CGI An Example

go to http://127.0.0.1/cgi-bin/hello.pl This causes the execution of the perl script hello.pl

Note: Although our examples use Perl, CGI scripts can be written in any language Perl, C, C++, VB, Python, SmallTalk, Assembly, Lisp etc...

CGI Model (Pieces)

- Clients: web browsers ie IE, Netscape
- Web Server (WS): Apache, Netscape Enterprise, IIS
- CGI Protocol: Specifying what a request/responce looks like
- Handler programs: Any executable residing on the web server

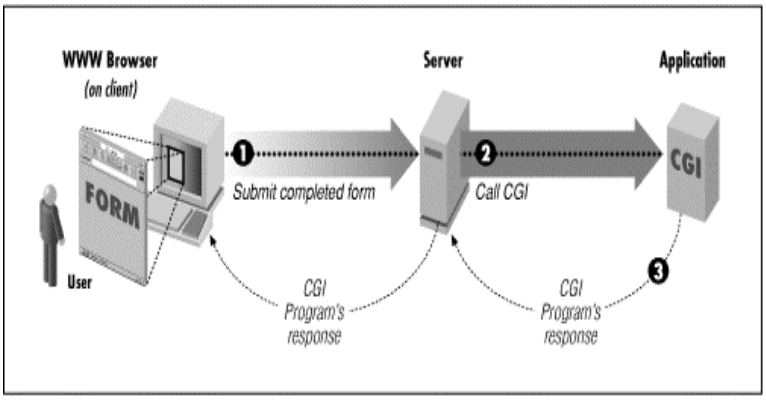
Interaction

- Client makes a request by specifying a URL+additional info.
- WS (in the URL) receives the request.
- WS identifies the request as a CGI request
- WS locates the program corresponding to the request.
- WS starts up the handling program (heavy weight process creation!!)
- WS feeds request parameters to handler (through stdin or environment variables).

Interaction (continued)

- Handler executes
- Output of the handler is sent via stdout back to the webserver for rerouting back to the requesting web browser.
- Output is typically a web page.
- Handler terminates.

Interaction (continued)



CGI

- Together the HTTP server and the CGI script are responsible for servicing a client request by sending back responses.
- The client request comprises
  - a Universal Resource Identifier (URI)
  - a request method
  - additional information about the request provided by the transport mechanism

CGI

- CGI defines the abstract parameters, known as metavariables, which describe the client's request. Together with a concrete programmer interface this specifies a platform-independent interface between the script and the HTTP server.

Script URI

- The Script-URI has the syntax of generic-RL as defined in section 2.1 of RFC 1808

`<scheme>://<host><port>/<path>?<query>`

- The various components of the Script-URI are defined by some of the metavariables (see Metavariables below);

Script URI (more detail)

```
script-uri = protocol "://" SERVER_NAME
  ":" SERVER_PORT enc-script enc-path-
  info "?" QUERY_STRING
```

where 'protocol' is obtained from
`SERVER_PROTOCOL`,

'enc-script' is a URL-encoded version of
`SCRIPT_NAME`

'enc-path-info' is a URL-encoded version of
`PATH_INFO`

Script URI (example)

```
script-uri = protocol "://" SERVER_NAME ":"
  SERVER_PORT enc-script enc-path-info "?"
  QUERY_STRING
```

`http://finance.yahoo.com/q?s=NT.TO&d=t`

Item	**Value**
protocol	http
SERVER_NAME	finance.yahoo.com
SERVER_PORT	not specified (default to 80)
enc-script	q
enc-path-info	not specified
QUERY_STRING	s=NT.TO&d=t

Data Input to the CGI Script

- Information about a request comes from
  - the request header
  - associated message-body.
- Servers MUST make portions of this information available to scripts.

Request Metadata (Metavariables)

```
AUTH_TYPE
CONTENT_LENGTH
CONTENT_TYPE
GATEWAY_INTERFACE
PATH_INFO
PATH_TRANSLATED
QUERY_STRING
REMOTE_ADDR
REMOTE_HOST
REMOTE_IDENT
REMOTE_USER
REQUEST_METHOD
SCRIPT_NAME
SERVER_NAME
SERVER_PORT
SERVER_PROTOCOL
SERVER_SOFTWARE
```

GET (part of http-spec)

- Default method for communicating query information to the script
- Simply specify the URL as above
- Don't need a form
- Everything after the ? in the URL appears in the `QUERY_STRING` environment variable
- Limited amount of information can be passed this way
  - URL may have a length restriction on the server
  - Environment variable may be restricted

GET (part of http-spec)

- You must do your own URL-Encoding (see below). URL-Encoding in this case is up to the web page designer and script writer. It is a good idea to conform to standards (see below).
- In forms
  - Can specify `method=get`
  - Form data will be URL-Encoded (see below) by the browser before sent to the server.
- `QUERY_STRING` is visible in the URL (at the browser) and appear in server logs (which are sometimes public).

