VM/ESA and VSE/ESA Technical Conference
Orlando - Summer 2000
Session # M23

Writing CGI Scripts

Richard F. Lewis
IBM Washington System Center
rflewis@us.ibm.com

RETURN TO INDEX

Disclaimer

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either expressed or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. customers attempting to adapt these techniques to their own environments do so at their own risk.

In this document, any references made to an IBM licensed program are not intended to state or imply that only IBM's licensed program may be used; any functionally equivalent program may be used instead.

Any performance data contained in this document was determined in a controlled environment and, therefore, the results which may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environments.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming or services in your country.

Trademarks

The following are trademarks of International Business Machines Corporation. Those identified with an (*) are registered trademarks of International Business Machines

- Enterprise Systems Architecture/370
- Enterprise Systems Architecture/390
- ESA/370
- ESA/390
- IBM*
- System/370
- VM/ESA*

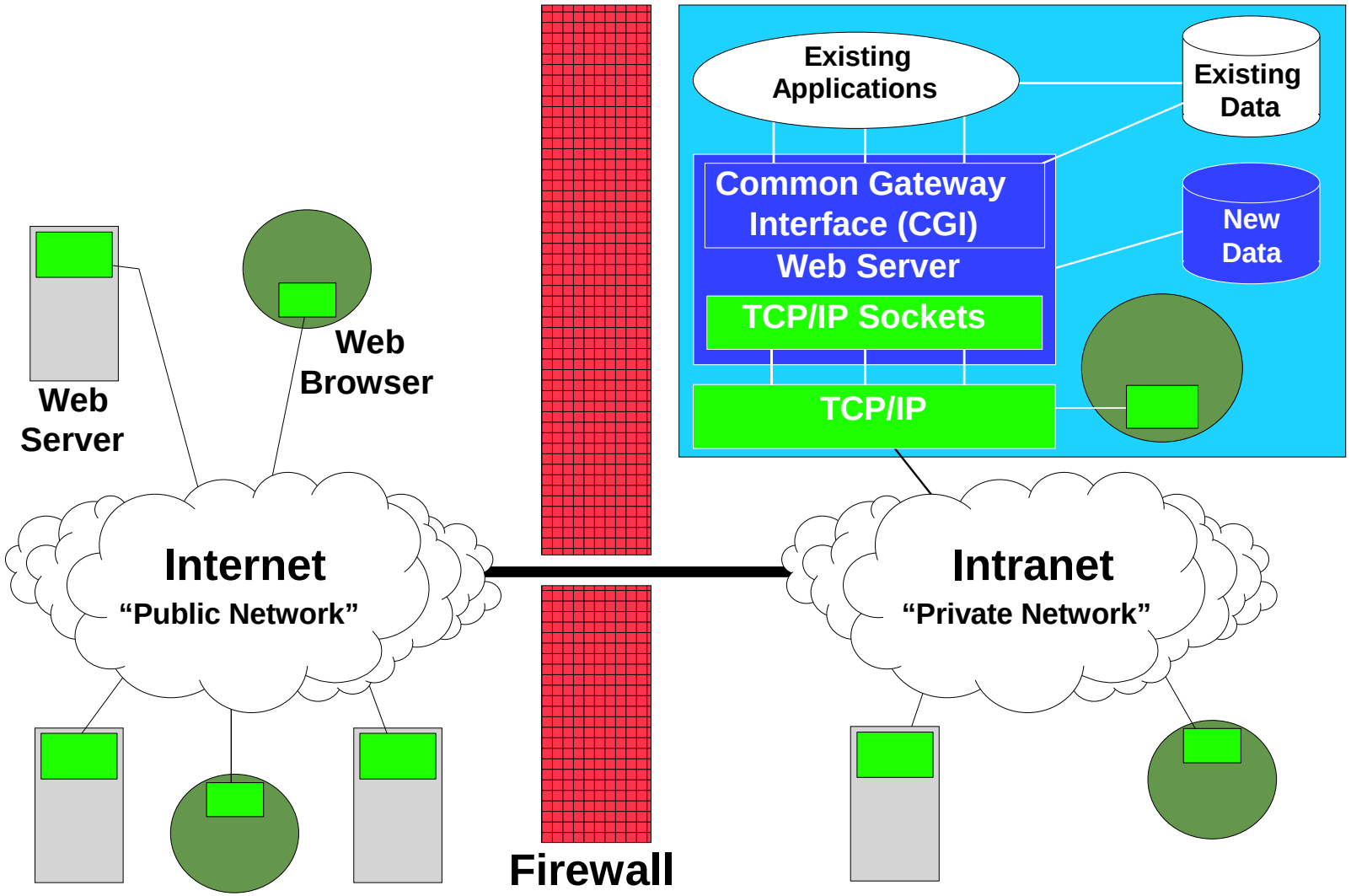
Following are trademarks of the company specified

- VM:Webserver - Sterling Software, Inc.
- EnterpriseWeb/VM - Beyond Software Incorporated
- Netscape Netsite - Netscape
- Microsoft IIS - Microsoft
- Java - Sun Microsystems
- ESAWEB - Velocity Software

