The CGI Specification

This is the specification for CGI version 1.1, or CGI/1.1. Further revisions of this protocol are guaranteed to be backward compatible.

The server and the CGI script communicate in four major ways. Each of the following is a hotlink to graphic detail.

CGI Environment Variables

In order to pass data about the information request from the server to the script, the server uses command line arguments as well as environment variables. These environment variables are set when the server executes the gateway program.

Specification

The following environment variables are not request-specific and are set for all requests:

- `SERVER_SOFTWARE`

  The name and version of the information server software answering the request (and running the gateway). Format: name/version

- `SERVER_NAME`

  The server's hostname, DNS alias, or IP address as it would appear in self-referencing URLs.

- `GATEWAY_INTERFACE`

  The revision of the CGI specification to which this server complies. Format: CGI/revision

The following environment variables are specific to the request being fulfilled by the gateway program:

- `SERVER_PROTOCOL`

  The name and revision of the information protcol this request came in with. Format: protocol/revision

- `SERVER_PORT`

  The port number to which the request was sent.

- `REQUEST_METHOD`

  The method with which the request was made. For HTTP, this is "GET", "HEAD", "POST", etc.

- `PATH_INFO`

  The extra path information, as given by the client. In other words, scripts can be accessed by their virtual pathname, followed by extra information at the end of this path. The extra information is sent as PATH_INFO. This information should be decoded by the server if it comes from a URL before it is passed to the CGI script.

- `PATH_TRANSLATED`

  The server provides a translated version of PATH_INFO, which takes the path and does any virtual-to-physical mapping to it.

- `SCRIPT_NAME`

  A virtual path to the script being executed, used for self-referencing URLs.

- `QUERY_STRING`

  The information which follows the ? in the URL which referenced this script. This is the query information. It should not be decoded in any fashion. This variable should always be set when there is query information, regardless of command line decoding.

- `REMOTE_HOST`

  The hostname making the request. If the server does not have this information, it should set REMOTE_ADDR and leave this unset.

- `REMOTE_ADDR`

  The IP address of the remote host making the request.

- `AUTH_TYPE`

  If the server supports user authentication, and the script is protects, this is the protocol-specific authentication method used to validate the user.

- `REMOTE_USER`

  If the server supports user authentication, and the script is protected, this is the username they have authenticated as.

- `REMOTE_IDENT`

  If the HTTP server supports RFC 931 identification, then this variable will be set to the remote user name retrieved from the server. Usage of this variable should be limited to logging only.

- `CONTENT_TYPE`

  For queries which have attached information, such as HTTP POST and PUT, this is the content type of the data.

- `CONTENT_LENGTH`

  The length of the said content as given by the client.

In addition to these, the header lines received from the client, if any, are placed into the environment with the prefix HTTP_ followed by the header name. Any - characters in the header name are changed to _ characters. The server may exclude any headers which it has already processed, such as Authorization, Content-type, and Content-length. If necessary, the server may choose to exclude any or all of these headers if including them would exceed any system environment limits.

An example of this is the HTTP_ACCEPT variable which was defined in CGI/1.0. Another example is the header User-Agent.

- `HTTP_ACCEPT`

  The MIME types which the client will accept, as given by HTTP headers. Other protocols may need to get this information from elsewhere. Each item in this list should be separated by commas as per the HTTP spec.

  Format: type/subtype, type/subtype

- `HTTP_USER_AGENT`

  The browser the client is using to send the request. General format: `software/version library/version`.

Examples

Examples of the setting of environment variables are really much better demonstrated than explained.

CGI Command line options

Specification

The command line is only used in the case of an ISINDEX query. It is not used in the case of an HTML form or any as yet undefined query type. The server should search the query information (the `QUERY_STRING` environment variable) for a non-encoded = character to determine if the command line is to be used, if it finds one, the command line is not to be

used. This trusts the clients to encode the = sign in ISINDEX queries, a practice which was considered safe at the time of the design of this specification.

For example, use the finger script and the ISINDEX interface to look up "httpd". You will see that the script will call itself with `/cgi-bin/finger?httpd` and will actually execute "finger httpd" on the command line and output the results to you.

