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 view the entire list of current Internet-Drafts, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), ftp.nordu.net (Northern Europe), ftp.nis.garr.it (Southern Europe), munnari.oz.au (Pacific Rim), ftp.ietf.org (US East Coast), or ftp.isi.edu (US West Coast).

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 "current practice" parameters of the 'CGI/1.1' interface developed and documented at the U.S. National Centre for Supercomputing Applications [NCSA-CGI]. This document also defines the use of the CGI/1.1 interface on the Unix and AmigaDOS(tm) systems.

Discussion of this draft occurs on the CGI-WG mailing list; see the project Web page at <URL:http://Web.Golux.Com/coar/cgi/> for details on the mailing list and the status of the project.
1. Introduction

1.1. Purpose

Together the HTTP [3],[8] 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 meta-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

Coar, et al.                 INTERNET-DRAFT                     [Page 3]

CGI/1.1           Expires: 14 May, 1999

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/1.1 specification [8]; 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.

meta-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 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.

\[
\begin{align*}
\text{alpha} & \quad = \text{lowalpha} \mid \text{hialpha} \\
\text{lowalpha} & \quad = "a" \mid "b" \mid "c" \mid "d" \mid "e" \mid "f" \mid "g" \mid "h" \\
& \quad \mid "i" \mid "j" \mid "k" \mid "l" \mid "m" \mid "n" \mid "o" \mid "p" \\
& \quad \mid "q" \mid "r" \mid "s" \mid "t" \mid "u" \mid "v" \mid "w" \mid "x" \\
& \quad \mid "y" \mid "z" \\
\text{hialpha} & \quad = "A" \mid "B" \mid "C" \mid "D" \mid "E" \mid "F" \mid "G" \mid "H" \\
& \quad \mid "I" \mid "J" \mid "K" \mid "L" \mid "M" \mid "N" \mid "O" \mid "P" \\
& \quad \mid "Q" \mid "R" \mid "S" \mid "T" \mid "U" \mid "V" \mid "W" \mid "X" \\
& \quad \mid "Y" \mid "Z" \\
\text{hex} & \quad = \text{digit} \mid "A" \mid "B" \mid "C" \mid "D" \mid "E" \mid "F" \mid "a" \\
& \quad \mid "b" \mid "c" \mid "d" \mid "e" \mid "f" \\
\text{digit} & \quad = "0" \mid "1" \mid "2" \mid "3" \mid "4" \mid "5" \mid "6" \mid "7" \\
& \quad \mid "8" \mid "9" \\
\text{OCTET} & \quad = \text{<any 8-bit byte>} \\
\text{CHAR} & \quad = \text{<any character>} \\
\text{CTL} & \quad = \text{<any control character>} \\
\text{SP} & \quad = \text{<space character>} \\
\text{HT} & \quad = \text{<horizontal tab character>} \\
\text{NL} & \quad = \text{<newline>} \\
\text{LWSP} & \quad = \text{SP} \mid \text{HT} \mid \text{NL} \\
\text{tspecial} & \quad = "]" \mid "]" \mid "]" \mid "]" \mid "]" \mid "]" \mid "]" \mid "]" \mid "]" \\
& \quad \mid "]" \mid "]" \mid "?" \mid "]" \mid "]" \mid "]" \mid "?" \mid "?" \\
& \quad \mid \text{SP} \mid \text{HT} \\
\text{token} & \quad = 1\times\text{<any CHAR except CTLs or tspecials>}
\end{align*}
\]
3. Protocol Parameters

3.1. URL Encoding

Some variables and constructs used here are described as being 'URL-encoded'. This encoding is described in section 2 of RFC 2396 [4].

An alternate "shortcut" encoding for representing the space character exists and is in common use. Scripts MUST be prepared to recognise both '+' and '%20' as an encoded space in a URL-encoded value.

