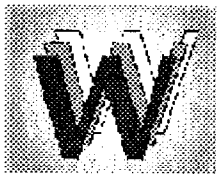
# Writing CGI Scripts for WWW Using Rexx

**Les Cottrell**
**Bebo White**
**Stanford Linear Accelerator Center**

**Pages 68-99**

# Writing CGI Scripts For



# Using


*Les Cottrell and Bebo White, SLAC*
*6th International REXX Symposium*
*May 2, 1995*

Start

# Writing WWW CGI Scripts in REXX

This presentation may be found at:

`http://www.slac.stanford.edu/~bebo/rexx/title.html`

---

Next

# What is WWW?

## The largest service on the Internet

- The Internet is like the road system
- WWW is like the parcel delivery service

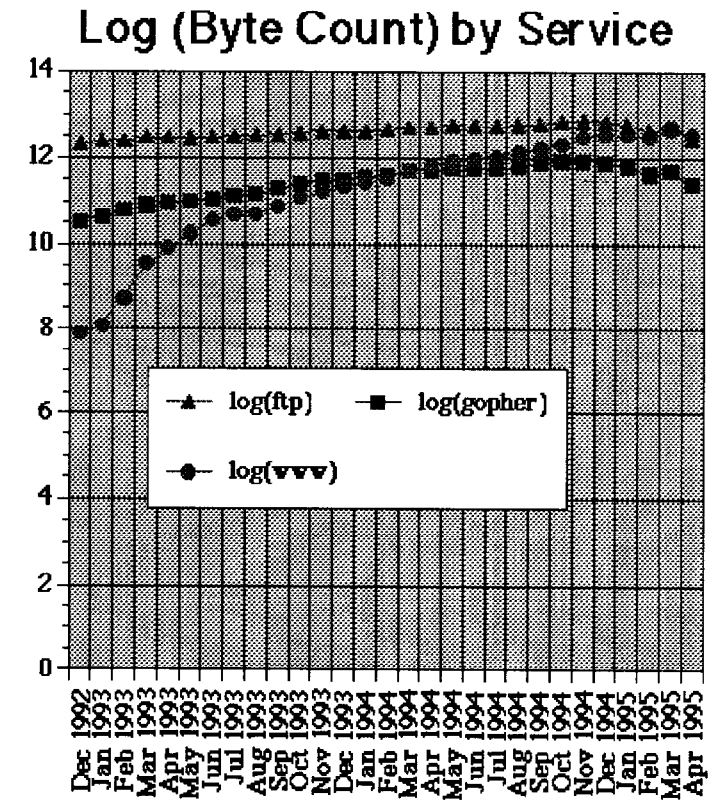
---

- 27,000 current WWW sites
- Number now doubling every 53 days
- 5 million documents stored in WWW sites

Source:Quoted in *BusinessWeek*, February 27, 1995

---

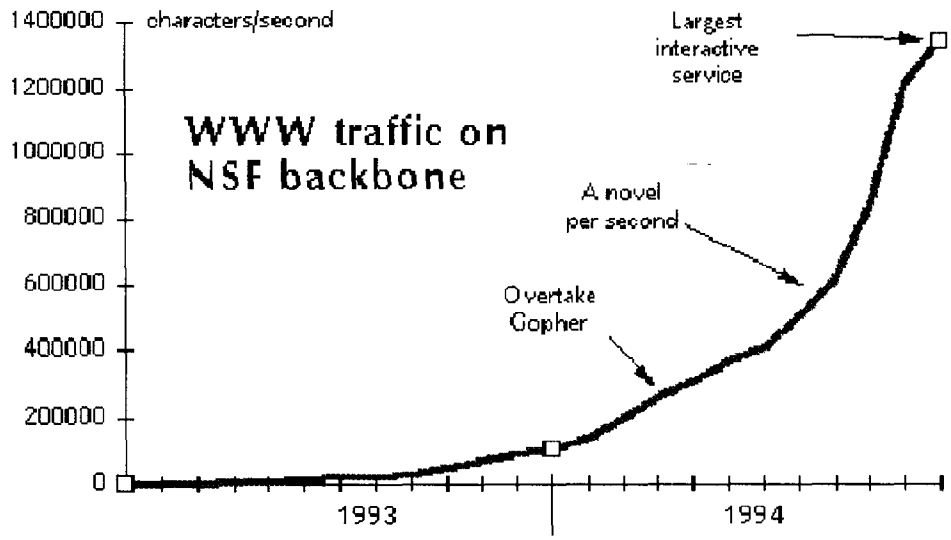
[Next]




