CGI ENVIRONMENT VARIABLES

The original specification for the Common Gateway Interface (CGI) established a list of environment variables, or aspects of the server and of the user's connection that would always be made known to CGI scripts. These variables remain a basic part of writing CGI scripts and have also been adopted in other server-side scripting languages, including server-side includes, ASP, and PHP.

The complete CGI specification, http://hoohoo.ncsa.uiuc.edu/cgi/interface.html

This list of the CGI environment variables below is annotated.

SERVER_SOFTWARE

The name and complete version information of the Web server software. Some server-side software adds itself to SERVER_SOFTWARE string. For example, Apache/1.3.26 (Unix) PHP/4.2.3.

SERVER_NAME

The server's domain name or IP address.

GATEWAY_INTERFACE

The version of the CGI specification in use. For example, CGI/1.1.

CGI scripts are executable programs that can also be run as command-line programs on the server. The script can use the presence of a GATEWAY_INTERFACE environment variable to determine that it is being run as a CGI script.

SERVER_PROTOCOL

The communications protocol and version number used to request the script. For example, HTTP/1.1.

SERVER_PORT

The port number to which the request was sent. Sites running Web services on ports other than the default of 80 may want scripts to behave differently for users on different ports.

REQUEST_METHOD

The HTTP method used for the request. For example, GET or POST.

A common use of this variable is have one script handle both the job of displaying a form to the user and of handling the form's submission. If the REQUEST_METHOD is GET, the script writes out a form with a method attribute of POST and an action of its own SCRIPT_NAME (see below). If the REQUEST_METHOD is POST it processes the form submission and then either displays results or redirects the user to the appropriate exit page.

PATH_INFO

Directory path information added to the name of a script. If a request is made as [script name]/alpha/beta.pdf the PATH_INFO is /alpha/beta.pdf. This may or may not refer to an actual alpha subdirectory under the script's location, with an actual beta.pdf file in it.

Using PATH_INFO allows scripts to sit in the middle of a URI, making transparent to users that a script is even at work. This transparency makes possible upgrading services from collections of HTML documents to database-driven dynamic services. For example, a site may maintain a small set of help documents with URIs such as /help-docs/keyword-searching.html or /help-docs/

sorting-results.html. When the help documents grow in number, or need to be customized on the fly, the help-docs directory can be replaced by a script named help-docs that customizes its output depending on the PATH_INFO value. The actual URL for the individual help documents does not change.

PATH_TRANSLATED

In cases where a PATH_INFO value corresponds to an actual file path on the server, it will be relative to the server's document root. PATH_TRANSLATED shows the physical directory path on the server.

SCRIPT_NAME

Uniform Resource Identifier (URI) is the part of a URL that comes after http://server.somewhere.org/ and is the part that the server actually receives as the user's request.

The URI of the script, up to but not including any question mark. So a request of /ejournals/titles/alpha-sort.cgi?letter=J has a SCRIPT_NAME of /ejournals/titles/alpha-sort.cgi. Scripts that write out HTML with links back to themselves can use SCRIPT_NAME instead of a hard-wired value; if these scripts are ever moved to different locations, the links would not break.

QUERY_STRING

The part of a URI coming after any question mark. In /ejournals/titles/alpha-sort.cgi?letter=J the QUERY_STRING is letter=J.

For forms submitted by the GET method, the CGI specification and most scripting languages expect the QUERY_STRING to provide the form's input names and their values in the syntax ?name1=value1&name2=value2….

AUTH_TYPE

If the request was made with a type of user authentication, AUTH_TYPE shows the authentication system used. In typical Web practice, the authentication system almost always is Basic. Apache and recent versions of IIS also support a standard called Digest authentication, which encrypts passwords, and Basic authentication sends passwords in the clear making it more secure. Unfortunately, browser support has been poor, so Basic authentication remains the nearly universal system for username/password access on the Web.

REMOTE_USER

If the request was made with user authentication, this variable will contain the username provided. For security reasons, the user's password is not available to scripts.

REMOTE_IDENT

A user identity provided by the authentication system described in RFC 1413. This variable is not widely used.

CONTENT_TYPE

For queries with a REQUEST_METHOD that supplies incoming data, this variable contains the MIME content type of that data. For example, form data being submitted with a POST method typically has a content type of application/x-www-form-urlencoded.

