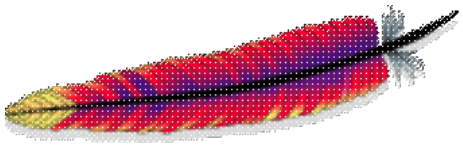

Writing CGI Scripts in REXX

By Steve Swift (aka "Swifty")

Tuesday, 19th May, 2009 13:30-14:30

http://www.swiftys.org.uk/symposium/P01-Introduction.html

<>

About:

Steve Swift (aka "Swifty")

This Session

Mr Coopers's Law:

If you do not understand a particular word in a piece of technical writing, ignore it.
The piece will make perfect sense without it.

http://www.swiftys.org.uk/wiz?263

<>

CGI Scripts.

What are they?

HTML:

http://www.swiftys.org.uk/hello.html

```
<H2>Hello <U>world</U>!</H2>
```

Result:

Hello world!

CGI:

http://www.swiftys.org.uk/cgi-bin/hello.rex

```
#!/usr/bin/rexx
Say 'Content-type: text/html'
Say
Say '<H2>Hello <U>world</U>!</H2>'
```

Result:

Hello world!

Simple Rexx tracing

test.rex

```
/* Simple test */
Trace ?r
C = 'L'
-- lots of stuff here
Say 'The time is' time(C)
```

Result:

```
  +++ "WindowsNT SUBROUTINE c:\Test.rex"
3 *-* C = 'L'
  >>>   "L"
+++ Interactive trace. "Trace Off" to end debug, ENTER to Continue. +++

5 *-* Say 'The time is' time(C)
  >>>   "L"
  >>>   "The time is 13:17:10.328000"
The time is 13:17:10.328000
```

Tracing CGI?

There is a problem:

CGI:	STDOUT Captured by webserver; sent to browser STDERR Captured by webserver; sent to error log

Solution #1:

Insert debugging into HTML

```
C = 'L'
-- lots of stuff here
Say 'C='c
Say 'The time is' time(C)
```

Result:

```
C=L The time is 13:43:28.484793
```

Problems:

1. The debugging affects the structure of the webpage
2. You might not find it easily in a complex page
3. If the data contains an "<" then it will be interpreted as HTML
4. Once you've found the problem you have to take the debugging out

Solution #2:

Defer the debugging to the end of the page

```
Debug.0 = 0
C = 'L'
-- lots of stuff here
Call Debug 'C='c'
Say 'The time is' time(C)

If debug.0 > 0 then do
  Say '<H2>Debug:</H2>
  Do I = 1 to debug.0
    Say debug.i'<BR>'
    End
  End
Exit

Debug:
Debug.0 = Debug.0 + 1; Debug.[debug.0] = arg(1)
Return
```

Result:

The time is 13:43:28.484793

Debug:

C=L

Problems:

1. The debugging is always on
2. If the data contains an "<" then it will be interpreted as HTML
3. Once you've found the problem you have to take the debugging out

Solution #3:

Control the debugging with a cookie

Steps:

1. How to toggle the cookie on and off
2. How to check the cookie in REXX

Toggling a cookie (Zero REXX interest)

In your page header:

```
<HEAD>
<SCRIPT SRC=/debug.js></SCRIPT>
</HEAD>
```

In your HTML directory: (/debug.js)

```
function getCookie(c_name)
{
if (document.cookie.length>0) {
  c_start=document.cookie.indexOf(c_name + "=")
  if (c_start!=-1) {
    c_start=c_start + c_name.length+1
    c_end=document.cookie.indexOf(";",c_start)
    if (c_end==-1) c_end=document.cookie.length
    return unescape(document.cookie.substring(c_start,c_end))
    }
  }
return ""
}

function setCookie(c_name,value,expiredays)
{
var exdate=new Date()
exdate.setDate(exdate.getDate()+expiredays)
document.cookie=c_name+ "=" +escape(value)+
((expiredays==null) ? "" : ";expires="+exdate.toGMTString())
}

function toggle_Debug()
{
Debug = getCookie('Debug');
if (Debug==1) {
  setCookie('Debug',0)
  alert('Debugging is now off')
  }
else {
  setCookie('Debug',1)
  alert('Debugging is now on')
  }
return false
```

```
    }
```

In your HTML:

```
    <A HREF="" onClick="return toggle_Debug()">Toggle Debug</A>
```

Checking a cookie in REXX

Functions:

```
::Routine Cookie public
Name = '; 'arg(1)'='    /* Cookie: */
Parse value '; 'value('HTTP_COOKIE',,'ENVIRONMENT')';' with (name) value ';'
Return value

::Routine Debugging public
Return cookie('Debug')=1
```

The effect on the debugging:

```
If debug.0 > 0 & 'debugging'() then do
  Say '<H2>Debug:</H2>
  Do I = 1 to debug.0
    Say debug.i'<BR>'
    End
  End
```

Handling & and < in the debug data

Function:

```
::Routine NoHTML public
Return changestr('<',changestr('&',arg(1),'&amp;'),'&lt;')
```

The effect on the debugging:

```
If debug.0 > 0 & 'debugging'() then do
  Say '<H2>Debug:</H2>
  Do I = 1 to debug.0
    Say 'nohtml'(debug.i)'<BR>'
    End
  End
```

Removing the debugging when you're done

There is now no need to remove the debugging.