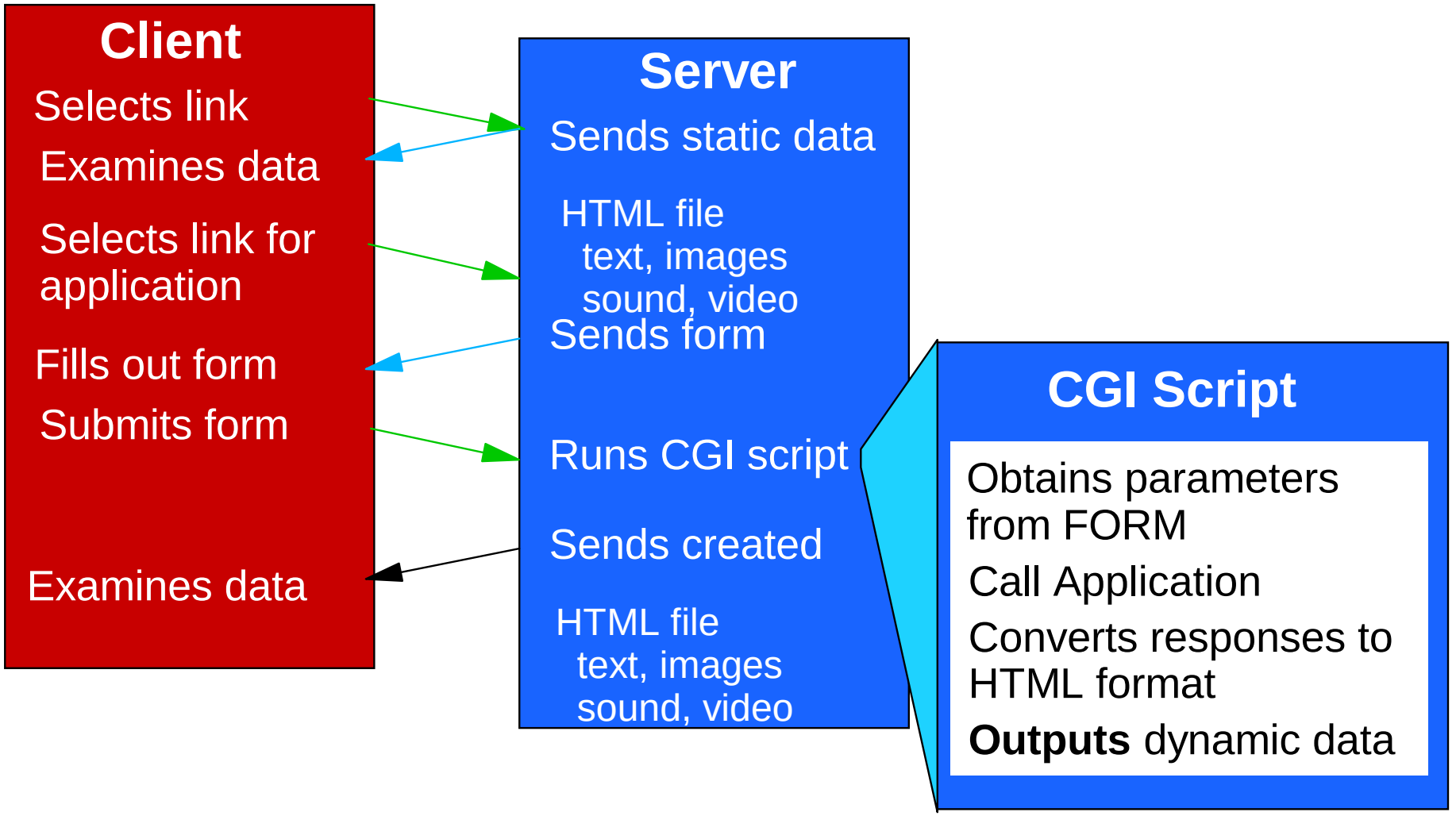
Agenda

- Introduction
  - Overview of WWW
- Overview of CGI Programming
  - What is a CGI
  - Operation of a CGI
  - Headers
- How to:
  - Send output to browser
  - Access Environment Variables
  - Access request headers
  - Obtain data sent from a browser using POST
- Basic flow of a CGI
- Considerations
- A brief look into the future
- Summary

Overview


Existing Applications
Existing Data
Common Gateway Interface (CGI)
Web Server
New Data
TCP/IP Sockets
TCP/IP
Web Server
Web Browser
Internet
"Public Network"
Intranet
"Private Network"
Firewall

What Does a Web Server Do?

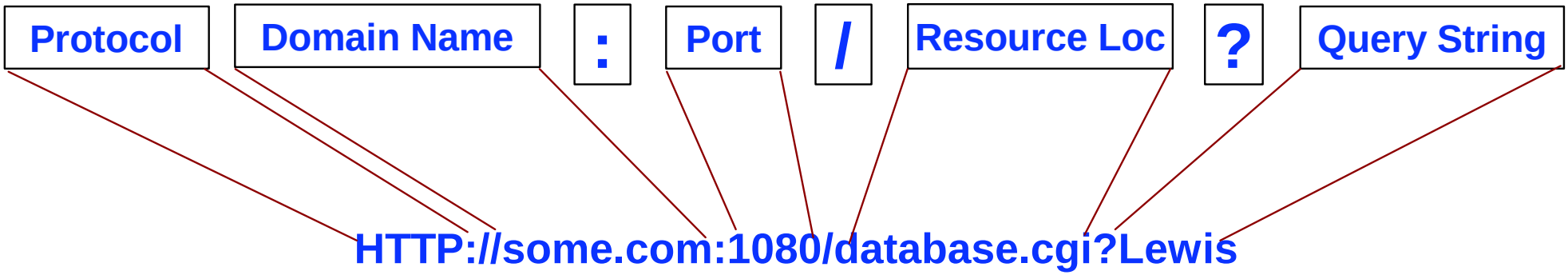


What is a CGI?

- Hypertext Transfer Protocol (HTTP) provides means for browser to send data to server
  - GET, POST, PUT
- Servers not written to process this data
  - Servers typically as generic as possible
- Common Gateway Interface is specification that determines how server will link to separate program to process the data
  - Defines how:
    - Browser sends data to server
    - Server provides data to program
    - Program passes data back to server
- Example:
  - Browser sends database query to server
  - Server needs to link to a program that can interface with database
  - Server passes query from browser to program

Browser Sending Data To Server...

- Universal Resource Locator (URL), query string
- URL
  - Specify internet resources using single line of characters
  - Flexible scheme
  - Supports all major internet resources (FTP, Gopher, e-mail, http...)
  - Form:



- Data in query string passed to CGI program
- Encoding of characters done:
  - For characters disallowed in a URL
  - For special characters (%xx where xx is hex code for ASCII char)

Browser Sending Data To Server

- Extra Path Information
  - Resembles directory information added to URL after name of CGI but before any query string information
  - E.g. http://some.com/cgi-bin/proga/extra/path?query+string
- Data sent to server in message body
  - HTTP POST method
  - Used with HTML FORMs
  - Different encoding schemes used by different browsers
    - Form-urlencoded
    - MIME/form-data
  - Server handles decoding of message body

Server Sending Data To Program

- Command line argument
  - Data from document with ISINDEX HTML tag, or FORM, GET method
  - Data sent from browser as encoded query string, with + delimited words
  - Server decodes query string, replaces + sign with blank
  - CGI program receives blank delimited words as command line input
- Environment variables
  - Variable names and contents defined by CGI standard
  - Variables loaded by server before launch of CGI program
  - Variables contain extra path information, and query string information
  - Other variables contain
    - Request header information
    - Server information
  - VM webservers typically use GLOBALV for environment variables
- Standard input
  - Typically used with POST method from an HTML FORM
  - Program receives name=value strings

Program Passes Data To Server

- Write to standard output
  - Most common method for program to send results to browser
  - Data stream includes headers and HTML source
  - Output may also include server directives
  - Server adds headers to data from CGI program
- Non-parsed header program
  - Specially named CGI
    - Program name begins with nph
  - Server sends CGI output directly to browser
  - No processing of output by server
  - No additional headers created by server
  - CGI responsible for providing all headers along with output data
  - On VM this is implemented by:
    - Options to commands used to write output
    - Accessing the socket write routines directly from a CGI

Operating Characteristics

- HTTP protocol is a stateless protocol
  - Server and CGI retain no knowledge of previous transactions
- CGI program must have mechanisms for keeping track of information from previous transaction
- Most common method is to use TYPE="hidden" INPUT elements within HTML forms
  - Data passed back and forth between client and server
  - End user at browser never sees data
- Netscape extension provides alternative
  - Cookies
    - Set-cookie: HTTP response header, name=value data
  - Browser stores state information sent by server, on hard drive
  - Cookie content sent as part of HTTP request header when appropriate
    - Tied to URL domain name value
    - Defined in Set-cookie header
  - Only supported by Netscape and Internet Explorer browsers

Request Headers

- Part of data sent from client to server
- Request header consists of:
  - Request fields (single line of ASCII text terminated with CRLF)
  - Blank line with CRLF terminus
- Method field of request header is always first
  - Specifies HTTP method to use
  - Location of desired resource on server
  - Version of HTTP protocol client would like to use
- Request fields follow method field
  - Provide information to server about client
  - Information about the nature of data (if any) being sent by client
- Server typically places all data from request headers into environment variables

Example Request Headers


```
GET /Tests/file.html HTTP/1.0
Accept: text/plain
If-Modified-Since: Wed, 19 May 1999 17:23:31 GMT
Referer: http://www.place.com/webstuff/page.html
User-Agent: Mozilla/1.01
 a blank line with CRLF
```

