INTERNET-DRAFT
<draft-robinson-www-interface-01.ps>
Expires 15 August 1996

D.R.T. Robinson
University of Cambridge
15 February 1996

The WWW Common Gateway Interface Version 1.1

Status of this memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as 'work in progress'.

To learn the current status of any Internet-Draft, please check the '`1id-abstracts.txt`' listing contained in the Internet-Drafts Shadow Directories on `ftp.is.co.za` (Africa), `nic.nordu.net` (Europe), `munnari.oz.au` (Pacific Rim), `ds.internic.net` (US East Coast), or `ftp.isi.edu` (US West Coast).

Distribution of this document is unlimited. Please send comments to the author; general discussion about CGI should take place on the <`www-talk@w3.org`> mailing list.

Abstract

The Common Gateway Interface (CGI) is a simple interface for running external programs, software or gateways under an information server in a platform-independent manner. Currently, the supported information servers are HTTP servers.

The interface has been in use by the World-Wide Web since 1993. This specification defines the interface known as 'CGI/1.1', and its use on the Unix(R) and AmigaDOS(tm) systems.

1 Introduction

1.1 Purpose

Together the HTTP [3] 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) [1], a request method and various ancillary information about the request provided by the transport mechanism.

The CGI defines the abstract parameters, known as environment variables, 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.

1.2 Requirements

This specification uses the same words as RFC 1123 [5] to define the significance of each particular requirement. These are:

must
: This word or the adjective 'required' means that the item is an absolute requirement of the specification.

should
: This word or the adjective 'recommended' means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.

may
: This word or the adjective 'optional' means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

An implementation is not compliant if it fails to satisfy one or more of the 'must' requirements for the protocols it implements. An implementation that satisfies all of the 'must' and all of the 'should' requirements for its features is said to be 'unconditionally compliant'; one that satisfies all of the 'must' requirements but not all of the 'should' requirements for its features is said to be 'conditionally compliant'.

1.3 Specifications

Not all of the functions and features of the CGI are defined in the main part of this specification. The following phrases are used to describe the features which are not specified:

system defined
: The feature may differ between systems, but must be the same for different implementations using the same system. A system will usually identify a class of operating-systems. Some systems are defined in section 12 of this document. New systems may be defined by new specifications without revision of this document.

implementation defined
: The behaviour of the feature may vary from implementation to implementation, but a particular implementation must document its behaviour.

1.4 Terminology

This specification uses many terms defined in the HTTP/1.0 specification [3]; however, the following terms are used here in a sense which may not accord with their definitions in that document, or with their common meaning.

environment variable
A named parameter that carries information from the server to the script. It is not necessarily a variable in the operating-system's environment, although that is the most common implementation.

script
The software which is invoked by the server via this interface. It need not be a standalone program, but could be a dynamically-loaded or shared library, or even a subroutine in the server.

server
The application program which invokes the script in order to service requests.

2 Notational Conventions and Generic Grammar

2.1 Augmented BNF

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) similar to that used by RFC 822 [6]. This augmented BNF contains the following constructs:

name = definition
The name of a rule is simply the name itself; it is separated from the definition by the equal character ("="). Whitespace is only significant in that continuation lines of a definition are indented.

"literal"
Quotation marks (") surround literal text, except for a literal quotation mark, which is surrounded by angle-brackets ("<" and ">"). Unless stated otherwise, the text is case-sensitive.

rule1 | rule2
Alternative rules are separated by a vertical bar ("|").

(rule1 rule2 rule3)
Elements enclosed in parentheses are treated as a single element.

***rule**
A rule preceded by an asterisk ("*") may have zero or more occurrences. A rule preceded by an integer followed by an asterisk must occur at least the specified number of times.

[rule]
A element enclosed in square brackets ("[" and "]") is optional.

2.2 Basic Rules

The following rules are used throughout this specification to describe basic parsing constructs.

```
alpha      = lowalpha | hialpha
lowalpha   = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" |
             "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
```

hialpha = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P"
| "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
OCTET = <any 8-bit byte>
CHAR = <any character>
CTL = <any control character>
SP = <space character>
NL = <newline>
LWSP = SP | NL | <horizontal-tab>
tspecial = "(" | ")" | "@" | "," | ";" | ":" | "\" | <"> | "/" | "[" | "]" | "?" | SP
token = 1*<any CHAR except CTLs or tspecials>
quoted-string = (<"> *qdtext <">) | ("<" *qatext ">")
qdtext = <any CHAR except <"> and CTLs but including LWSP>
qatext = <any CHAR except "<", ">" and CTLs but including LWSP>

Note that newline (NL) need not be a single character, but can be a character sequence.