**Statistics provided by Merit NIC Services machine**
**Graph by: James E. Pitkow, pitkow@cc.gatech.edu**

---

|Next|



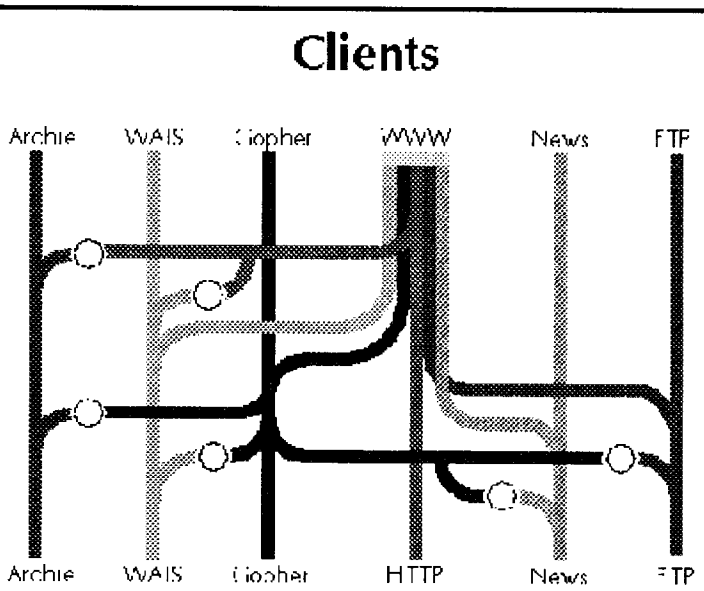

"Over the past three years the traffic on the NSF backbone has increased from 1 TB per month to 18 TB, with a good portion attributable to WWW services." - *Vinton Cerf*

---

|Next|

# Integration of earlier Internet systems

- Gopher, NEWS, Archie, WAIS, ftp, ... are all seamlessly available




[Next]

# Format independent

- HTML is *not a format* but a way of structuring documents
- Formatted documents are available through their native applications

---

# Multiple Media

- Images
- Sound
- Movies
- Launching of sessions (video, telnet, conferencing, ...

---

[Next]

# Some Neglected Uses of WWW

## By the provider:

- CWIS
- Search engines
- Data Base interfaces
- Collaborative work
- Special formats

## By the user:

- Home pages
- Structuring of own local information
- Participation (delurking?)

---

|Next|

# What is CGI?

The Common Gateway Interface, or CGI, is a standard for external gateway programs to interface with information servers such as HTTP servers.

Gateways are programs (called CGI scripts) that serve data which is not directly readable by a client program, such as a database of high energy physics preprints or personnel information, and convert it to an acceptable form, such as an HTML page, a PostScript file, some images, or a combination of these.

Familiar examples of Web applications which use CGI scripts are fill-out forms, interactive graphics (e.g., banners, maps, and menubars, etc.) and search engines (e.g., Yahoo, Lycos).

---

[Next]

# How Do Gateways Operate?

Gateways can be run by themselves, but are designed to be run by the HTTP server. The server sets up an environment which is

- secure (to prevent willful damage by users)
- informative (to allow communication between httpd server and gateway)
- contained - so all output is sent by the server back to the client

An example of a database query

Another example of a database query

Communication between client, server, and CGI gateway

This is a searchable index. Enter search keywords:

# SLAC SPIRES: HEP Preprint database search

Send corrections to: LIBRARY@SLAC.STANFORD.EDU . Use QSPIRES search language (see examples below). Note that there is no possibility for iterative search (yet) in WWW. Therefore, when needed, combine several criteria in a single request. Need more help ?
Examples:

```
show indexes
find author perl, m and title tau and date before 1980
find title prefix supercollid and date 1994
find t so2n+1                                [finds title SO(2n+1)!]
find bulletin-bd  hepex and date-added 9/94
find cn mark-iii and date after march 1991
browse coden physics letters
find c phlta, 70b, 487              [finds citations of a paper!]
find a abe and date 1988 (using wwwcite          [shows citations!]
find author gross, david and journal phys rev
browse affiliation caltech
find af cal tech and date 1994 (result
browse topic higgs
find topic higgs boson or title higgs and date 1-95 (using wwwbrief
browse last ppf
find ppf 9442 (seq rs
```

To learn more about authors, institutions, or acronyms, try **WHOIS, WHEREIS, or WHATIS:**

```
whois ginsparg
whereis cern
whatis sld
```

See also other SPIRES databases, or SPIRES News, or the SLAC home page.

*19 April 1995*

# SLAC Phone Directory: Search Form

*SLAC 13 Apr 1995*
This fill-out form can be used to search the SLAC phone directory (previously called BINLIST). Fill in the entries you know, leave others blank. Family name, First name and E-mail ID will support truncated searches, using an asterisk (eg: john* ).

```
    Family name:
     First name:

       E-mail ID:

Work Extension:
```

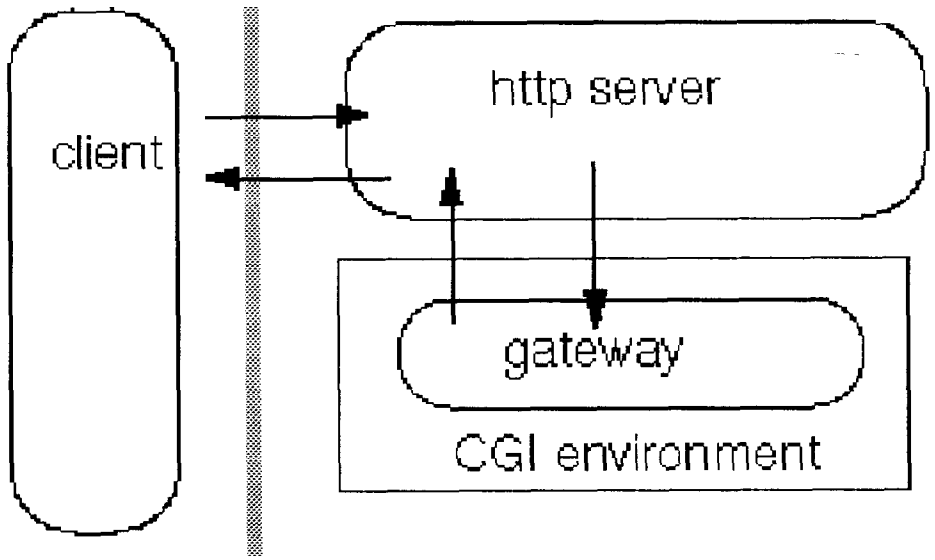
**HELP**

---

The original of this form was created by Evelyn Aviles-Hernandez.

*Diana Gregory*

# Communication Between Client, Server, and CGI Script




---

Next

# Design of a CGI Script

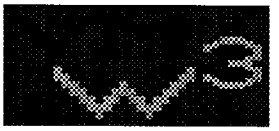
1. Design the CGI script interface:
   - Use of the <ISINDEX> or <ISMAP> in an HTML page;
   - Design of a fill-out form using HTML;
   - Specify the format of a CGI script-compliant URL which can be used as a link in an HTML page.

2. Design and code a CGI script which performs the following:
   - Argument decoding;
   - Argument validation;
   - Processing (including error handling);
   - Output and cleanup;

---

INext|

# CGI Scripts In REXX

- Webshare - the VM HTTP server 
  - Written entirely in REXX;
  - Supports CGI scripting in REXX;
  - Information at

    `http://ualvm.ua.edu/~troth/software/cmshttpd.html`

- Today's discussion is limited to use with Unix-based servers;

---

|Next|

# Providing Input to a CGI Script

- The `QUERY_STRING` environmental variable
  - `QUERY_STRING` is defined as anything following the first "?" in the script-invoking URL;
  - Generated automatically by the HTML tag <ISINDEX> or a fill-out form (with method=GET);
  - Encoded to include URL information with spaces converted to "+" and special characters according to their hex encoding;
  - Example of `QUERY_STRING` use.

