

CGI

"Web Application"

e.g. yahoo email, google, amazon

Partly, it's the old client/server structure -- interface on the client,
computation/storage centralized on the server.

Application expressed through a web interface

Web Platform

The web is a "platform" choice -- it's an attractive platform, since it works with
95% of computers

Microsoft has 90% marketshare, but that is divided among many OSes --

Win95, WinNT, 2000, XP ... -- the "web platform" is an attractive way for one
version to work for all those OSes.

Standards based platform -- HTTP, HTML -- not tied to any one vendor

Microsoft vs. Web Platform

Why Microsoft saw the web as terrifying -- if "applications" are coded to the
standard web platform, then Win32 has no special advantage vs. other
operating systems and so Win32 would have to compete with other operating
systems on price and features.

Note the pattern: standards lead to competition -- they allow one vendor can be
replaced with another. Put another way: standards avoid "vendor lock-in"

Therefore Microsoft wanted to control the browser market -- bundled IE with the
OS, ... this led to their conviction of illegal monopoly behavior. It is not legal in
the U.S. to use monopoly strength in one market (OSes) to take over another
market (browsers).

Server 1: Static Pages

HTTP server sits there

Request comes in

Typically HTTP server maps that request into its file system

The server decides what the MIME type is -- probably based on its file extension.

This "file system" model is sometimes called "static pages"

Serving static pages is easy -- bottleneck is inevitably the network, not the CPU
or file system

Server 2: Dynamic Pages

Suppose the server is has a database of books

When a request comes in like "/the+perfect+storm", or "/geography"

1. Do some sort of lookup using the words from the request
2. Dynamically generate a page containing the data that matched

How the server interprets the request is up to the server

Point: server can do a computation with a centralized resource, and then send results back to the client. The interesting part of the computation runs on the server.

CGI

Common Gateway Interface -- a standard which interfaces the HTTP server software with CGI programs which run on the server.

The CGI standard defines how the server communicates all the various facts about the requesting URL (date, referer, accept) to the CGI program, and how the CGI program's output should be handled and sent back to the requesting client.

CGI is language independent: C, Perl, Unix shell script.

Security

Security concerns -- because the CGI's run on the server, they need to take care not to allow a rogue client to compromise the server either by hurting the server or serving up information which was supposed to be secret.

CGI Dialog

1. URL

The client has a URL as usual. However, the URL identifies not a page of HTML, but a CGI program on the server.

e.g. <http://www-cs-faculty.stanford.edu/cgi-bin/nick/hello.pl>

2. Client GET

The client does a GET request as usual

3. Server -> CGI

The server looks at the GET request, realizes its a CGI.

4. Run CGI

Server runs the CGI program, feeding it the GET request as input. The GET information is fed to the CGI through environment variables.

Alternately, if the client does a POST, then the data is fed to the CGI via its standard input.

5. CGI program

The CGI looks at the input and computes what it wants to compute.

6. HTTP Header

First, the CGI prints the HTTP header fields it wants, followed by a blank line.

7. HTML content

Then the CGI prints the body content it wants and then exits

8. HTTP Server

The HTTP server gathers the output of the CGI, adds more fields to the header, and sends it along to the client.