Note that some unsafe characters may have different semantics if they are encoded. The definition of which characters are unsafe depends on the context. For example, the following two URLs do not necessarily refer to the same resource:

http://somehost.com/somedir%2Fvalue
http://somehost.com/somedir/value


3.2. The Script URI

The 'Script-URI' is defined as the URI of the resource identified by the meta-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 meta-variables (see section 4 below);

\[
\text{script-uri} = \text{protocol} "://" \text{SERVER_NAME} ":" \text{SERVER_PORT} \text{ enc-script} \\
\text{enc-path-info} "?" \text{ QUERY_STRING}
\]

where 'protocol' is obtained 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. See section 4.6 for more information about the PATH_INFO meta-variable.

Note that the scheme and the protocol are not identical; for instance, a resource accessed via an SSL mechanism may have a Client-URI with a scheme of "https" rather than "http". CGI/1.1 provides no means for the script to reconstruct this, and therefore the Script-URI includes the base protocol used.

4. Invoking the Script

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

5. The CGI Script Command Line

Some systems support a method for supplying an 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 query string not containing any unencoded "=" characters. For such a request, servers SHOULD parse the search string into words, using the following rules:

\[
\begin{align*}
\text{search-string} & = \text{search-word} \ast ( \text{"+"} \text{ search-word} ) \\
\text{search-word} & = \text{1*schar} \\
\text{schar} & = \text{xunreserved} | \text{escape} | \text{xreserved} \\
\text{xunreserved} & = \text{alpha} | \text{digit} | \text{xsafe} | \text{extra} \\
\text{xsafe} & = \text{"$"} | \text{"-"} | \text{"_"} | \text{"."} \\
\text{xreserved} & = \text{";"} | \text{"/"} | \text{"?"} | \text{":"} | \text{"@"} | \text{"&"}
\end{align*}
\]

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 NOT generate any command line information. For example, the number of arguments may be greater than operating system or server limitations permit, or one of the words may not be representable as an argument.

Scripts SHOULD check to see if the QUERY_STRING value contains an unencoded "=" character, and SHOULD NOT use the command line arguments if it does.

6. Data Input to the CGI Script

Information about a request comes from two different sources: the request header, and any associated content-body. Servers MUST make portions of this information available to scripts.

6.1. Request Metadata (Meta-Variables)

Each CGI server implementation MUST define a mechanism to pass data about the request from the server to the script. The meta-variables containing these data are accessed by the script in a system defined manner. In all cases, a missing meta-variable is equivalent to a zero-length (NULL) value, and vice versa. The representation of the characters in the meta-variables is system defined.

Case is not significant in the meta-variable names, in that there cannot be two different variables 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 than this.

Meta-variable values MUST be considered case-sensitive except as noted otherwise.

The canonical variables defined by this specification are:

- AUTH_TYPE
- CONTENT_LENGTH
- CONTENT_TYPE
- GATEWAY_INTERFACE
- HTTP_*
6.1.1. AUTH_TYPE

This variable is specific to requests made via the HTTP scheme.

If the script-URI required access authentication for external access, then the server SHOULD set the value of this variable from the 'auth-scheme' token in the request's "Authorization" header field. Otherwise it is set to NULL.

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

HTTP access authentication schemes are described in section 11 of the HTTP/1.1 specification [8]. The auth-scheme is not case-sensitive.

Servers SHOULD provide this meta-variable to scripts if the request header included an "Authorization" field.

6.1.2. CONTENT_LENGTH

This meta-variable is set to the size of the content-body entity attached to the request, if any, in decimal number of octets. If no data are attached, then this meta-variable is either NULL or not defined. The syntax is the same as for the HTTP "Content-Length" header field (section 14.14, HTTP/1.1 specification [8]).

```
CONTENT_LENGTH = "" | 1*digit
```
Servers MUST provide this meta-variable to scripts if the request was accompanied by a content-body entity.