Response Headers

- Part of data returned to client from server
- Consists of:
  - Response header
  - Response header fields
    - Single lines of text terminated by a CRLF
  - Response header terminated by blank line containing only CRLF
- Used to communicate information about a txn to the client
- Message data may follow response header
- First line in response header is status line
  - Identifies protocol level used by server, and whether request successful
  - Format:
    - http_version status_code explanation
  - Other response header fields include:
    - Date header - date and time object was assembled for transmission
    - MIME-version field - MIME version used by server to compose response
    - Server field - returns name and version of server software
    - Content-type - indicates MIME content-type of date being returned

Example Response Headers


```
HTTP/1.0 200 OK
Content-Type: text/html
Last-Modified: Tue, 18 May 1999 22:53:06 GMT
Server: Aserver/01.2
Date: Wed, 19 May 1999 13:19:29 GMT
 a blank line with CRLF
<HTML>
<HEAD>
```

CGI Execution Environment

- Method by which CGIs are invoked differs by web server
  - EnterpriseWeb/VM and Webshare
    - CGI is a pipeline filter (user written stage)
    - 3 input streams (only two with Webshare)
      - ✓Primary - input parameters from POST method or query string from GET
      - ✓Secondary - header fields from browser
      - ✓Tertiary - environment variables (on Webshare environment variables are accessed from GLOBALV group HTTPD)
    - 1 output stream
      - ✓Primary - contents sent directly to client browser
  - VM:Webgateway
    - Preferred environment for a CGI program is a REXX EXEC
    - Pipeline command can be included in the EXEC
    - Input from client, output to client, and environment variables processed by CGI REXX function
    - CGI can be a pipeline filter for compatibility with WebShare
      - ✓Less efficient environment than the preferred REXX EXEC environment
    - CGI programs can be run in the server virtual machine, or in a worker (adjunct) virtual machine

Storing CGI Programs

- CGI programs are stored in the file system managed by the webserver (SFS, BFS, Minidisk)
  - Could be stored along with html files in directories that reflect the URL path to the CGI program
  - Could be stored in a central CGI location (web server may have special settings to facilitate this)
- CGI programs are named with any file name, and a file type that matches an entry in the web server's file type table or media map table identifying an executable file
  - Standard file type is CGI
  - Enterprise/Web VM
    - EWEBCGI is an executable file type
  - VM:Webgateway
    - CGI, CGIREXX, HTTPPROC for Webshare compatibility
    - WRKEXEC - REXX CGI running in a worker virtual machine
    - VMGW - REXX CGI running in the server
- Modify the table to add a new file type

Invoking CGI Programs

- CGI programs typically invoked in one of 3 ways:
  - Using a URL:
    - http://wscm2vm/abstract.vmgw?speaker=Lewis
    - http://wscm2vm/abstract.ewebcgi
  - From an html page hyperlink (browser user clicks on the text)
    - some text
    - some text
  - From an html page form tag
    - <form action="abstract.vmgw" method="POST" ...>
    - <form action="abstract.ewebcgi method="GET" ...>

Determining Web Server Execution Environment

- You can determine whether or not you are running as a pipeline stage using the REXX PARSE SOURCE statement
  - Pipeline stage
    - Parse SOURCE . . . . . env .
    - Say env (results in ?)
  - Called EXEC
    - Parse SOURCE . . . . . env .
    - Say env (results in CMS)
- You can identify server software executing CGI, by examining the SERVER_SOFTWARE global variable
  - Address "COMMAND" 'GLOBALV SELECT HTTPD GET SERVER_SOFTWARE'
  - Say SERVER_SOFTWARE (results in)
    - VM:Webgateway/3.0 (for Sterling Software's VM:Webgateway)
    - EnterpriseWeb/1.1.4 (for Beyond Software's EnterpriseWeb/VM)
    - Webshare/1.2.3 (for Rick Troth's Webshare)

Sending Output To Browser VM:Webgateway

- Use WRITE option of VM:Webgateway CGI command
  - Write header information, and document (body) information
  - Translate output from EBCDIC to ASCII
  - Add CRLF to end of each line
  - Write from string, variable, or stem
  - For binary data (image) specify TRANSLATE NONE, and NOCRLF

```
/******/
Line.0 = 3
Line.1 = "<html><body>"
Line.2 = "<p>Some output"
Line.3 = "</body></html>"
'CGI WRITE HEADER (STRING Status: 200 OK'
headerdata = "Content-Type: text/html'
'CGI WRITE HEADER (VAR HEADERDATA'

'CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STEM LINE.'
```

- VIGRTNS SUBSTITUTE is a utility that can be used to replace variables in an html file with values of Rexx variables in CGI program

Sending Output To Browser EnterpriseWeb/VM

- Everything from the primary output stream of the CGI is returned to the browser
  - Headers
  - Document data (body)

```
/*****/
'OUTPUT' "HTTP/1.0 200 OK"
'OUTPUT' "Content-Type: text/html"
'OUTPUT' "<html><body>"

'CALLPIPE < BEGIN DATA * | *:'

'OUTPUT' "</body></html>"
```

- EWSET is a utility provided to simplify updating variable data in an html form being sent to a browser
  - Use to initialize a form, or redisplay form with partially filled data
  - Reads html on primary input stream, list of variables to update on secondary stream

Sending Output To Browser Webshare

- Everything from the primary output stream of the CGI is returned to the browser
  - Headers
  - Document data (body)

```
/*****/
'OUTPUT' "HTTP/1.0 200 OK"
'OUTPUT' "Content-Type: text/html"
'OUTPUT' "<html><body>"

'CALLPIPE < BEGIN DATA * | *:'

'OUTPUT' "</body></html>"
```