Trivial Example — hello.pl

```
#!/usr/bin/perl -w

## Hello.pl -- demonstrate a trivial CGI that prints
## out some HTML and the current time on this server.

use strict 'vars';

my($EOL) = "\015\012";

## This is a human-readable str of the current time
my($nowStr);
$nowStr = localtime();

## This line must be included in the header
print "Content-type: text/html$EOL$EOL";

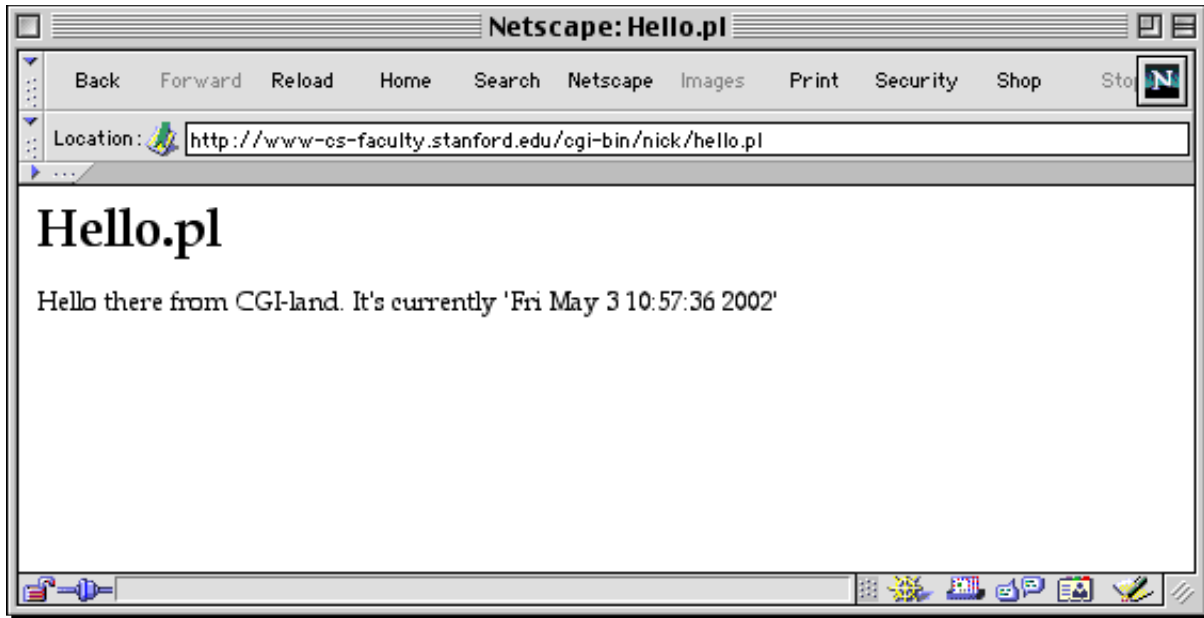
## Write out the HTML content
print "<html><head><title>Hello.pl</title></head>\n";
print "<body bgcolor=white>\n";
print "<h1>Hello.pl</h1>\n";
print "Hello there from CGI-land. It's currently '$nowStr'\n";

print "</body></html>\n";
```

hello.pl CGI Process

This CGI doesn't look at any input, it just produces an HTTP header followed by the same HTML body every time.

For its HTTP header, it just writes the "content-type: text/html" line followed by a blank line. The HTTP server fixes up the rest of the header on its way out to the client.



Input to CGI programs

Before the CGI is run, its environment variables are set to communicate the request — the primitive Unix method for programs to communicate with each other.

Most likely, the CGI runs as a user called "WWW" or "CGI" which has limited read/write permission on the system. This helps limit some of the security danger, even if the CGI is poorly written.

GET Method

With the "GET" method, the client does a regular GET request to the server, and the data for the CGI is after the "?" in the request URL

The QUERY_STRING environment variable will be the text after the ? in the URL. There are many other environment variables set to indicate other things about the request -- see the DumpEnv example below.

POST Method

The client sends the data on the socket after the blank line

The web server sends the data to the CGI on the CGI's standard input

DumpEnv Perl Script

```
#!/usr/bin/perl
# Print out the values of all the environment variables
# in an HTML <ul>.
# Call from the shell or invoke as a CGI script.

# HTTP header section
print "content-type: text/html\r\n\r\n";

$header = <<EOT;
<html>
<head><title>DumpEnv</title></head>
<body bgcolor=white>
EOT

$trailer = <<EOT;
</body>
</html>
EOT

# Emit an HTML <ul> for all the environment vars
# set up for the CGI
print $header

print "<ul>\n";
## iterate over the keys, but sort them first
foreach $key (sort (keys %ENV)) {
    print "<li><b>$key</b> = $ENV{$key}\n";
}
print "</ul>\n";
print $trailer;
```

Environment Vars

Query_String (after the ? in the URL)

Script_Name (eg /cgi-bin/nick/dumpenv.pl)

