA )
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
State,FL)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Zip,33634)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Country,USA )
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Email,vmrexx@somewhere.com)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Phone,DoN-otC-all1)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Affiliation, )
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
```

```
Nickname,No)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
AmountDue,50)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Payby,Mail)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Sent,2/3/2013 9:31 PM)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Symp,Daily Sessions)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Days,Monday Wednesday)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
ApplID,E196700EF69068)
SYMREG      24Feb2013 04:43:11 props~setproperty(vmrexx@somewhere.com
Symposium registration added
SYMREG      24Feb2013 04:43:11 props~save(C:\Symposium_Registration_2013)
SYMREG      24Feb2013 04:43:32 vmrexx@somewhere.com saved in:
C:\Symposium_Registration_2013
SYMREG      24Feb2013 04:43:32 Symposium registration added
SYMREG>>>>> 24Feb2013 04:43:32 Finished with rc=0 at line 299
```

Quite obviously I've mixed the code from one program with the log file from another, but you get the idea.

Summary and Conclusion

Summary

You've seen:

- The environment and the problem.
- The first debugging attempt using the **msg** command.
- A simple subroutine, **LOGIT** for accumulating data to a file
- Some of the entries in the log file.
- How the details in the log file helped me find the problem.
- A recent experience where the log file helped me find a corrupted file.

- How progress messages and logging can be combined

Conclusion

It is well worth the minimum effort required to include a logging capability in your code. The benefits are:

- It makes analysis and debugging easier
- It provides a record of significant events