3 URL Encoding

Some variables and constructs used here are described as being ‘URL-encoded’. This encoding is described in section 2.2 of RFC 1738 [4]. In a URL encoded string an escape sequence consists of a percent character ("%") followed by two hexadecimal digits, where the two hexadecimal digits form an octet. An escape sequence represents the graphic character which has the octet as its code within the US-ASCII [11] coded character set, if it exists. If no such graphic character exists, then the escape sequence represents the octet value itself.

Note that some unsafe characters may have different semantics if they are encoded. The definition of which characters are unsafe depends on the context.

4 The Script URI

A ‘Script URI’ can be defined; this describes the resource identified by the environment variables. Often, this URI will be the same as the URI requested by the client (the ‘Client URI’); however, it need not be. Instead, it could be a URI invented by the server, and so it can only be used in the context of the server and its CGI interface.

The script URI has the syntax of generic-RL as defined in section 2.1 of RFC 1808 [7], with the exception that object parameters and fragment identifiers are not permitted:

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

The various components of the script URI are defined by some of the environment variables (see below);

script-uri = protocol "://" SERVER_NAME ":" SERVER_PORT enc-script enc-path-info "?" QUERY_STRING where 'protocol' is found from SERVER_PROTOCOL, 'enc-script' is a URL-encoded version of SCRIPT_NAME and 'enc-path-info' is a URL-encoded version of PATH_INFO.

5 Environment variables

Environment variables are used to pass data about the request from the server to the script. They are accessed by the script in a system defined manner. In all cases, a missing environment variable is equivalent to a zero-length (NULL) value, and vice versa. The representation of the characters in the environment variables is system defined.

Case is not significant in the names, in that there cannot be two different variable whose names differ in case only. Here they are shown using a canonical representation of capitals plus underscore ("_"). The actual representation of the names is system defined; for a particular system the representation may be defined differently to this.

The variables are:

AUTH_TYPE
CONTENT_LENGTH
CONTENT_TYPE
GATEWAY_INTERFACE
HTTP_*
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

AUTH_TYPE

This variable is specific to requests made with HTTP.

If the script URI would require access authentication for external access, then this variable is found from the 'auth-scheme' token in the request, otherwise NULL.

AUTH_TYPE = "" | auth-scheme
auth-scheme = "Basic" | token

HTTP access authentication schemes are described in section 11 of the HTTP/1.0 specification [3]. The auth-scheme is not case-sensitive.

CONTENT_LENGTH

The size of the entity attached to the request, if any, in decimal number of octets. If no data is attached, then

NULL. The syntax is the same as the HTTP Content-Length header (section 10, HTTP/1.0 specification [3]).

CONTENT_LENGTH = "" | [1*digit]

CONTENT_TYPE

The Internet Media Type [9] of the attached entity. The syntax is the same as the HTTP Content-Type header.

```
CONTENT_TYPE = "" | media-type
media-type   = type "/" subtype *( ";" parameter)
type         = token
subtype      = token
parameter    = attribute "=" value
attribute    = token
value        = token | quoted-string
```

The type, subtype and parameter attribute names are not case-sensitive. Parameter values may be case sensitive. Media types and their use in HTTP are described section 3.6 of the HTTP/1.0 specification [3]. Example:

```
application/x-www-form-urlencoded
```

There is no default value for this variable. If and only if it is unset, then the script may attempt to determine the media type from the data received. If the type remains unknown, then `application/octet-stream` should be assumed.

GATEWAY_INTERFACE

The version of the CGI specification to which this server complies. Syntax:

GATEWAY_INTERFACE = "CGI" "/" 1*digit "." 1*digit

Note that the major and minor numbers are treated as separate integers and that each may be incremented higher than a single digit. Thus CGI/2.4 is a lower version than CGI/2.13 which in turn is lower than CGI/12.3. Leading zeros must be ignored by scripts and should never be generated by servers.

This document defines the 1.1 version of the CGI interface.

HTTP_*

These variables are specific to requests made with HTTP. Interpretation of these variables may depend on the value of SERVER_PROTOCOL.