Lab Notes

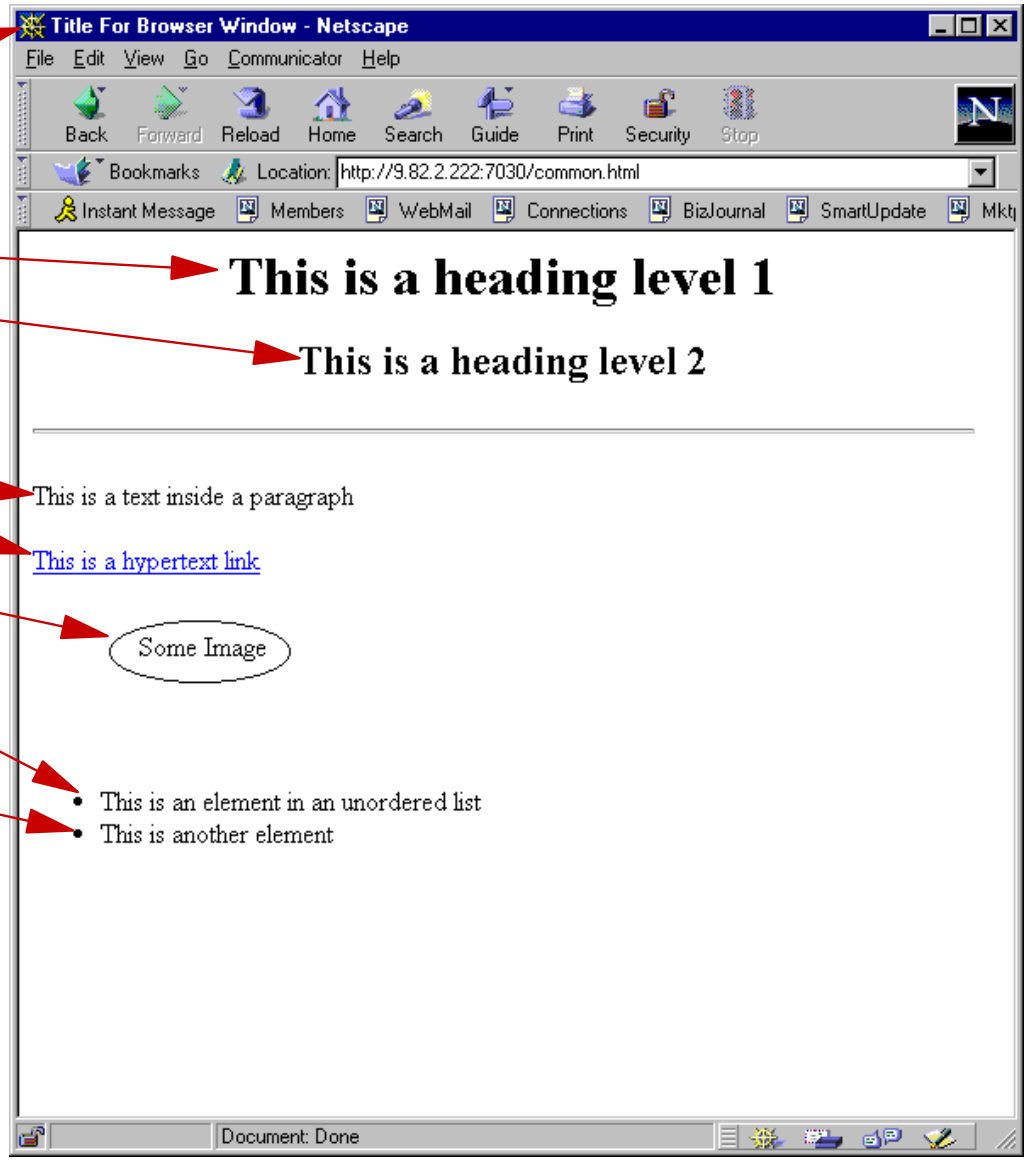
- Each team will logon to CGIUSERn where:
  - n = 1- F
  - password = CGIUSERn
- Each Thinkpad has Netscape browser installed, URL to use is:
  - http://9.130.31.15:xxxx/CGIUSERn/file.html where:
    - xxxx = 9080 for VM:Webserver
    - xxxx = 6080 for Webshare
    - xxxx = 3080 for Enterprise/Web
- Each CGI userid has three SFS directories accessed
  - To see where the directories are accessed enter QUERY ACCESSED from CMS command line
  - CGICLASS.CGIUSERn is directory to place all objects such as html, or cgi programs
  - HTTPD3.WEBSHARE.CGIUSERn is directory for Webshare objects
    - May want to just alias all items placed into CGICLASS.CGIUSERn
    - ALIAS * * CGICLASS.CGIUSERn = = HTTPD3.WEBSHARE.CGIUSERn
  - Example files are in CGICLASS.EXAMPLES

Lab Assignment 1

- We will start off with a trivial CGI program so that you can get used to the mechanics of creating the CGI file, saving it in the proper place, and testing it with your web browser
- Create a CGI that simply displays the text "Hello World"
  - Output of the CGI should be <html><body><p>Hello World</body></html>
  - The CGI should be invoked from your web browser using a URL similar to:
    - http://9.130.31.15:xxxx/CGIUSERn/hellow.vmgw (or .ewebcgi)
- If you want more of a challenge code the CGI program to run under any of the web servers
- If you are done early with this assignment, feel free to add additional output to your CGI, for example the date and time, and the system netid of the host
- Example files:
  - LAB1 EXAMPLE, SINGLE SOURCE, COMMON HTMLELEM, FORM HTMLELEM

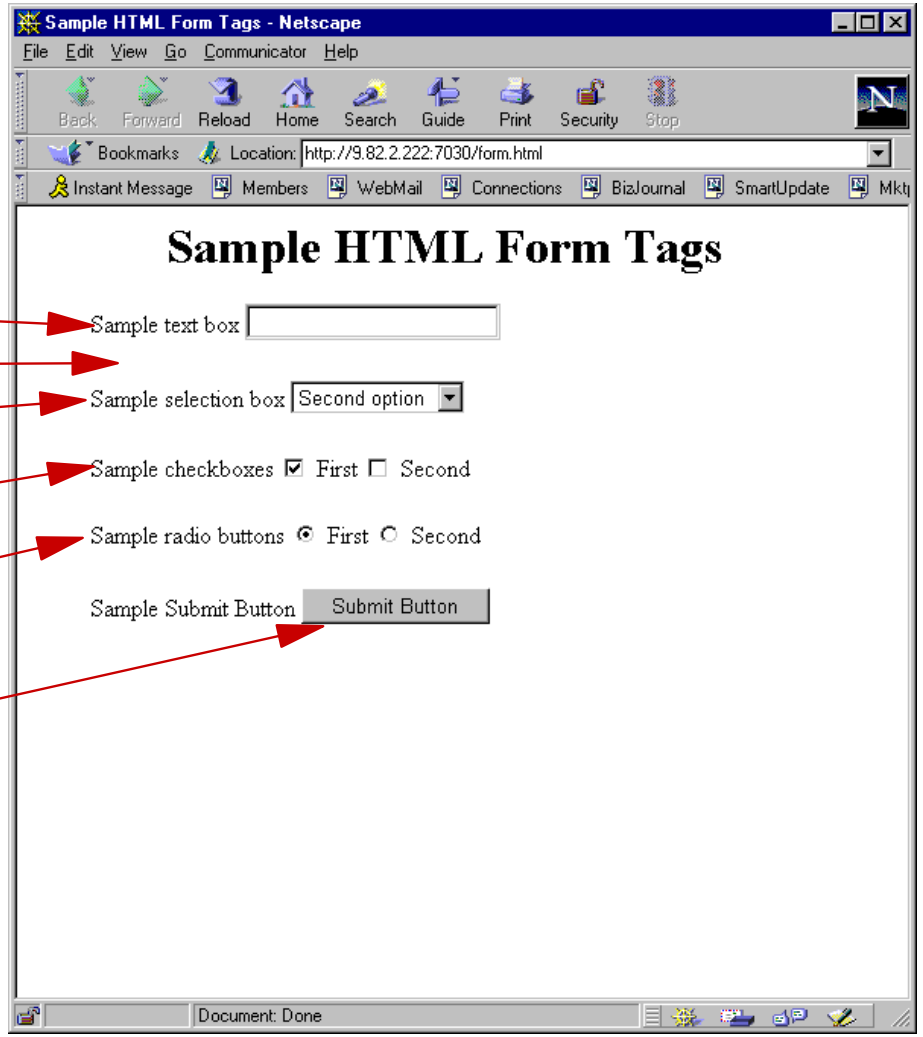
Some Sample HTML Tags

```
<html>
<head>
<title>Title For Browser Window</title>
</head>
<body>
<center>
<h1>This is a heading level 1</h1>
<h2>This is a heading level 2</h2>
</center>
<hr>
<br>
<p>
This is a text inside a paragraph
</p>
<a href="some.html">This is a hypertext link</a>
<img src="some.gif" alt="Text if image not
displayed">
<ul>
<li>This is an element in an unordered list
<li>This is another element
</ul>
</body>
</html>
```