6.1.3. CONTENT_TYPE

If the request includes a content-body, CONTENT_TYPE is set to the Internet Media Type [9] of the attached entity if the type was provided via a "Content-type" field in the request header, or if the server can determine it in the absence of a supplied "Content-type" field. The syntax is the same as for the HTTP "Content-Type" header field.

```
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 in section 3.7 of the HTTP/1.1 specification [8].

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 the script MAY choose to either assume a content-type of application/octet-stream or reject the request with a 415 ("Unsupported Media Type") error. See section 7.2.1.3 for more information about returning error status values.

Servers MUST provide this meta-variable to scripts if a "Content-Type" field was present in the original request header. If the server receives a request with an attached entity but no "Content-Type" header field, it MAY attempt to determine the correct datatype, or it MAY omit this meta-variable when communicating the request information to
the script.

6.1.4. GATEWAY_INTERFACE

This meta-variable is set to the dialect of CGI being used by the server to communicate with the script. Syntax:

GATEWAY_INTERFACE = "CGI" "\" major "." minor
major         = 1*digit
minor         = 1*digit

Note that the major and minor numbers are treated as separate integers and hence each may be more 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 in either the major or the minor number MUST be ignored by scripts and SHOULD NOT be generated by servers.

This document defines the 1.1 version of the CGI interface ("CGI/1.1").

Servers MUST provide this meta-variable to scripts.

6.1.5. HTTP_*

These variables are specific to requests made via the HTTP scheme. Interpretation of these variables depends on the value of the SERVER_PROTOCOL meta-variable (see section 6.1.17).

Meta-variables with names beginning with "HTTP_" contain values from the request header, if the scheme used was HTTP. Each HTTP header field name is converted to upper case, has all occurrences of "-" replaced with ",", and has "HTTP_" prepended to form the meta-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 header fields with the same field-name are received then the server MUST rewrite them as though they had been received as a single header field having the same semantics before being represented in a meta-variable. Similarly, a header field that is received on more than one line MUST be merged into 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 meta-variable.

Servers are not required to create meta-variables for all the
request header fields that they receive. In particular, they MAY decline to make available any header fields carrying authentication information, such as "Authorization"; and they MAY header fields the values of which are available to the script via other metavariables, such as "Content-Length" and "Content-Type".

6.1.6. PATH_INFO

The PATH_INFO meta-variable specifies a path to be interpreted by the CGI script. It identifies the source or sub-resource to be returned by the CGI script, and it is derived from the portion of the URI path following the script name but preceding any query data. The syntax and semantics are similar to a decoded HTTP URL 'path' token (defined in RFC 2396 [4]), with the exception that a PATH_INFO of "/" represents a single void path segment.

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 (see section 3.2) that follows the SCRIPT_NAME portion of the path.

Servers MAY impose their own restrictions and limitations on what values they will accept for PATH_INFO, and MAY reject or edit any values they considers objectionable before passing them to the script.

Servers MUST make this URI component available to CGI scripts. The PATH_INFO quantity is case-sensitive, and the server MUST preserve the case of the PATH_INFO element of the URI when making it available to scripts.

6.1.7. PATH_TRANSLATED

PATH_TRANSLATED is derived by taking any path-info component of the request URI (see section 6.1.6), decoding it (see section 3.1), parsing it as a URI in its own right, and performing any virtual-to-physical filesystem translation.

PATH_TRANSLATED = *CHAR

For a request such as the following:
the PATH_INFO component would be decoded, and the result parsed as though it were a request for the following:

```
http://somehost.com/this.is.the.path.info
```

This would then be translated to a filesystem location, perhaps something like this:

```
/usr/local/www/htdocs/this.is.the.path.info
```

This resulting filesystem path is the value of PATH_TRANSLATED.

The value of PATH_TRANSLATED may or may not map to a valid filesystem location or file. Servers MUST preserve the case of the path-info segment if and only if the underlying filesystem supports case-sensitive names. If the filesystem is only case-aware, case-preserving, or case-blind with regard to filenames, servers are not required to preserve the case of the original segment through the translation.

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

Servers SHOULD provide this meta-variable to scripts.

### 6.1.8. QUERY_STRING

A URL-encoded string; the <query> part of the script-URI. (See section 3.2.)