Environment variables with names beginning with "HTTP_" contain header data read from the client, if the protocol used was HTTP. The HTTP header name is converted to upper case, has all occurrences of "-" replaced with "_" and has "HTTP_" prepended to give the environment variable name. The header data may be presented as sent by the client, or may be rewritten in ways which do not change its semantics. If multiple headers with the same field-name are received then they must be rewritten as a single header having the same semantics. Similarly, a header that is received on more than one line must be merged onto a single line. The server must, if necessary, change the representation of the data (for example, the character set) to be appropriate for a CGI environment variable.

The server is not required to create environment variables for all the headers that it receives. In particular, it may remove any headers carrying authentication information, such as "Authorization"; it may remove headers whose value is available to the script via other variables, such as "Content-Length" and "Content-Type".

PATH_INFO

A path to be interpreted by the CGI script. It identifies the resource or sub-resource to be returned by the CGI script. The syntax and semantics are similar to a decoded HTTP URL 'hpath' token (defined in RFC 1738 [4]), with the exception that a PATH_INFO of "/" represents a single void path segment. Otherwise, the leading "/" character is not part of the path.

```
PATH_INFO =  "" | "/" path
path      =  segment *( "/" segment )
segment   =  *pchar
pchar     =  <any CHAR except "/">
```

The PATH_INFO string is the trailing part of the <path> component of the script URI that follows the SCRIPT_NAME part of the path.

PATH_TRANSLATED

The OS path to the file that the server would attempt to access were the client to request the absolute URL containing the path PATH_INFO. i.e for a request of

```
protocol "://" SERVER_NAME ":" SERVER_PORT enc-path-info
```

where 'enc-path-info' is a URL-encoded version of PATH_INFO. If PATH_INFO is NULL then PATH_TRANSLATED is set to NULL.

```
PATH_TRANSLATED = *CHAR
```

PATH_TRANSLATED need not be supported by the server. The server may choose to set PATH_TRANSLATED to NULL for reasons of security, or because the path would not be interpretable by a CGI script; such as the object it represented was internal to the server and not visible in the file-system; or for any other reason.

The algorithm the server uses to derive PATH_TRANSLATED is obviously implementation defined; CGI scripts which use this variable may suffer limited portability.

QUERY_STRING

A URL-encoded search string; the <query> part of the script URI.

```
QUERY_STRING =  query-string
query-string =  *qchar
qchar        =  unreserved | escape | reserved
unreserved   =  alpha | digit | safe | extra
reserved     =  ";" | "/" | "?" | ":" | "@" | "&" | "="
safe         =  "$" | "-" | "_" | "." | "+"
extra        =  "!" | "*" | "'" | "(" | ")" | ","
escape       =  "%" hex hex
hex          =  digit | "A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f"
```

The URL syntax for a search string is described in RFC 1738 [4].

REMOTE_ADDR

The IP address of the agent sending the request to the server. Not necessarily that of the client.

```
REMOTE_ADDR = hostnumber
hostnumber  = digits "." digits "." digits "." digits
digits      = 1*digit
```

REMOTE_HOST

The fully qualified domain name of the agent sending the request to the server, if available, otherwise NULL. Not necessarily that of the client. Fully qualified domain names take the form as described in section 3.5 of RFC 1034 [8] and section 2.1 of RFC 1123 [5]; a sequence of domain labels separated by ".", each domain label starting and ending with an alphanumerical character and possibly also containing "-" characters. The rightmost domain label will never start with a digit. Domain names are not case sensitive.

```
REMOTE_HOST  = "" | hostname
hostname     = *( domainlabel ".") toplabel
domainlabel  = alphadigit [ *alphahypdigit alphadigit ]
toplabel     = alpha [ *alphahypdigit alphadigit ]
alphahypdigit = alphadigit | "-"
alphadigit   = alpha | digit
```

REMOTE_IDENT

The identity information reported about the connection by a RFC 931 [10] request to the remote agent, if available. The server may choose not to support this feature, or not to request the data for efficiency reasons.

```
REMOTE_IDENT = *CHAR
```

The data returned is not appropriate for use as authentication information.

REMOTE_USER

This variable is specific to requests made with HTTP.

If AUTH_TYPE is "Basic", then the user-ID sent by the client. If AUTH_TYPE is NULL, then NULL, otherwise undefined.

```
REMOTE_USER = "" | userid | *OCTET
userid      = token
```

REQUEST_METHOD

This variable is specific to requests made with HTTP.