Sample HTML Form Tags

```
<html>
<head>
<title> Sample HTML Form Tags </title>
</head>
<body>
<center><h1> Sample HTML Form Tags </h1> </center>
<blockquote>
<form action="some.cgi" method="post">
<p>
Sample text box
<input type="text" name="atextb" size="20">
<br>
<img src="blank.gif" height="20" width="2">
<br>
Sample selection box
<select name="aselect">
  <option> First option
  <option selected> Second option
</select>
<br>
Sample checkboxes
<input type="checkbox" name="achkb" value="First" checked> First
<input type="checkbox" name="achkb" value="Second"> Second
<br>
Sample radio buttons
<input type="radio" name="aradio" value="First" checked> First
<input type="radio" name="aradio" value="Second"> Second
<br>
Sample Submit Button
<input type="submit" name="sub" value="Submit Button">
</form>
</blockquote>
</body>
</html>
```



Break

Accessing Environment Variables VM:Webgateway

- Use the GETVAR option of VM:Webgateway CGI command
  - Will retrieve value of single variable or all variables in a stem
  - With stem, element zero contains a list of names, rather than a count
- Example below obtains all environment variables, and logs values to console

```
/****/
'CGI GETVAR * ( STEM CGIVARS.'

Do I = 1 to WORDS(CGIVARS.0)
  varname = WORD(CGIVARS.0,I)
  'CGI LOG (VAR CGIVARS.'varname
End

Exit
```

Data Examples

```
Assuming WORD(CGIVARS.0,2) is
HTTP_USER_AGENT

Say CGIVARS.HTTP_USER_AGENT
  results in: Mozilla/2.02E (OS/2;I)
```

Accessing Environment Variables EnterpriseWeb/VM

- Use CMS GLOBALV command with group HTTPD
- Transaction specific information can be obtained from group THREADn (where n represents a service machine)

```
/****/
'CALLPIPE COMMAND GLOBALV SELECT HTTPD LIST',
   '| DROP FIRST 1',
   '| STRIP',
   '| XLATE fieldsep = f1 upper',
   '| change //=PARM./',
   '| varload'

Say PARM.VERSION
'StreamState INPUT 2'
If rc=0 | rc = 4 Then
  Do
    'CALLPIPE *.input.2: | varload'
  End
'CALLPIPE COMMAND GLOBALV SELECT THREAD'eweb.thread 'LIST',
  '| DROP FIRST 1',
  '| STRIP',
  '| XLATE fieldsep = f1 upper',
  '| change //=PARM./',
  '| varload'

Say PARM.REMOTE_ADDR
```

```
Say PARM.VERSION would result in:
 EnterpriseWeb 1.1.4 VM/CMS

Say PARM.REMOTE_ADDR would result in:
 9.82.1.101
```

Accessing Environment Variables Webshare

- Use CMS GLOBALV command with group HTTPD

```
/***/
'CALLPIPE COMMAND GLOBALV SELECT HTTPD LIST',
   '| DROP FIRST 1',
   '| SORT',
   '| NLOCATE 1-6 / TYPE./',
   '| NLOCATE 1-6 / PIPE./',
   '| NLOCATE 1-7 / GTYPE./',
   '| NLOCATE 1-7 / MTYPE./',
   '| STRIP',
   '| XLATE fieldsep = f1 upper',
   '| CHANGE //=PARM./',
   '| VARLOAD'

Say PARM.HTTP_USER_AGENT
```

Accessing Request Headers VM:Webgateway

- Use GETVAR option of VM:Webgateway CGI command
  - Request headers have names beginning with HTTP
    - E.g. HTTP_REFERER, HTTP_USER-AGENT
  - Obtain single header, or all headers into a stem

```
/****/
'CGI GETVAR HTTP_REFERER (VAR header_host'

Say header_host

Exit
```

```
Result of Say statement would be:

Referer: http://www.place.com/webstuff/page.html
```

Accessing Request Headers EnterpriseWeb/VM

- Request headers sent to secondary input stream of CGI
  - No reformatting of header data done by server
  - Can use algorithm shown for Webshare
  - EnterpriseWeb/VM provides option on EWGET command to retrieve headers

```
/********/

'CALLPIPE EWGET headers'

Do I = 1 to header.0
   Say header.I
End

Say header.1
```

```
EWGET with headers option creates stemmed array
containing the request headers.  Element 0 contains
the number of elements in the array.

Result of Say header.1 would be:

Referer: http://www.place.com/webstuff/page.html
```

Accessing Request Headers Webshare

- Request headers sent to secondary input stream of CGI
  - No reformatting of the header data is done by server

```
/******/

'CALLPIPE (endchar ?) *.INPUT.1:',
   '| strip',
   '|a: fanout',
   '| specs /HDR./ 1 w1 n /=/ n w2-* n',
   '| xlate fieldsep = f1 upper',
   '| change /HDR./=HDR./',
   '| change /:=/=/',
   '| varload',                 Algorithm places headers into stemmed array, using the
   '?',                        header name as the name of the array element. Element 0
   'a:',                       contains a list of names used for all other elements
   '| specs w1 1',
   '| change /:/ /',           Results of Say statement would be:
   '| xlate 1-*',
   '| join *',                  Referer is Referer:
   '| var HDR.0'                http://www.place.com/webstuff/page.html

Say 'Referer is" HDR.REFERER
```

Lab Assignment 2

- Create a CGI program to display all environment variables, and the following request header fields
  - Request method
  - User Agent
- Invoke the CGI program with URL similar to Lab 1
- Optionally code the CGI such that it can run under any of the three VM web servers
- Create an html form that has:
  - Title and header information of your choosing
  - A select box listing CP and CMS commands that can be executed
  - A text input box that can be used to enter the user's name
  - A submit button

Note: since this is not an html class, a sample page has been provided in the examples directory that demonstrates how the various html elements are coded (CGICLASS HTML)

Break

Obtaining Data Sent By POST VM:Webgateway

- Use READ option of VM:Webgateway CGI command
  - Translates data from ASCII to EBCDIC
  - Data still in "encoded form" (field_name=value&field_name=value...)
- Use URLDECODE option of VM:Webgateway CGI command
  - Breaks down an encoded string
  - Creates stem listing variable names as well as values
    - stem.0 contains list of variable names
    - stem.var_name contains variable value

```
/****/
'CGI READ 1 (VAR BLOCK TRANSLATE USENGLISH'

If Length(BLOCK) <> 0 Then
  Do
    'CGI URLDECODE (VAR BLOCK INTO DECODED.'
    Do I = 1 to WORDS(DECODED.0)
     'CGI LOG (STRING' WORD(DECODED.0,I)
     'CGI LOG (VAR DECODED.'WORD(DECODED.0,I)
    End
  End
```

```
Say BLOCK results in:
 fname=Richard&mname=F&lname=Lewis

Log output would contain

 ... FNAME
 ... Richard
 ... MNAME
 ... F
 ... LNAME
 ... Lewis
```

Obtaining Data Sent By Post EnterpriseWeb/VM

- Use EWGET utility (pipeline filter)
  - Option VAR, or PARMS will obtain data posted from a form and decode it.
  - Two stems are used
    - First stem specified contains values of variables
    - Second stem specified contains variable names
    - Second stem is a pointer to values in first stem

```
/****/
'CALLPIPE *: | EWGET vars p. ptr.'

Do I = 1 to ptr.0
   'OUTPUT' ptr.I
   name = ptr.I
   Do J = 1 to VALUE(name'.0')
      'OUTPUT' VALUE(name'.'J)
   End
End
```

```
Assuming appropriate html in
the example, the browser
screen would contain:

P.FNAME
Richard
P.MNAME
F
P.LNAME
Lewis
```