Unless you click the "Toggle Debug" link, then it is invisible.

Questions and answers:

Q1.

Wouldn't it be better not to collect the debug information when debugging is off?

A1.

Probably not. If your code encounters a fatal error condition, it can set a flag which causes the debug information to come out anyway So when the user reports the problem, and sends you a screenshot, you will have a traceback of what happened.

Q2.

Is it possible to trace external functions/subroutines as well?

A2.

Yes. read on!

⇦

Tracing External routines

There are a couple of problems with the debug routines as developed:

1. They cannot be used to trace external functions and subroutines
2. You have to expose "debug." in every "Procedure" that contains debugging code, or a call to a routine which does.

The solution to this lies in using the "local environment object" (.local). If you are unfamiliar with this, then it can be seen as a way of creating variables which are available across all of the rexx routines that are running under the same invocation of rexx. This means all subroutines and functions called from your main program. If you execute external code by invoking a new copy of rexx then the .local object will not cross this boundary.

The following pages show the exact version of the debug routines that I'm currently using.

The current Debug routine

This code is in a file called "subroutines.rex" and is included from the main routine using ::Requires 'subroutines.rex'

```
      -- Initialisation code
      If .local~debug.state = .Nil then do                                          /* If we have not initialised debug     */
        Debug.0 = 0                                                                 /* Create the stem variable             */
        .local['DEBUG'] = debug.                                                    /* Create pointer to it in .local       */
        .local~debug.state = 0                                                      /* Turn debugging off by default        */
        End
      Exit

      ::Routine Debug public
      Parse arg text,line,email                                                     /* Debug: Save debug data               */
      If email<>'' then if .local~owner.email=.Nil then .local~owner.email=email    /* Email to use if we hit a fatal error */
      If text \== '' then do                                                        /* If a non null text is passed...      */
        D = .debug[0]+1                                                             /* Increment the line count             */
        If line <>  '' then .debug[D] = line text                                   /* Record source line and comment       */
        Else .debug[D] = text                                                       /* Record just the comment              */
        .debug[0]=D                                                                 /* Save new line count                  */
        End
      Else .local~debug.state = 1                                                   /* Null comment? Turn debugging on      */
      Return
```

≤

The current Debug_List routine

This code is in a file called "subroutines.rex" and is included from the main routine using ::Requires 'subroutines.rex'
It uses a couple of routines, "Systrace" and "Threads" which are not included here. "Systrace" just creates entries in a system-wide trace log. "Threads" works out the program being run under rexx in the parent thread. They are both highly specific to the system where the code is running.

```
::Routine Debug_List public
If .fatal.error = 1 then .local~debug.state = 1                                                                                                  /* Always deb
If .debug[0] = 0 | .local~debug.state \== 1 then return                                                                                          /* Debug_List
Arg parms                                                                                                                                        /* Get option
Parse arg ,efn                                                                                                                                   /* Get caller
If efn = '' then do
  Call SysTrace 'Something called Debug_List without argument 2 (efn)',word('env'('Remote_User') 'env'('Remote_Addr'),1)                        /* Blow whist
  Call 'Threads' 'Debug_List()'                                                                                                                  /* Try to wor
  End
HTML    = wordpos('HTML',parms) > 0                                                                                                              /* Do we want
Lines   = wordpos('LINES',parms) > 0                                                                                                             /* Do we want
If html then say copies('</TABLE>',6)'hr'()'<H2 STYLE="border-width:0;padding:0;margin:0 0 0 0">Debug:</H2><TABLE CELLSPACING=0>'
Else say 'Debug:'
Trace = (.fatal.error = 1) & \'swifty'()                                                                                                         /* SysTrace a
Do I = 1 to .debug[0]
  Select
    When html & lines then say '<TR VALIGN=TOP><TD ALIGN=RIGHT>'word(.debug[I],1)'<TD>'nohtml(subword(.debug[I],2))/* HTML output with line n
    When html then say '<TR VALIGN=TOP><TD>'nohtml(.debug[I])                                                                                    /* HTML output
    Otherwise say .debug[I]                                                                                                                      /* Plain text
    End
  If trace then call SysTrace .debug[I],efn
  End
If html then say '</TABLE>'
Return
```