The method with which the request was made, as described in section 5.1.1 of the HTTP/1.0 specification [3].

```
REQUEST_METHOD   = http-method
http-method      = "GET" | "HEAD" | "POST" | extension-method
extension-method = token
```

The method is case sensitive.

SCRIPT_NAME

A URL path that could identify the CGI script (rather then the particular CGI output). The syntax and semantics are identical to a decoded HTTP URL ‘hpath’ token [4].

```
SCRIPT_NAME = "" | "/" [ path ]
```

The leading "/" is not part of the path. It is optional if the path is NULL.

The SCRIPT_NAME string is some leading part of the <path> component of the script URI derived in some implementation defined manner.

SERVER_NAME

The name for this server, as used in the <host> part of the script URI. Thus either a fully qualified domain name, or an IP address.

SERVER_NAME = hostname | hostnumber

SERVER_PORT

The port on which this request was received, as used in the <port> part of the script URI.

SERVER_PORT = 1*digit

SERVER_PROTOCOL

The name and revision of the information protocol this request came in with.

```
SERVER_PROTOCOL =  HTTP-Version | extension-version
HTTP-Version    =  "HTTP" "/" 1*digit "." 1*digit
extension-version = protocol "/" 1*digit "." 1*digit
protocol        =  1*( alpha | digit | "+" | "-" | "." )
```

'protocol' is a version of the <scheme> part of the script URI, and is not case sensitive. By convention, 'protocol' is in upper case.

SERVER_SOFTWARE

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

SERVER_SOFTWARE = *CHAR

6 Invoking the script

This script is invoked in a system defined manner. Unless specified otherwise, this will be by treating the file containing the script as an executable, and running it as a child process of the server.

7 The CGI script command line

Some systems support a method for supplying a array of strings to the CGI script. This is only used in the case of an 'indexed' query. This is identified by a "GET" or "HEAD" HTTP request with a URL search string not containing any unencoded "=" characters. For such a request, the server should parse the search string into words, using the rule:

```
search-string = search-word *( "+" search-word )
search-word   = 1*schar
schar         = xunreserved | escape | xreserved
```

```
xunreserved  =  alpha | digit | xsafe | extra
xsafe        =  "$" | "-" | "_" | "."
xreserved    =  ";" | "/" | "?" | ":" | "@" | "&"
```

After parsing, each word is URL-decoded, optionally encoded in a system defined manner and then the argument list is set to the list of words.

If the server cannot create any part of the argument list, then the server should generate no command line information. For example, the number of arguments may be greater than operating system or server limitations, or one of the words may not be representable as an argument.

8 Data input to the CGI script

As there may be a data entity attached to the request, there must be a system defined method for the script to read this data. Unless defined otherwise, this will be via the 'standard input' file descriptor.

There will be at least CONTENT_LENGTH bytes available for the script to read. The script is not obliged to read the data, but it must not attempt to read more than CONTENT_LENGTH bytes, even if more data is available.

For non-parsed header (NPH) scripts (see below), the server should attempt to ensure that the script input comes directly from the client, with minimal buffering. For all scripts the data will be as supplied by the client.

9 Data output from the CGI script

There must be a system defined method for the script to send data back to the server or client; a script will always return some data. Unless defined otherwise, this will be via the 'standard output' file descriptor.

There are two forms of output that the script can give; non-parsed header (NPH) output, and parsed header output. A server is only required to support the latter; distinguishing between the two types of output (or scripts) is implementation defined.

9.1 Non-Parsed Header Output

The script must return a complete HTTP response message, as described in Section 6 of the HTTP specification [3]. Note that this allows an HTTP/0.9 response to an HTTP/1.0 request.

The server should attempt to ensure that the script output is sent directly to the client, with minimal buffering.

9.2 Parsed Header Output

The script returns a CGI response message.

```
CGI-Response = *( CGI-Header | HTTP-Header ) NL [ Entity-Body ]
CGI-Header   = Content-type
             | Location
             | Status
             | extension-header
```

The response comprises headers and a body, separated by a blank line. The headers are either CGI headers to be interpreted by the server, or HTTP headers to be included in the response returned to the client if the request method is HTTP. At least one CGI-Header must be supplied, but no CGI header can be repeated with the same field-name. If a body is supplied, then a Content-type header is required, otherwise the script must send a Location or Status header. If a Location header is returned, then no HTTP-Headers may be supplied.