Obtaining Data Sent By Post Webshare

- Form data sent using POST method is supplied on the primary input stream to the CGI program
- Melinda Varian of Princeton University has created a very good subroutine pipeline to process this data with Webshare

```
'CALLPIPE (name GetForm)',        /* Input:  name=value          */
  '  *:',                         /* Get input from sample form. */
  '| xlate 1-* 05 40',            /* Convert tabs to blanks.     */
  '| strip',                      /* Strip leading/trailing.     */
  '| locate 1',                   /* discard null lines, if any. */
  '| xlate fieldsep = f1 upper', /* Upper-case variable names.  */
  '| change //=PARM./',          /* ==> =PARM.name=value        */
  '| varload'
```

Obtaining Form Data Sent With GET - VM:Webgateway

- Use GETVAR option of VM:Webgateway CGI command
  - Specify QUERY_STRING as variable to "GET"
  - Data still in "encoded form" (field_name=value&field_name=value+value...)
- Use URLDECODE option of VM:Webgateway CGI command
  - Breaks down an encoded string
  - Creates stem listing variable names as well as values
    - stem.0 contains list of variable names
    - stem.var_name contains variable value
- USE GETVAR option of CGI command with REQUEST_METHOD to determine whether GET or POST was used by client

```
/****/
'CGI GETVAR QUERY_STRING (VAR BLOCK'

If Length(BLOCK) <> 0 Then
  Do
    'CGI URLDECODE (VAR BLOCK INTO DECODED.',
      'MODE TRANSFORMED TRANSLATE USENGLISH'
    Do I = 1 to WORDS(DECODED.0)
      'CGI LOG (STRING' WORD(DECODED.0,I)
      'CGI LOG (VAR DECODED.'WORD(DECODED.0,I)
    End
  End
```

```
Say BLOCK results in:
 fname=Richard&mname=F&lname=Lewis

Log output would contain

 ... FNAME
 ... Richard
 ... MNAME
 ... F
 ... LNAME
 ... Lewis
```

Obtaining Form Data Sent With GET - EnterpriseWeb/VM

- Use same method as described earlier for POST
  - Code would look similar except for initial callpipe
  - callpipe var ARG | EWGet decoded | EWGet vars p. ptr.
- REQUEST_METHOD server variable is located under the thread specific globalv group

```
/****/
'StreamState INPUT 2'
If rc=0 | rc = 4 Then
  Do
    'CALLPIPE *.input.2: | varload'
  End
'CALLPIPE COMMAND GLOBALV SELECT THREAD'eweb.thread 'LIST',
  '| DROP FIRST 1',
  '| STRIP',
  '| XLATE fieldsep = f1 upper',
  '| change //=PARM./',
  '| varload'

Say PARM.REQUEST_METHOD
```

- Encoded data also available as input ARG to CGI program
  - Still in varname=value&varname=value+value form

Obtaining Form Data Sent With GET - Webshare

- Use same method as described earlier for POST
  - Code would look the same
- REQUEST_METHOD server variable is located under the HTTPD global variable group

```
'CALLPIPE COMMAND GLOBALV SELECT HTTPD GET REQUEST_METHOD | VAR RMETHOD'

'CALLPIPE (name GetForm)',        /* Input:  name=value          */
  '  *:',                         /* Get input from sample form. */
  '| xlate 1-* 05 40',            /* Convert tabs to blanks.     */
  '| strip',                      /* Strip leading/trailing.     */
  '| locate 1',                  /* discard null lines, if any. */
  '| xlate fieldsep = f1 upper', /* Upper-case variable names.  */
  '| change //=PARM./',          /* ==> =PARM.name=value        */
  '| varload'
```

- Encoded data also available as input ARG to CGI program
  - Still in varname=value&varname=value+value form

Basic Flow Of CGI Script

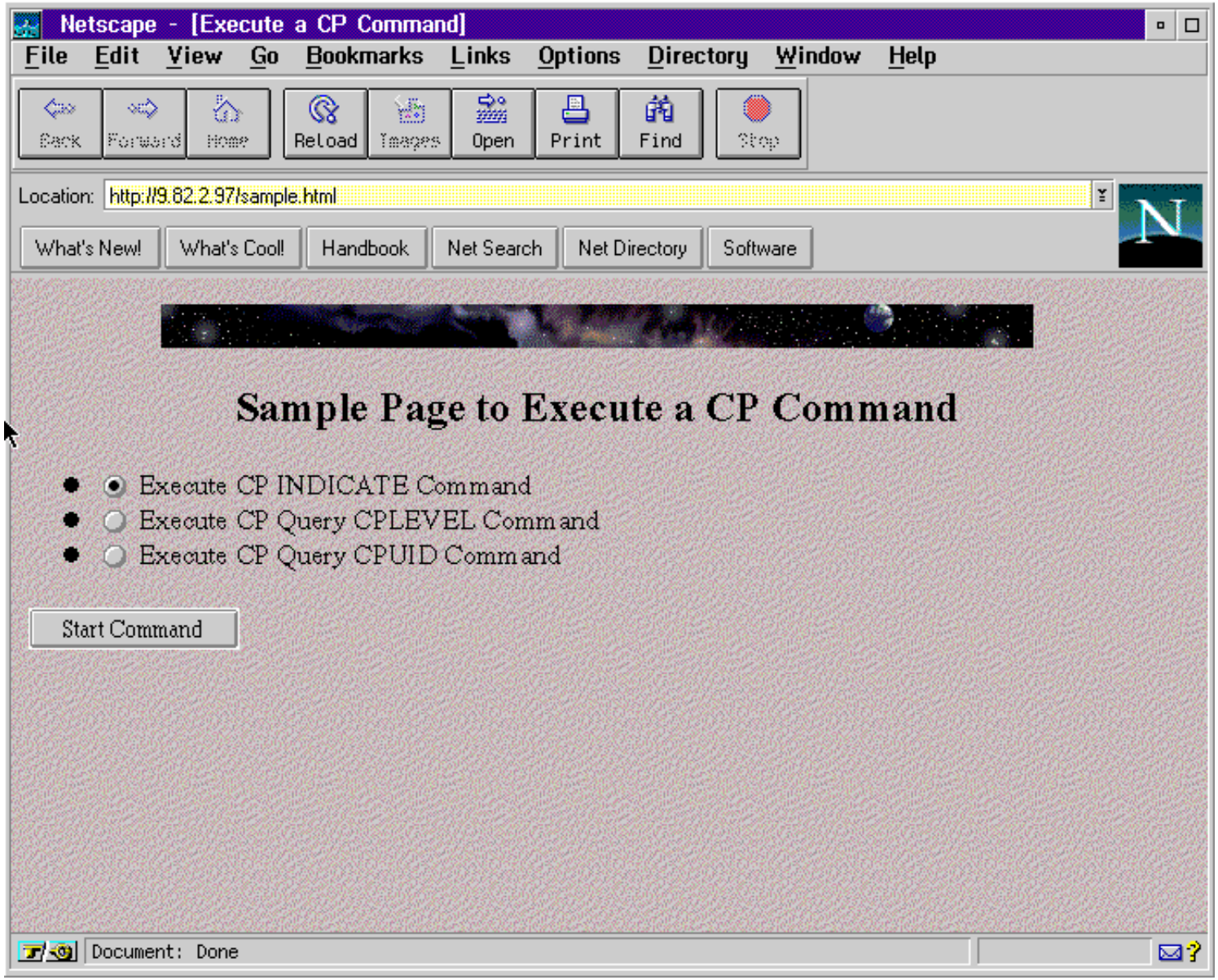
- Obtain any input
  - Command line input if encoded in URL
  - Message body input for FORM using POST method
- Perform processing algorithm
  - May involve
    - Calling other programs
    - Submitting queries to a database engine
    - Reading and writing files accessible by web server
- Send output back to browser
  - Send appropriate header information
  - Use appropriate techniques to maintain state information if needed