---

|Next|

# Providing Input to a CGI Script

- The **PATH_INFO** environmental variable
  - CGI allows additional context-specific information to be embedded in a URL;
  - This additional information is contained in the **PATH_INFO** environmental variable;
  - The information in **PATH_INFO** is not encoded;
  - Example of `PATH_INFO` use.

---

[Next]

# Providing Input to a CGI Script

- Standard Input (stdin)
  - Used with fill-out forms using method=POST;
  - The environmental variable **CONTENT_LENGTH** contains the amount of data to be read from standard input;
  - Example of `CONTENT_LENGTH` use.

---

|Next|

# Guide to Writing CGI Scripts in REXX or Perl

*9 Apr, 1995*



## Contents



## Introduction

This Guide is aimed at people who wish to write their own WWW executable scripts using WWW's *Common Gateway Interface* ( CGI). Since there are security and other risks associated with executing user scripts in a WWW server, the reader may wish to first view a document providing information on a SLAC Security Wrapper for users' CGI scripts. Besides improving security, this wrapper also simplifies the task of writing a CGI script for a beginner.

The CGI is an interface for running external programs, or gateways, under an information server. Currently, the supported information servers are HTTP (the Transport Protocol used by WWW) servers.

Gateway programs are executable programs (e.g. UNIX scripts) which can be run by themselves (but you wouldn't want to except for debugging purposes). They have been made executable to allow them to run under various (possibly very different) information servers interchangeably. Gateway programs conforming to this specification can be written in any language, including REXX or Perl, which produces an executable file

## Getting the Input to the Script

The input may be sent to the script in several ways depending on the client's *Uniform Resource Locator* (URL) or an *HyperText Markup Language* (HTML) Form:

- `QUERY_STRING` Environment Variable

QUERY_STRING is defined as anything which follows the first ? in the URL used to access your gateway. This information could be added by an HTML ISINDEX document, or by an HTML Form (with the GET action). It could also be manually embedded in an HTML hypertext link, or *anchor*, which references your gateway. This string will usually be an information query, e.g. what the user wants to search for in databases, or perhaps the encoded results of your feedback Form. It can be accessed in REXX by using `String=GETENV('QUERY_STRING')`
or in Perl by using
`$string=$ENV('QUERY_STRING');`

This string is encoded in the standard URL format which changes spaces to +, and encoding special characters with %xx hexadecimal encoding. You will need to decode it in order to use it. You can review the REXX or Perl code fragments giving an example of how to decode the special characters.

If your server is not decoding results from a Form, you will also get the query string decoded for you onto the command line. This means that the query string will be available in REXX via the `PARSE ARG` command, or in the Perl `$ARGV[n]` array.

For example, if you have a URL `http://www.slac.stanford.edu/cgi-bin/foo?hello+world` and you use the REXX command `PARSE ARG Arg1 Arg2` then `Arg1` will contain `"hello"` and `Arg2` will contain `"world"` (i.e. the + sign is replaced with a space).
In Perl `$ARGV[1]` contains `"hello"` and `$ARGV[2]` contains `"world"`. If you choose to use the command line to access the input, you need to do less processing on the data before using it.

- `PATH_INFO` Environment Variable

Much of the time, you will want to send data to your gateways which the client shouldn't muck with. Such information could be the name of the Form which generated the results they are sending.

CGI allows for extra information to be embedded in the URL for your gateway which can be used to transmit extra context-specific information to the scripts. This information is usually made available as "extra" information after the path of your gateway in the URL. This information is not encoded by the server in any way. It can be accessed in REXX by using
`String=GETENV('PATH_INFO')`, or in Perl by using `$string=$ENV('PATH_INFO');`

To illustrate this, let's say I have a CGI script which is accessible to my server with the name `foo`. When I access foo from a particular document, I want to tell foo that I'm currently in the English language directory, not the Pig Latin directory. In this case, I could access my script in an HTML document as:

```
<A HREF="http://www/cgi-bin/foo/language=english">foo</A>
```

When the server executes foo, it will give me `PATH_INFO` of `/language=english`, and my program can decode this and act accordingly.