POST (part of http-spec)

- In forms
  - Can specify `method=post`
  - Form data will be URL-Encoded (see below) by the browser before sent to the server.
- Can not be used from URL
- Form data appears in the scripts stdin (standard in)
- Can still populate `QUERY_STRING` using the URL
- Arbitrarily long form data can be communicated (some browsers may have limits (ie 7k)).
- Form data is not visible in the URL, usually does not appear in server logs.

URL-Encoding

- Standard way to send many name/value pairs in a single string (`QUERY_STRING` or `stdin`)
- Specified in RFC 2396 'Uniform Resource Identifiers (URI): Generic Syntax'
- Why encode?
  - Prevent confusion between CGI URL and HTML tags
  - Can think of a CGI script as a function, send arguments by specifying name/value pairs.
  - Forms consist of many elements, usually want all available to the script so need a way to pack and unpack them into a single string (`QUERY_STRING` or `stdin`)
  - Use a standard set of libraries to pack and unpack cgi arguments

Rules of URL-Encoding

- All submitted form data will be concatenated into a single string of ampersand (&) separated name=value pairs, one pair for each form tag. Like this:

  `form_tag_name_1=value_1&form_tag_name_2=value_2&...`
- Spaces in a name or value are replaced by a plus (+) sign. This is because url's cannot have spaces in them and under `METHOD=GET`, the form data is supplied in the query string in the url.
- Other characters (ie, =, &, +) are replaced by a percent sign (%) followed by the two-digit hexadecimal equivalent of the punctuation character in the Ascii character set.
  - Otherwise, it would be hard to distinguish these characters inside a form variable from those between the form variables in the first rule above.

Hello Example

Follow http://127.0.0.1/cgi-bin/hello.pl

Taking this one step at a time:

Using the command prompt, telnet to 127.0.0.1, port 80 and issue the following HTTP Get

```
get /cgi-bin/hello.pl http/1.0
```

This executes the script hello.pl. The transcript is here Notice the http header that comes back to the client!

Environment Example

Follow http://127.0.0.1/cgi-bin/environment.pl?var1=val1&var2=val2&var3=val3

Taking this one step at a time

telnet to 127.0.0.1, port 80 and issue the following HTTP Get

```
get /cgi-
  bin/environment.pl?var1=val1&var2=val2&var3=val3
  http/1.0
```

This executes the script environment.pl which prints its environment. The transcript is here Notice where the query variables end up?

Form Example

See form.html

Notes:

- Observe the url-encoded form variables in the GET form
- Observe stdin in the POST form
- Observe the hidden variables in both forms

Output from the CGI Script

- Standard output (stdout) is redirected to the webserver for relay to the client (browser)
- May or may not include a header
- Non-Parsed Header Output
  - Output not parsed by the web server
  - consists of a complete http response message.
- Parsed Header Output: Server creates a complete http response

  Consists of:

```
header      <-see below
            <-blank line separating header/body
body        <-message body (optionally null)
```

  where header consists of HTTP-fields (relayed to the client) as well as the additional CGI-Fields (interpreted by the server)

Parsed Header Output

Header	Explanation
`Content-type`	MIME Type
`Location`	specify to the server that the cript is returning a reference to a document. Causes the webserver to generate a redirect. A browser may choose to load the specified page.
`Status`	Becomes the status code in the servers responce message extension-header additional fields recognized by the server

Parsed Header Example

Content-type: text/html (source webPage.pl)

Content-type: image/jpeg (source getImage.pl)

Location http://127.0.0.1/cgi-bin/redirect.pl (source redirect.pl)

Note: This could have redirected to yahoo.com or any other URL.

Taking this one step at a time

telnet to 127.0.0.1, port 80 and issue the following HTTP Get

```
get /cgi-bin/redirect.pl http/1.0
```

The transcript is here