Lab Assignment 3

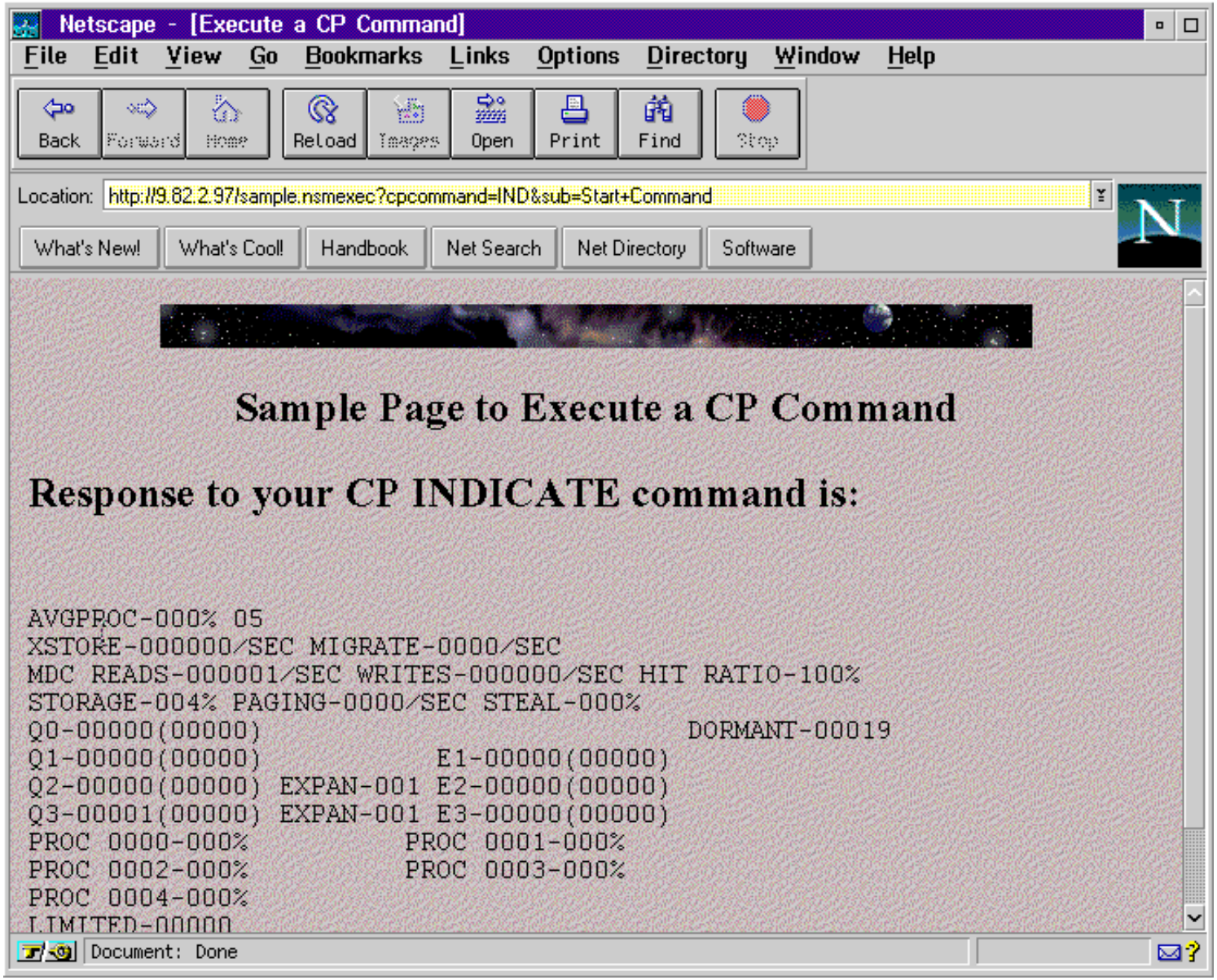
- Add the following functions to the CGI you created in lab 1:
  - Process the form data sent by the browser
  - Create and execute the requested CP or CMS command
  - Format the response and send back to the browser
  - Put a link in the response to the original form so that the user can request the submit form and send another request
- If you are done with this assignment early, and are feeling bored, try modifying the assignment such that:
  - Instead of a link back to the form, your CGI dynamically generates the form at the top of the response page so that the end user can simply scroll up to submit another command
  - Dynamically modify the selection box such that the highlighted choice is the command they previously selected
  - A hidden variable is created that contains the response from their previous command (so for example if the command is indicate they can see the current and previous response from indicate).

Break

Putting The Parts Together - A Sample CGI Script...



Putting The Parts Together - A Sample CGI Script...



Putting The Parts Together - A Sample CGI Script...


```
/*********************************************************************/
/*   Sample CGI Program - processes query command form               */
/*********************************************************************/

/*********************************************************************/
/*   Initialize some constants containing html                       */
/*********************************************************************/
init_out.0 = 10
init_out.1 = '<!doctype html public "-//IETF//DTD HTML 2.0//EN">'
init_out.2 = '<html><head>'
init_out.3 = '<title>Execute a CP Command</title> '
init_out.4 = '</head>'
init_out.5 = '<body background="page1bck.gif">'
init_out.6 = '<center>'
init_out.7 = '<img src="spacebar.gif" alt="*"><br>'
init_out.8 = '</center>'
init_out.9 = '<center><h2>Sample Page to Execute a CP Command</h2>',
             '</center>'
init_out.10 = '<p>'

end_out.0 = 2
end_out.1 = '</body>'
end_out.2 = '</html>'

output. = ''
```

Putting The Parts Together - A Sample CGI Script...


```
/*********************************************************************/
/*   Find out whether we are an EXEC or a stage in a pipeline      */
/*********************************************************************/

Address "COMMAND" 'GLOBALV SELECT HTTPD GET SERVER_SOFTWARE'
Parse SOURCE . . . . . env .
If env = 'CMS' Then
  Do
    Address 'COMMAND' 'NUCEXT CGI'
    vm_webserver = (rc = 0)
  End
```

Putting The Parts Together - A Sample CGI Script...


```
/*****************************************************************/
/*   Determine which server we are running under                 */
/*****************************************************************/


Select

  When SUBSTR(SERVER_SOFTWARE,1,5) = "Enter" Then
    Do
      Parse ARG args
      'StreamState INPUT 2'
      If rc=0 | rc=4 Then
        Do
         'Callpipe (name GetEwVars) *.input.2: | varload'
        End
      else
        Do
         'Output' "An error was detected.  No THREAD parameter was passed.",
                  "</body></html>"
         Exit 0
        End

      what_server = 'Beyond'
      pipe_command = 'callpipe'
    End
```