CONTENT_LENGTH

For queries with a REQUEST_METHOD that supplies incoming data, this variable is the length of that data in bytes. The CONTENT_LENGTH value can be compared with the actual length of data received to confirm that the submission was completely received.

In addition to these variables from the official CGI specification, servers may make other variables available to scripts. These variable below are some of

the most common, but their availability and the availability of still other variables depends not only on the server involved, but the version. Server documentation provides a complete list.

REMOTE_ADDR

The user's IP address, as seen by the server. This variable may be the address of a proxy, gateway, or firewall making the request on behalf of the user, whose actual IP address remains unknown. Available in Apache and IIS.

URL / REQUEST_URI

The requested URL, up to but not including any question mark. This variable is often the same as SCRIPT_NAME. Available as URL in IIS and as REQUEST_URI in Apache.

PATH

The list of directories on the server that are automatically searched when a script requests another program be run.

SERVER_ADDR

The server's IP address. Available in Apache.

SERVER_NAME

The server's domain name. Available in Apache and IIS; if the server does not have a domain name, IIS assigns the IP address to SERVER_NAME.

SERVER_ADMIN

The e-mail address of the Web server administrator, as assigned in the server configuration files. Available in Apache.

The CGI specification also requires that a server make all HTTP headers sent by the user available to scripts. The environment variables are given the name HTTP_[Header Name], with the header name converted to all capital letters and hyphens changed to underscores. For example, the value of the User-Agent header is HTTP_USER_AGENT.

Several HTTP headers values are commonly used by scripts:

HTTP_ACCEPT

The Accept header is intended as a way for the browser to send a prioritized list of MIME types it will accept. In practice, it is seldom used, but it could be used to determine whether to output images in PNG or GIF format, for example.

HTTP_ACCEPT_LANGUAGE

The Accept-Language header provides a list of preferred languages, using the ISO's standard abbreviations for languages and dialects. Scripts on a bilingual site could default to English if the first language listed is "en" and to Spanish if the first language is "es." Web managers should be aware that users may not always configure their browsers to send an accurate set of language preferences, so any default taken from Accept-Language should allow the user to select other available languages in the interface.

HTTP_COOKIE

Most scripting languages have simplified access to cookie values, but ultimately they all come from what the browser sends in its Cookie header. Web managers should be aware that browsers and third-party software increasingly give users control over which cookies to accept, keep, and send back to

servers. In addition, sufficiently motivated hackers can send any cookies and values they choose, so using cookies to determine whether a user may access a resource is poor security.

HTTP_REFERER

The Referer header provides the URL of the page from which the user followed a link to the current page. When the user follows a bookmark or enters a page's URL manually, the HTTP specification requires that the no referer value be sent. Some browsers can be configured not to send Referer headers at all, and as with cookies, a determined hacker can send an arbitrary value, so security based on the Referer value is weak.

In the mid-1990s, the practice of browser sniffing by looking for specific User-Agent values was so entrenched that many pages failed to work if the User-Agent did not appear to be Netscape 3 by including that browser's internal name, Mozilla/3.0. To prevent being excluded from these pages, Microsoft released Internet Explorer 3.0 with a User-Agent string that included Mozilla/3.0 (Compatible). When Explorer's Version 4 was released shortly after Netscape 4, its User-Agent string included Mozilla/4.0 (Compatible). This string has remained in Explorer to the present.

More recently, the open-source Mozilla 1.0 browser chose a User-Agent string of "Mozilla 5.0" to prevent it being confused with Netscape 1.0. This string is included in other browsers built on Mozilla, including current versions of Netscape.

So Internet Explorer 6.0 tacitly claims to be compatible with Mozilla 4.0 and Netscape 7 claims to be Mozilla 5.0.

HTTP_USER_AGENT

A user agent is software that makes HTTP requests on behalf of a user. In most cases the user agent is a browser, but it also can be a proxy, firewall, or software to collect pages automatically for offline browsing. The User-Agent header provides a way for this software to identify itself and its version, often with other information to identify the user's operating system and in some cases claims of compatibility with other user agents.