The PATH_INFO and the QUERY_STRING may be combined. For example, the URL:
`http://www/cgi-bin/htimage/usr/www/img/map?404,451`
will cause the server to run the script called `htimage`. It would pass remaining path information

"/usr/www/img/map" to htimage in the PATH_INFO environment variable, and pass "405,451" in the QUERY_STRING variable. In this case, htimage is a script for implementing active maps supplied with the CERN HTTPD.

- Standard Input

  If your Form has METHOD="POST" in its FORM tag, your CGI program will receive the encoded Form input on standard input (stdin in Unix). The server will NOT send you an EOF on the end of the data, instead you should use the environment variable CONTENT_LENGTH to determine how much data you should read from stdin. You can accomplish this in REXX by using
  ```
  In=CHARIN(,1,GETENV('CONTENT_LENGTH')), or in Perl by using
  read(STDIN,$in,$ENV{'CONTENT_LENGTH'});
  ```

You can review the REXX Code Fragment giving an example of how to read the various form of input into your script.

# 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. The value each of the input items is given by the user is called the value.

Form data is a stream of name=value pairs separated by the & character. Each name=value pair is URL encoded, i.e. spaces are changed into plusses and some characters are encoded into hexadecimal.

You can review the REXX or the Perl code fragment giving examples of decoding the Form input.

# Sending Document Back to Client

CGI programs can return a myriad of document types. They can send back an image to the client, an HTML document, a plaintext document, a Postscript documents or perhaps even an audio clip of your bodily functions. They can also return references to other documents (to save space we will ignore this latter case here, more information may be found in NCSA's CGI Primer). The client must know what kind of document you're sending it so it can present it accordingly. In order for the client to know this, your CGI program must tell the server what type of document it is returning.

In order to tell the server what kind of document you are sending back, CGI requires you to place a short header on your output. This header is ASCII text, consisting of lines separated by either linefeeds or carriage returns followed by linefeeds. Your script must output at least two such lines before its data will be sent directly back to the client. These lines are used to indicate the MIME type of the following document

Some common MIME types relevant to WWW are:

- A "text" Content-Type which is used to represent textual information in a number of character sets and formatted text description languages in a standardised manner. 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, which is used to transmit application data or binary data. Two frequently used subtypes are:
    - `application/postscript`: The data is in *PostScript*, and should be fed to a *PostScript* interptreter.
    - `application/binary`: the data is in some unknown binary format, such as the results of a file transfer.
- An `"image"` Content-Type for transmitting still image (picture) data. There are many possible subtypes, but the ones most often used on WWW are:
    - `image/gif`: an image in the GIF format.
    - `image/xbm`: an image in the X Bitmap format.
    - `image/jpeg`: an image in the JPEG format.

In order to tell the server your output's content type, the 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 the second line. With the current specification, THE SECOND LINE SHOULD BE BLANK. This means that it should have nothing on it except a linefeed. Once the server retrieves this line, it knows that you're finished telling the server about your output and will now begin the actual output. If you skip this line, the server will attempt to parse your output trying to find further information about your request and you will become very unhappy.

You can review a REXX Code Fragment giving an example of handling the `Content-type` information.

After these two lines have been outputted, any output to `stdout` (e.g. a REXX SAY command) will be included in the document sent to the client.

# Diagnostics and Reporting Errors

Since `stdout` is included in the document sent to the, diagnostics diagnostics outputted with the SAY command will appear in the document. This output will need to be consistent with the `Content-type: type/subtype` mentioned above.

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, too many arguments specified, requested an invalid command to be executed, invalid syntax in the REXX exec) 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 that are set.

You can review a REXX Code Fragment giving an example of error reporting and Typical Output Generated from such a code fragment.

# My First REXX CGI Script

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

- Write the script and save it somewhere. For example let's say we write a trivial REXX script called `cgi1.rxx` and save it in our home bin directory (e.g. in `/u/sf/cottrell/bin/cgi1.rxx`).
- Make the script executable by your Web server. On Unix this is done using the `chmod` command, e.g.
  - `chmod o+x /u/sf/cottrell/bin/cgi1.rxx`
- Get your *Web-Master* to add a rule to the Web server's rules file to allow the Web server to execute your script. You can look at SLAC WWW Rules to see how SLAC URLs currently map to the UNIX file system.