The CGI headers have the generic syntax:

```
generic-header = field-name ":" [ field-value ] NL
field-name     = 1*<any CHAR, excluding CTLs, SP and ":">
field-value    = *( field-content | LWSP )
field-content  = *( token | tspecial | quoted-string )
```

The field-name is not case sensitive; a NULL field value is equivalent to the header not being sent.

Content-Type

The Internet Media Type [9] of the entity body, which is to be sent unmodified to the client.

```
Content-Type = "Content-Type" ":" media-type NL
```

Location

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

```
Location         = "Location" ":" ( fragment-URI | rel-URL-abs-path ) NL
fragment-URI     = URI [ # fragmentid ]
URI              = scheme ":" *qchar
fragmentid       = *qchar

rel-URL-abs-path = "/" [ hpath ] [ "?" query-string ]
hpath            = fpsegment *( "/" psegment )
fpsegment        = 1*hchar
psegment         = *hchar
hchar            = alpha | digit | safe | extra | ":" | "@" | "& | "="
```

The location value is either an absolute URI with optional fragment, as defined in RFC 1630 [1], or an absolute path and optional query-string. If an absolute URI is returned by the script, then the server will generate a redirect HTTP response message, and if no entity body is supplied by the script, then the server will produce one. If the Location value is a path, then the server will generate the response that it would have produced in response to a request containing the URL

```
protocol "://" SERVER_NAME ":" SERVER_PORT rel-URL-abs-path
```

The location header may only be sent if the REQUEST_METHOD is HEAD or GET.

Status

The Status header is used to indicate to the server what status code it will use in the response message. It should not be sent if the script returns a Location header.

Status	=	"Status" ":" 3digit SP reason-phrase NL
reason-phrase	=	*<CHAR, excluding CTLs, NL>

The valid status codes are listed in section 6.1.1 of the HTTP/1.0 specification [3]. If the script does not return a Status header, then "200 OK" should be assumed.

HTTP headers

The script may return any other headers defined by the HTTP/1.0 specification [3]. The server must translate the header data from the CGI header syntax to the HTTP header syntax if these differ. For example, the character sequence for newline (such as Unix's ASCII NL) used by CGI scripts may not be the same as that used by HTTP (ASCII CR followed by LF). The server must also resolve any conflicts between headers returned by the script and headers that it would otherwise send itself.

10 Requirements for servers

Servers must support the standard mechanism (described below) which allows the script author to determine what URL to use in documents which reference the script. Specifically, what URL to use in order to achieve particular settings of the environment variables. This mechanism is as follows:

The value for SCRIPT_NAME is governed by the server configuration and the location of the script in the OS file-system. Given this, any access to the partial URL

SCRIPT_NAME extra-path ? query-information

where extra-path is either NULL or begins with a "/" and satisfies any other server requirements, will cause the CGI script to be executed with PATH_INFO set to the decoded extra-path, and QUERY_STRING set to query-information (not decoded).

Servers may reject with error 404 any requests that would result in an encoded "/" being decoded into PATH_INFO or SCRIPT_NAME, as this might represent a loss of information to the script.

Although the server and the CGI script need not be consistent in their handling of URL paths (client URLs and the PATH_INFO data, respectively), server authors may wish to impose consistency. So the server implementation should define its behaviour for the following cases:

- define any restrictions on allowed characters, in particular whether ASCII NULL is permitted;
- define any restrictions on allowed path segments, in particular whether non-terminal NULL segments are permitted;
- define the behaviour for `"."` or `".."` path segments; i.e. whether they are prohibited, treated as ordinary path segments or interpreted in accordance with the relative URL specification [7];

- define any limits of the implementation, including limits on path or search string lengths, and limits on the volume of headers the server will parse.

Servers may generate the script URI in any way from the client URI, or from any other data (but the behaviour should be documented).

11 Recommendations for scripts

Scripts should reject unexpected methods (such as DELETE etc.) with error 405 Method Not Allowed. If the script does not intend processing the PATH_INFO data, then it should reject the request with 404 Not Found if PATH_INFO is not NULL.

If the output of a form is being processed, check that CONTENT_TYPE is "application/x-www-form-urlencoded" [2].

If parsing PATH_INFO, PATH_TRANSLATED or SCRIPT_NAME then be careful of void path segments ("//") and special path segments ("." and ".."). They should either be removed from the path before use in OS system calls, or the request should be rejected with 404 Not Found. It is very unlikely that any other use could be made of these.