Over the years, many scripts have customized their output for different browsers based on the User-Agent value. Yet again, determined users could configure their browsers to send a misleading User-Agent value or no value at all; many users also are behind proxy servers that send their own User-Agent value. So scripts cannot reliably assume they know the user's browser and must make any browser-specific customizations with caution.

PARSING LOG FILES WITH PERL

Web server log files are lists of predictably formatted lines of text. For needs that are more specific than the reporting abilities of log analyzer programs can support, Web managers often turn to system commands or short scripts to find exactly what they are looking for. Tools such as the Grep command in Unix and Linux, or the command-line Find command in Windows can select and print out log file lines based on relatively basic criteria.

Other commands, text editors, and scripting languages provide greater or lesser capabilities for finding text, but many developers rely on the Perl scripting language as the best tool available.

Perl is a complete programming language and this example is not offered as a tutorial. Perl also provides multiple ways to do the same thing, so many Web managers with Perl experience already have their own methods for parsing log files.

The example below converts each line of an Apache-combined format log file into a set of variable names, which can then be examined or manipulated as needed. This particular example looks for any lines in the log file with status codes corresponding to errors and displays a brief report showing the page requested, the error status, and the URL of the referring page that provided the link (unless the referer was logged with the blank value of "-").

The only real challenge Perl faces in parsing the combined log format is that, although its fields are separated by spaces, fields such as the HTTP request and User Agent contain spaces inside quotes. Simply splitting the line at each space cannot predictably keep all the fields together.

A more reliable method is to use the quotewords routine that found in Perl's Text::Parse module, which is designed to do exactly what is needed here: split a string of text at a certain character, or pattern of characters, unless that pattern is inside quotes.

This example uses the Unix "#!" syntax for specifying the Perl program on the first line. Windows users will typically need to run this script as "D:\perl\perl.exe [scriptname.pl]."

```
#!/usr/bin/perl

# log-parser.pl
# One of many ways to parse lines from the combined
# logfile format, such as:

# staff-pc-118.foo.org - - [11/Nov/2002:18:33:17 –0500]
# "GET /info/ill-policy.html HTTP/1.1" 200 14316
# "http://www.ourlibrary.org/circulation.html"
# "Opera 6.05"


# To use this script, enter "log-parser.pl [log file]"
# use the Text::Parsewords module to get the quotewords
# routine for intelligently handling quoted values
use Text::ParseWords;

while (<>) {
```

```
 chop;
 # Separate the line into the following values using
 # whitespace (\s+) that is not inside quotes
 ($remote_host, $ident, $remote_user, $time, $time_zone,
$request,
 $status_code, $bytes_sent, $referrer, $user_agent) =
 &quotewords('\s+', 0, $_);

 # Remove the square brackets from the date and time zone
 $time =~ s/^\[//;
 $time_zone =~ s/\]$//;

 # The following lines will vary depending and specific
 # needs. This version report only on lines with 4xx and
 # 5xx status codes.
 if ($status_code >= 400) {
 print "$request\n";
 print "Returns status code $status_code\n";

 # If there is a real value for the referrer, print it.
 if ($referrer ne "-") {
 print "Referring page was $referrer\n";
 } else {
 print "[No referring page was logged]\n";
 }

 print "\n";
 }
}
```

BIBLIOGRAPHY

Anonymous. 2002. *Maximum Apache Security.* Indianapolis: Sams Publishing.

Architectural and Transportation Barriers Compliance Board. *Electronic and Information Technology Accessibility Standards,* Dec. 21, 2000. Available from www.access-board.gov/sec508/508standards.htm.

Badre, Albert N. 2002. *Shaping Web Usability: Interaction Design in Context.* Boston: Addison-Wesley.

Bahadur, Gary, Mike Shema. 2001. Improving Apache. *Information Security Magazine,* April 2001. Available from www.infosecuritymag.com/articles/april01/features1_web_server_sec.shtml.

Baron, David. *Mozilla's DOCTYPE Sniffing,* July 10, 2002. Available from www.mozilla.org/docs/web-developer/quirks/doctypes.html.

Battleson, Brenda, Austin Booth, and Jane Weintrop. 2001. Usability Testing of an Academic Library Web Site: A Case Study. *Journal of Academic Librarianship* 27, no. 3: 188-198.

Bos, Bert, Håkon W. Lie, Chris Lilley, and Ian Jacobs. *Cascading Style Sheets, Level 2.* May 12, 1998. Available from www.w3.org/TR/REC-CSS2.