The Web-Master will want to insure that Security Aspects of your script have been addressed before adding your script to the Rules file.

# Acknowledgements

Much of the text on the Common Gateway Interface and Forms comes from NCSA documents. Useful information and text was also obtained from *The World-Wide Web: How Servers Work*, by Mark Handley and John Crowcroft, published in ConneXions, February 1995.

---

*Les Cottrell 15 Jan 1995*
[ Top | Suggestion Box | Disclaimer ]

```
/* ********************************************************************** */
/* Some browsers insert ASCII codes (preceded by a %) for some characters */
/* such as space or +.  We replace them by the appropriate character      */
/* N.B. the encodings maybe in upper or lower case (e.g. %2F=/ or %2f=/   */
/* ********************************************************************** */
Hex='%2b=+&%20=+&%2f=/&%3f=+&'
/*   %2b=+ %20=  %2f=/ %3f=? */
Hex=Hex||TRANSLATE(Hex) /*Allow for upper case*/
IF POS('%',Str)/=0 THEN DO/*Any %s in input*/
 DO UNTIL Hex=''/*Check for hex codes*/
  PARSE VAR Hex Code'='Char'&'Hex; Out=''
  PARSE VAR Str Pre (Code) Str
  DO WHILE (Str /== '')
     Out=Out||Pre||Char
     PARSE VAR Str Pre (Code) Str
  END /*DO*/
 Str=Out||Pre
 END /*DO*/
END /*IF*/
```

# Reading HTML Forms Input in REXX

```
PARSE ARG Parms
StdinFile='/tmp/wrap-stdin'_GETPID()/* Get unique name*/
Script=GETENV('SCRIPT_NAME')
...
/* ************************************************** */
/* Read the input from the various possible sources      */
/* Note that we preserve or save all                     */
/* input in case we need to send it to another command.  */
/* If so we can restore the stdin for the called command */
/* by calling it using the REXX command:                 */
/* ADDRESS UNIX script '<' StdinFile                     */
/* ************************************************** */
IF GETENV('REQUEST_METHOD')="POST" THEN DO
   IF LINES()=0 THEN,
      CALL Exit 400, Script': null input from POST method!<br>'
   Line.0=0
   DO L=1 BY 1 WHILE LINES()>0
      Line.L=LINEIN(); Line.0=Line.0+1
      IF L>1 THEN Fail=LINEOUT(StdinFile,Line.L)
      ELSE        Fail=LINEOUT(StdinFile,Line.L,1)
      IF Fail=0 THEN LEAVE L
   END L
   Fail=LINEOUT(StdinFile) /* Close the file*/
END
Path_info=GETENV('PATH_INFO')/*Insert path info in front*/
IF Path_info/='' THEN,
   Cmd=SUBSTR(Path_info,2) Cmd/*Remove leading / from path*/
Temp=GETENV('QUERY_STRING')
IF Temp='' THEN Temp=Parms
IF Temp/='' THEN Cmd=Cmd Temp /* Insert query info at end*/
IF Cmd='' THEN,
   CALL Exit 402, Script': no input provided!<br>'
...
```

# Example of Decoding HTML Forms input in REXX

```
/* ************************************ */
/* Input from a form comes in the form:  */
/* name1=value1&name2=value2             */
/* Here we decode the input into an      */
/* array of names and values.            */
/* ************************************ */
DO I=1 BY 1 UNTIL Input=''
   PARSE VAR Input Name.I'='Value.I'&'Input
END I
```