If the server does find a "=" in the `QUERY_STRING`, then the command line will not be used, and no decoding will be performed. The query then remains intact for processing by an appropriate FORM submission decoder. Again, as an example, use this hyperlink to submit `"httpd=name"` to the finger script. Since this `QUERY_STRING` contained an unencoded "=", nothing was decoded, the script didn't know it was being submitted a valid query, and just gave you the default finger form.

If the server finds that it cannot send the string due to internal limitations (such as exec() or /bin/sh command line restrictions) the server should include NO command line information and provide the non-decoded query information in the environment variable `QUERY_STRING`.

Examples

Examples of the command line usage are much better demonstrated than explained. For these examples, pay close attention to the script output which says what argc and argv are.

CGI Script Input

Specification

For requests which have information attached after the header, such as HTTP POST or PUT, the information will be sent to the script on stdin.

The server will send CONTENT_LENGTH bytes on this file descriptor. Remember that it will give the CONTENT_TYPE of the data as well. The server is in no way obligated to send end-of-file after the script reads `CONTENT_LENGTH` bytes.

Example

Let's take a form with METHOD="POST" as an example. Let's say the form results are 7 bytes encoded, and look like `a=b&b=c`.

In this case, the server will set CONTENT_LENGTH to 7 and CONTENT_TYPE to application/x-www-form-urlencoded. The first byte on the script's standard input will be "a", followed by the rest of the encoded string.

CGI Script Output

Script output

The script sends its output to stdout. This output can either be a document generated by the script, or instructions to the server for retrieving the desired output.

Script naming conventions

Normally, scripts produce output which is interpreted and sent back to the client. An advantage of this is that the scripts do not need to send a full HTTP/1.0 header for every request.

Some scripts may want to avoid the extra overhead of the server parsing their output, and talk directly to the client. In order to distinguish these scripts from the other scripts, CGI requires that the script name begins with nph- if a script does not want the server to parse its header. In this case, it is the script's responsibility to return a valid HTTP/1.0 (or HTTP/0.9) response to the client.

Parsed headers

The output of scripts begins with a small header. This header consists of text lines, in the same format as an HTTP header, terminated by a blank line (a line with only a linefeed or CR/LF).

Any headers which are not server directives are sent directly back to the client. Currently, this specification defines three server directives:

- `Content-type`

  This is the MIME type of the document you are returning.

- `Location`

  This is used to specify to the server that you are returning a reference to a document rather than an actual document.

  If the argument to this is a URL, the server will issue a redirect to the client.

  If the argument to this is a virtual path, the server will retrieve the document specified as if the client had requested that document originally. ? directives will work in here, but # directives must be redirected back to the client.

- `Status`

  This is used to give the server an HTTP/1.0 status line to send to the client. The format is `nnn xxxxx`, where `nnn` is the 3-digit status code, and `xxxxx` is the reason string, such as "Forbidden".

Examples

Let's say I have a fromgratz to HTML converter. When my converter is finished with its work, it will output the following on stdout (note that the lines beginning and ending with --- are just for illustration and would not be output):

```
--- start of output ---
Content-type: text/html

--- end of output ---
```

Note the blank line after Content-type.

Now, let's say I have a script which, in certain instances, wants to return the document `/path/doc.txt` from this server just as if the user had actually requested `http://server:port/path/doc.txt` to begin with. In this case, the script would output:

```
--- start of output ---
Location: /path/doc.txt

--- end of output ---
```

The server would then perform the request and send it to the client.

Let's say that I have a script which wants to reference our gopher server. In this case, if the script wanted to refer the user to `gopher://gopher.ncsa.uiuc.edu/`, it would output:

```
--- start of output ---
Location: gopher://gopher.ncsa.uiuc.edu/

--- end of output ---
```

Finally, I have a script which wants to talk to the client directly. In this case, if the script is referenced with `SERVER_PROTOCOL` of HTTP/1.0, the script would output the following HTTP/1.0 response:

```
--- start of output ---
HTTP/1.0 200 OK
Server: NCSA/1.0a6
Content-type: text/plain

This is a plaintext document generated on the fly just for you.

--- end of output ---
```