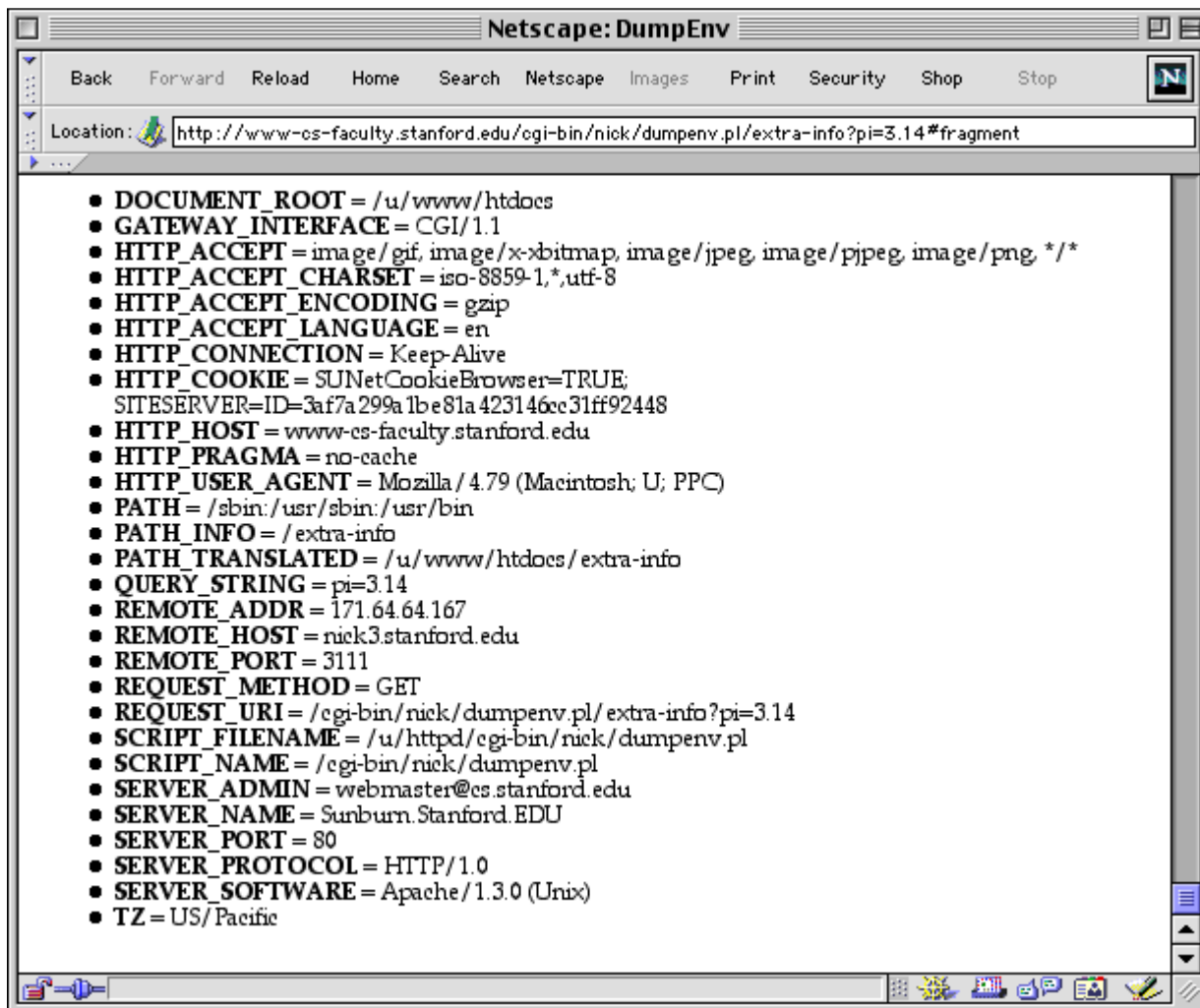
Request_Method (usually GET)

Path_Info (extra path after the name of the CGI in the URL)

Remote_Host -- The client's IP or DNS address -- from a privacy point of view, this is one thing the server will definitely know -- how else will it send the packets back to you but by knowing your IP addr?

DumpEnv Output

<http://www-cs-faculty.stanford.edu/cgi-bin/nick/dumpenv.pl/extra-info?pi=3.14#fragment>



HTML Forms

Present choices to the user in web page

action=url -- specifies the URL of the CGI that gets the data

<input name="a-name" ... > -- maps to a form element on screen

Submit -- send the data in the form to the CGI -- name1=value1&name2=value2...
 encoded. Uses URL %xx encoding for funny chars, uses =/& syntax

Trivial Form Example

```
<html>

<head>
<title>Form1</title>
</head>

<body bgcolor=white>

<!-- form tag -->

<h1>Form1</h1>

<form
  action=http://www-cs-faculty.stanford.edu/cgi-bin/nick/dumpenv.pl
  method=get>

<!-- text input -->
<p>
First name:
<input type=text name = "first-name" size = 40 value="Bob">

<!-- submit button -->
<p>
<input type=submit name=go value="Submit Request">

</form>

</body>
</html>
```