## Handling Content-type Info in REXX

```
/* ************************************************/
/* Code fragment to set the Contentype subtype based */
/* on the  file type (as determined by the characters*/
/* following the last period in the filename).       */
/* ************************************************/
FileName='/u/sf/cottrell/public_html/cgi.html'
...
L=LASTPOS('.',Filename); Type=''
IF L>0 THEN DO
  IF LENGTH(FileName)>L THEN
     Type=TRANSLATE(SUBSTR(FileName,L+1))
END

SELECT
   WHEN Type='HTM' | Type='HTML' THEN
      SAY 'Content-type: type/html'
   WHEN Type='PS'                THEN
      SAY 'Content-type: application/postscript'
   WHEN FIND('TXT RXX PL FOR C',Type)/=0 | Type='' THEN
      SAY 'Content-type: type/text'
   ...
   OTHERWISE DO
      SAY 'Content-type: type/html'; SAY ''
      CALL Exit 409, 'Unknown Content-type="'Type'".'
   END
END
SAY ''
...
```

# Reporting CGI Diagnostics in REXX

```
/* ************************************ */
/* Code fragment for reporting CGI Script */
/* diagnostic                           */
/* ************************************ */
PARSE SOURCE $ArchName $Efn $Fn .
...
Debug=1
SAY "Content-type: text/html";  SAY
...
IF Debug>0 THEN SAY,
   $Efn': PATH_INFO="'GETENV('PATH_INFO')'".<br>'
```

# Reporting CGI Errors in REXX

```
/****************************************** */
/*Code Fragment for REXX CGI Error Reporting*/
/****************************************** */
  ADDRESS 'COMMAND'; SIGNAL ON SYNTAX
  PARSE Arg Parms
  ...
  IF QueryInput='' THEN,
     CALL Exit 490,'No input given!<br>'
  ...
/*****************************************/
/*Rex will jump to this error exit if a    */
/*syntax error occurs. It returns the user */
/*ito the line in the exec with the error. */
/*****************************************/
Syntax:
   PARSE SOURCE Arch . $Fn .
   CALL Exit 499, 'Syntax error on line',
      SIGL 'of' $Fn'. Line="'SOURCELINE(SIGL)'"'
   ...
Exit: PROCEDURE EXPOSE Debug Parms
/* ************************************
| Exit - Assumes Content-type: text/html
| ************************************ */
PARSE ARG Code, Msg
SAY '<title>'GETENV('SCRIPT_NAME')' error' Code'</title>'
SAY '<h2>Error Code' Code 'reported by'
SAY GETENV('SCRIPT_NAME')'.</h2> The WWW utility on'
SAY '<tt>'GETENV('SERVER_NAME')
SAY ':'GETENV('SERVER_PORT')'</tt>that you are using'
...
SAY 'which reports the following error:'
IF Msg/='' THEN SAY '<hr><h1><code>'Msg'</code></h1>'
IF Debug>0 THEN DO
   SAY '<hr>The complete environment follows:<p><pre>'
   ADDRESS Unix "tcsh -c printenv"
   SAY '<br>Arguments="'Parms'".'
   SAY '</pre>'
END
SAY '<hr>['
SAY '<a HREF="/slac.html">SLAC Home Page</a> |'
SAY '<a HREF="/suggestion/cottrell">Suggestion Box</a> ]'
SAY '<address><a HREF="/owner/zaphod">Zaphod</a></address>'
IF Code=0 THEN RETURN
EXIT /*Code*/
```

# Error Code 490 reported by cgi-wrap

The WWW utility on `www1.slac.stanford.edu:80` that you are using from your WWW browser (*Mozilla/1.1b1 (X11; international; AIX 2 000003643500)*) on `ATLAS.SLAC.Stanford.EDU` called (using the `GET` method) a Common Gateway Interface (revision `CGI/1.1`) script command wrapper, which reports the following error:

---

**No input given!**

---

[ SLAC Home Page | Suggestion Box ]
*Les Cottrell*

## Sample Dangerous CGI Script in REXX

```
#!/usr/local/bin/rxx
/* Sample dangerous CGI Script written in */
/* Uni-REXX.                              */

SAY "Content-type: text/plain"; SAY

Query=TRANSLATE(GETENV('QUERY_STRING'),' ','+')

Valid=' abcdefghijklmnopqrstuvwxyz'
Valid=Valid||'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
Valid=Valid||'0123456789-_/.@'
V=VERIFY(Query,Valid)
IF V/=0 THEN DO
   SAY 'Invalid Character('SUBSTR(Query,V,1),
       ')in:"'Query'".'
   EXIT 99
END

ADDRESS UNIX Query

EXIT RC
```