As it is impossible for the script to determine the client URI that initiated this request without knowledge of the specific server in use, the script should not return text/html documents containing relative URL links without including a <BASE> tag in the document.

When returning headers, the script should try to send the CGI headers as soon as possible, and preferably before any HTTP headers. This may help reduce the server's memory requirements.

12 System specifications

12.1 AmigaDOS

Environment variables
: These are accessed by the DOS library routine `GetVar`. The flags argument should be 0. Case is ignored, but upper case is recommended for compatibility with case-sensitive systems.

The current working directory
: The current working directory for the script is set to the directory containing the script.

Character set
: The US-ASCII character set is used for the definition of environment variables and headers; the newline (NL) sequence is CR LF.

12.2 Unix

For Unix compatible operating systems, the following are defined:

Environment variables
: These are accessed by the C library routine `getenv`.

The command line
: This is accessed using the the `argc` and `argv` arguments to `main()`. The words are have any characters which are 'active' in the Bourne shell escaped with a backslash.

The current working directory
: The current working directory for the script is set to the directory containing the script.

Character set
: The US-ASCII character set is used for the definition of environment variables and headers; the newline (NL) sequence is LF; servers should also accept CR LF as a newline.

13 Security Considerations

13.1 Safe Methods

As discussed in the security considerations of the HTTP specification [3], the convention has been established that the GET and HEAD methods should be 'safe'; they should cause no side-effects and only have the significance of resource retrieval.

13.2 HTTP headers containing sensitive information

Some HTTP headers may carry sensitive information which the server should not pass on to the script unless explicitly configured to do so. For example, if the server protects the script using the Basic authentication scheme, then the client will send an Authorization header containing a username and password. If the server, rather than the script, validates this information then it should not pass on the password via the HTTP_AUTHORIZATION environment variable.

13.3 Script interference with the server

The most common implementation of CGI invokes the script as a child process using the same user and group as the server process. It should therefore be ensured that the script cannot interfere with the server process, its configuration or documents.

If the script is executed by calling a function linked in to the server software (either at compile-time or run-time) then precautions should be taken to protect the core memory of the server, or to ensure that untrusted code cannot be executed.

14 Acknowledgements

This work is based on the original CGI interface that arose out of discussions on the *www-talk* mailing list. In particular, Rob McCool, John Franks, Ari Luotonen, George Phillips and Tony Sanders deserve special recognition for their efforts in defining and implementing the early versions of this interface.

This document has also greatly benefited from the comments and suggestions made Chris Adie, Dave Kristol and Mike Meyer.

15 References

[1] Berners-Lee, T., 'Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web', RFC 1630, CERN, June 1994.

[2] Berners-Lee, T. and Connolly, D., 'Hypertext Markup Language - 2.0', RFC 1866, MIT/W3C, November 1995.

[3] Berners-Lee, T., Fielding, R. T. and Frystyk Nielsen, H., 'Hypertext Transfer Protocol – HTTP/1.0', Work in progress (draft-ietf-http-v10-spec-04.txt), MIT/LCS, UC Irvine, October 1995.

[4] Berners-Lee, T., Masinter, L. and McCahill, M., Editors, 'Uniform Resource Locators (URL)', RFC 1738, CERN, Xerox Corporation, University of Minnesota, December 1994.

[5] Braden, R., Editor, 'Requirements for Internet Hosts – Application and Support', STD 3, RFC 1123, IETF, October 1989.

[6] Crocker, D.H., 'Standard for the Format of ARPA Internet Text Messages', STD 11, RFC 822, University of Delaware, August 1982.

[7] Fielding, R., 'Relative Uniform Resource Locators', RFC 1808, UC Irving, June 1995.

[8] Mockapetris, P., 'Domain Names - Concepts and Facilities', STD 13, RFC 1034, ISI, November 1987.

[9] Postel, J., 'Media Type Registration Procedure', RFC 1590, ISI, March 1994.

[10] StJohns, M., 'Authentication Server', RFC 931, TPSC, January 1985.

[11] 'Coded Character Set – 7-bit American Standard Code for Information Interchange', ANSI X3.4-1986.

16 Author's Address

David Robinson
Institute of Astronomy
University of Cambridge
Madingley Road
Cambridge CB3 0HA
UK

Tel: +44 (1223) 337528
Fax: +44 (1223) 337523
EMail: drtr@ast.cam.ac.uk