```
QUERY_STRING = query-string
query-string = *qchar
qchar = unreserved | escape | reserved
unreserved = alpha | digit | safe | extra
reserved = ; | / | ? | : | @ | & | =
safe = $ | _ | . | +
extra = ! | * | " | ( | )
escape = % hex hex
```

The URL syntax for a query string is described in section 3 of
6.1.9. REMOTE_ADDR

The IP address of the agent sending the request to the server. This is not necessarily that of the client (such as if the request came through a proxy).

\[
\text{REMOTE_ADDR} = \text{hostnumber}
\]

\[
\text{hostnumber} = \text{ipv4-address} \mid \text{ipv6-address}
\]

\[
\text{ipv4-address} = \text{digits} \cdot \text{digits} \cdot \text{digits} \cdot \text{digits}
\]

\[
\text{ipv6-address} = \text{hexbit16} \cdot \text{hexbit16} \cdot \text{hexbit16} \cdot \text{hexbit16}
\]

Servers MUST supply this value to scripts.

6.1.10. REMOTE_HOST

The fully qualified domain name of the agent sending the request to the server, if available, otherwise NULL. (See section 6.1.9.) Fully qualified domain names take the form as described in section 3.5 of RFC 1034 [10] and section 2.1 of RFC 1123 [5]. Domain names are not case sensitive.

Servers SHOULD provide this information to scripts.

6.1.11. REMOTE_IDENT

The identity information reported about the connection by a RFC 1413 [11] request to the remote agent, if available. Servers MAY choose not to support this feature, or not to request the data for efficiency reasons.

\[
\text{REMOTE_IDENT} = \ast \text{CHAR}
\]

The data returned may be used for authentication purposes, but the level of trust reposed in them should be minimal.
Servers MAY supply this information to scripts if the RFC1413 [11] lookup is performed.

6.1.12. REMOTE_USER

If the request required authentication using the "Basic" mechanism (i.e., the AUTH_TYPE meta-variable is set to "Basic"), then the value of the REMOTE_USER meta-variable is set to the user-ID supplied. In all other cases the value of this meta-variable is undefined.

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

This variable is specific to requests made via the HTTP protocol.

Servers SHOULD provide this meta-variable to scripts.

6.1.13. REQUEST_METHOD

The REQUEST_METHOD meta-variable is set to the method with which the request was made, as described in section 5.1.1 of the HTTP/1.0 specification [3] and section 5.1.1 of the HTTP/1.1 specification [8].

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

The method is case sensitive. Note that of the new methods defined by the HTTP/1.1 specification [8], OPTIONS and TRACE are not appropriate for the CGI/1.1 environment.

This variable is specific to requests made with HTTP.

Servers MUST provide this meta-variable to scripts.

6.1.14. SCRIPT_NAME

The SCRIPT_NAME meta-variable is set to a URL path that could identify the CGI script (rather than the script's output). The syntax and semantics are identical to a decoded HTTP URL
'path' token (see RFC 2396 [4]).

SCRIPT_NAME = "" | ( "/" [ path ] )

The SCRIPT_NAME string is some leading part of the <path> component of the script-URI derived in some implementation defined manner. No PATH_INFO or QUERY_STRING segments (see sections 6.1.6 and 6.1.8) are included in the SCRIPT_NAME value.

Servers MUST provide this meta-variable to scripts.

6.1.15. SERVER_NAME

The SERVER_NAME meta-variable is set to the name of the server, as derived from the <host> part of the script-URI (see section 3.2).

SERVER_NAME = hostname | hostnumber

Servers MUST provide this meta-variable to scripts.

6.1.16. SERVER_PORT

The SERVER_PORT meta-variable is set to the port on which the request was received, as used in the <port> part of the script-URI.

SERVER_PORT = 1*digit

If the <port> portion of the script-URI is blank, the actual port number upon which the request was received MUST be supplied.

Servers MUST provide this meta-variable to scripts.

6.1.17. SERVER_PROTOCOL

The SERVER_PROTOCOL meta-variable is set to the name and revision of the information protocol with which the request arrived. This is not necessarily the same as the protocol version used by the server in its response to the client.

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

'protocol' is a version of the <scheme> part of the
script-URI, but is not identical to it. For example, the
scheme of a request may be "https" while the protocol remains
"http". The protocol is not case sensitive, but by convention,
'protocol' is in upper case.

A well-known extension token value is "INCLUDED", which
signals that the current document is being included as part of
a composite document, rather than being the direct target of
the client request.

Servers MUST provide this meta-variable to scripts.

6.1.18. SERVER_SOFTWARE