Boutin, Paul. *Web Standards for Hard Times,* August 6, 2002. Available from http://hotwired.lycos.com/webmonkey/02/33/index1a.html?tw=authoring.

Byerly, Suzanne L., Mary B. Chambers. 2002. Accessibility and Usability of Web-Based Library Databases for Non-Visual Users. *Library Hi Tech* 20, no. 2: 169-178.

Chisolm, Wendy, Gregg Vanderheiden, and Ian J. Jacobs. *Web Content Accessibility Guidelines 1.0,* May,5, 1999. Available from www.w3.org/TR/1999/WAI-WEBCONTENT-19990505.

Computer Incident Advisory Capability. *U.S. DOE-CIAC Bulletins: Apache Software Foundation,* Oct. 7, 2002. Available from www.ciac.org/ciac/bulletinsByType/vndr_apache_bulletins.html.

Computer Incident Advisory Capability. *U.S. DOE-CIAC Bulletins: Microsoft Corporation,* Nov. 1, 2002. Available from www.ciac.org/ciac/bulletinsByType/vndr_ms_bulletins.html.

Dickstein, Ruth, Vicki Mills. 2000. Usability Testing at the University of Arizona Library: How the Let the Users in on the Design. *Information Technology and Libraries* 19, no. 3: 144-151.

DiNicolo, Dan. *A Viable IIS Alternative? Apache 2.0 on Windows 2000.* 2002. Available from www.serverwatch.com/tutorials/article.php/10825_1474251.

Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol - HTTP/1.1 (RFC2616).* 1999. Available from ftp://ftp.isi.edu/in-notes/rfc2626.txt.

Guenther, Kim. 2002. Web Site Management. *Online* 26, no. 3: 82.

Hudson, Laura. 2000. Radical Usability (Or, Why You Need To Stop Redesigning Your Web Site). *Library Computing* 19, no. 1/2: 86-92.

Krug, Steve. 2000. *Don't Make Me Think.* Indianapolis: Que.

Lie, Håkon W., Bert Bos. *Cascading Style Sheets, Level 1,* Jan. 11, 1999. Available from www.w3.org/TR/REC-CSS1.

McMullen, Susan. 2001. Usability Testing in a Library Web Site Redesign Project. *Reference Services Review* 29, no. 1: 7-22.

Nielsen, Jakob. *Top Ten Guidelines for Homepage Usability.* 2002. Available from www.useit.com/alertbox/20020512.html.

Nielsen, Jakob. 2000. *Designing Web Usability: The Practice of Simplicity.* Indianapolis: New Riders.

Norlin, Elaina, and CM! Winters. 2002. *Usability Testing for Library Web Sites; A Hands-On Guide.* Chicago and London: American Library Association.

O'Brien, Gerry. 2000. *Microsoft IIS 5 Administration: The Authoritative Solution.* Indianapolis: Sams Publishing.

Raggett, Dave, Arnaud Le Hors, and Ian Jacobs. *HTML 4.01 Specification.* Dec. 24, 1999. Available from www.w3.org/TR/html4.

Silver, Lance. *CSS Enhancements in Internet Explorer 6,* March 2001. Available from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnie60/html/cssenhancements.asp.

Technical Overview of Internet Information Services (IIS) 6.0. July 2002. Available from www.microsoft.com/windows.netserver/docs/IISOverview.doc.

Thatcher, Jim, Paul Bohman, Michael Burks, Shawn L. Henry, Bob Regan, Sarah Swierenga, Mark D. Urban, and Cynthia D. Waddell. 2002. *Constructing Accessible Web Sites.* Birmingham, U.K.: Glasshaus.

W3C DOM Working Group. *Document Object Model FAQ.* 2001. Available from www.w3.org/DOM/faq.html.

W3C HTML Working Group. *XHTML 1.0: The Extensible HyperText Markup Language (Second Edition).* August 1, 2002. Available from www.w3.org/TR/xhtml1.

Web Standards Project. *Dreamweaver Task Force.* 2002. Available from www.webstandards.org/act/campaign/dwtf.

Zaner, John. 2001. Now That You've Made Your Website, Where Will You Host It? *Tech Directions* 60, no. 7: 16.