Putting The Parts Together - A Sample CGI Script...


```
/*****************************************************************/
/*   Determine which server we are running under                 */
/*****************************************************************/

  When SUBSTR(SERVER_SOFTWARE,1,4) = "Webs" Then
    Do
      Parse ARG args
      what_server = 'Webshare'
      pipe_command = 'callpipe'
    End

  Otherwise
    Do
      If vm_webserver Then
        Do
          what_server = 'Sterling'
          pipe_command = 'pipe'
          'CGI WRITE HEADER (STRING Status: 200 OK'
          header = 'Content_Type: text/html'
          'CGI WRITE HEADER (VAR HEADER'
        End
      Else
        Exit 12
    End
End
```

Putting The Parts Together - A Sample CGI Script...


```
/*****************************************************************/
/*   Do the real processing for this CGI program                 */
/*****************************************************************/

pipe_command 'stem init_out. | stem output.'
done = Produce_Output(what_server)
done = Read_Form_Data(what_server)
If done = 0 Then
  Do
    done = Process_Command(cmnd)
    done = Produce_Output(what_server)
    pipe_command 'stem end_out. | stem output.'
    done = Produce_Output(what_server)
  End

Exit 0
```

Putting The Parts Together - A Sample CGI Script...


```
/*********************************************************************/
/*   Subroutines                                                     */
/*********************************************************************/


/*********************************************************************/
/*   Write contents of output. stem to the server indicated          */
/*********************************************************************/

Produce_Output:
Parse ARG what_server

Select

  When what_server = 'Beyond' | what_server = 'Webshare' Then
    Do
      pipe_command 'STEM OUTPUT. | *:'
    End

  Otherwise
    Do
      'CGI WRITE DOCUMENT (TRANSLATE USENGLISH CRLF STEM OUTPUT.'
    End
End

output. = ''

Return 0
```

Putting The Parts Together - A Sample CGI Script...


```
/*****************************************************************/
/*   Read data sent from browser for method and server indicated   */
/*****************************************************************/
Read_Form_Data:
Parse ARG what_server
cmnd = ''
rc = 0

Select

  When what_server = 'Beyond' Then
    Do
      pipe_command '*: | EWGET vars p. ptr.'
      Do I = 1 to ptr.0
        name = ptr.I
        If name = 'P.CPCOMMAND' Then
          cmnd = VALUE(name'.1')
      End
    End
  When what_server = 'Sterling' Then
    Do
      'CGI GETVAR REQUEST_METHOD (VAR RMETHOD'
      If rmethod = 'POST' Then
        'CGI READ 1 (VAR BLOCK TRANSLATE USENGLISH'
      Else
        'CGI GETVAR QUERY_STRING (VAR BLOCK'
      If Length(BLOCK) <> 0 Then
        Do
          'CGI URLDECODE (VAR BLOCK INTO DECODED.'
          index = WORD(DECODED.0,WORDPOS('CPCOMMAND',DECODED.0))
          If index <> 0 Then
            cmnd = DECODED.index
        end
    End
```

Putting The Parts Together - A Sample CGI Script...


```
  When what_server = 'Webshare' Then
    Do
      pipe_command '(name GetForm)',
          '  *:',
          '| xlate 1-* 05 40',
          '| strip',
          '| locate 1',
          '| xlate fieldsep = f1 upper',
          '| change //=PARM./',
          '| varload'
      If PARM.cpcommand <> '' Then
        cmnd = PARM.cpcommand
    End
  Otherwise
    NOP
End

Select
  When cmnd = 'IND' Then
    cmnd = 'CP INDICATE'
  When cmnd = 'QCPL' Then
    cmnd = 'CP QUERY CPLEVEL'
  When cmnd = 'QCPID' Then
    cmnd = 'CP QUERY CPUID'
  Otherwise
    Do
      rc = 12
      cmnd = ''
    End
End

Return rc
```

Putting The Parts Together - A Sample CGI Script


```
/*****************************************************************/
/*   Run the specified command as a pipeline stage, placing result */
/*   into stem output.                                            */
/*****************************************************************/

Process_Command:
ARG cmnd

inconst.0 = 0
inconst.1 = '<h2> Response to your' cmnd 'command is: </h2><br>'
inconst.0 = inconst.0 + 1
inconst.2 = '<pre>'
inconst.0 = inconst.0 + 1

pipe_command '(endchar ?)',
   '     stem inconst.',
   '|a: fanin',
   '|    stem output.',
   '?',
         cmnd,
   '|    append literal </pre><br>',
   '|    a:'

Return rc
```

Wrap-up

Security Considerations

- CGI programs run in server virtual machine
  - Exceptions to this:
    - VM:Webgateway dynamic worker machines
    - Dedicated EnterpriseWeb/VM virtual machine
  - Run with full authority of server
    - Can harm system if server has CP privileges
    - Can harm server and data managed by server
- Restrict CGI programs
  - Keep them in few well defined directories
    - E.g. CGI-BIN directory
  - Only allow administrator to add or replace CGI programs
  - Dissallow CGI programs from running in user directories
  - Run in dynamic worker or dedicated EnterpriseWeb virtual machine
  - Ensure CGI does not return information to client that reveals file structure of system or other system related information
  - Ensure CGI programs carefully inspect client data that may cause commands to be executed

Performance Considerations

- CGI programs run in server virtual machines unless moved to dynamic worker or dedicated server
  - Can potentially block entire server with long running commands
  - Can consume excessive resources slowing other server functions
- Code CGI programs to:
  - Execute short running commands and avoid virtual machine blocking commands
  - Use a time-out mechanism for asynchronous operations
- CGI programs are restarted for each incoming request
  - Ensure fast initialization logic
- May want to code longer running CGI programs such that some information is delivered to client early, to avoid perception of hang
  - Block output into large block sizes

What's On The Horizon - Server APIs

- One problem with CGI programs is they must be started for each client reference
  - Performance impact to server
  - Difficult to maintain state information about client
- One solution to this problem is server application programming interfaces
  - Extend server's base functions
  - API is analogous to exit points
  - API programs run as part of server
    - Persistent across client requests
  - Typically faster than CGI programs
    - More complicated to code than standard CGI
    - Restricted to particular server brand
  - Examples
    - Netscape Netsite Server (NSAPI)
    - Microsoft IIS Web Server (ISAPI)

What's On The Horizon - JAVA Servlet Code

- Similar to API program without many of the drawbacks
  - Written in standard well known language
  - Portable, runs on Java virtual machine
  - Protected from doing harm by virtual machine boundaries
- Provides all of the positive features of API programs
  - Persistent execution
    - Avoid continuous startup cost of CGI programs
    - Easy to maintain client state information
  - May be faster than CGI programs
- Implemented by incorporating Java Servlet API and Classes into Web Server (Java Servlet Development Kit available to assist in doing this)
  - Applications can receive input data from server, and return output
  - Applications written with standard object set from Java

Summary

- HTTP protocol defines how browser interacts with server
  - Browser sends request header with header fields
    - Method field defines request server should act upon
  - Browser may send message or entity body with data
    - Depends on method being used
  - Processing of data sent by browser requires additional programs in server
    - Common Gateway Interface specification defines how server and CGI program pass data back and forth
    - CGI programs invoked by browser reference, to process request and input data
    - CGI programs respond to browser through server
  - Server sends response header with fields to browser
    - Status field indicates result of request processing to browser
    - HTML may follow response header as output to browser
- Creating CGI programs on VM is fun, and easy
  - Full power and capability of VM systems accessible from CGI program