The SERVER_SOFTWARE meta-variable is set to the name and
version of the information server software answering the
request (and running the gateway).

    SERVER_SOFTWARE = *CHAR

Servers MUST provide this meta-variable to scripts.

6.2. Request Content-Bodies

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

If the CONTENT_LENGTH value (see section 6.1.2) is non-NULL,
the server MUST supply at least that many bytes to scripts on
the standard input stream. Scripts are not obliged to read the
data. Servers MAY signal an EOF condition after CONTENT_LENGTH
bytes have been read, but are are not obligated to do so.
Therefore, scripts MUST NOT attempt to read more than
CONTENT_LENGTH bytes, even if more data are available.

For non-parsed header (NPH) scripts (see section 7.1 below),
servers 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.

7. 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 MUST always return some data. Unless defined otherwise, this will be via the 'standard output' file descriptor.

There are two forms of output that scripts can supply to servers: non-parsed header (NPH) output, and parsed header output. Servers MUST support parsed header output and MAY support NPH output. The method of distinguishing between the two types of output (or scripts) is implementation defined.

Servers MAY implement a timeout period within which data must be received from scripts. If a server implementation defines such a timeout and receives no data from a script within the timeout period, the server MAY terminate the script process and SHOULD abort the client request with a '500 Internal Server Error' response.

7.1. Non-Parsed Header Output

Scripts using the NPH output form MUST return a complete HTTP response message, as described in Section 6 of the HTTP specifications [3],[8]. NPH scripts MUST use the SERVER_PROTOCOL variable to determine the appropriate format for a response.

Servers SHOULD attempt to ensure that the script output is sent directly to the client, with minimal internal and no transport-visible buffering.

7.2. Parsed Header Output

Scripts using the parsed header output form MUST supply a CGI response message to the server as follows:

\[
\text{CGI-Response} = \star ( \text{CGI-field} \mid \text{HTTP-Field} ) \text{ NL } [ \text{Entity-Body} ] \\
\text{CGI-Field} = \text{Content-type} \\
\mid \text{Location} \\
\mid \text{Status} \\
\mid \text{extension-header}
\]

The response comprises a header and a body, separated by a blank line. The body may be NULL. The header fields are either
CGI header fields to be interpreted by the server, or HTTP header fields to be included in the response returned to the client if the request method is HTTP. At least one CGI-Field MUST be supplied, but no CGI field name may be used more than once in a response. If a body is supplied, then a "Content-type" header field MUST be supplied by the script, otherwise the script MUST send a "Location" or "Status" header field. If a Location CGI-field is returned, then the script MUST NOT supply any HTTP-Fields.

All header fields occurring in a CGI-Response MUST be specified one per line; CGI/1.1 makes no provision for continuation lines.

7.2.1. CGI header fields

The CGI header fields have the generic syntax:

\[
\text{generic-field} = \text{field-name} \:'' \text{field-value} \text{NL} \\
\text{field-name} = 1\ast<\text{any CHAR, excluding CTLs, SP and }''> \\
\text{field-value} = \ast( \text{field-content} | \text{LWSP} ) \\
\text{field-content} = \ast( \text{token} | \text{tspecial} | \text{quoted-string} )
\]

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

7.2.1.1. Content-Type

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

\[
\text{Content-Type} = "\text{Content-Type}" \:'' \text{media-type} \text{NL}
\]

This is actually an HTTP-Field rather than a CGI-field, but it is listed here because of its importance in the CGI dialogue as a member of the "one of these is required" set of header fields.

7.2.1.2. Location

This is used to specify to the server that the script is returning a reference to a document rather than an actual document.
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 '302 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

scheme "://" SERVER_NAME "::" SERVER_PORT rel-URL-abs-path

Note: If the request was accompanied by a content-body (such as for a POST request), the content-body will be lost if the script redirects the request with a Location field.

7.2.1.3. Status

The "Status" header field is used to indicate to the server what status code the server MUST use in the response message. It SHOULD NOT be sent if the script returns a "Location" header field.

Status = "Status" "":" digit digit digit 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 specifications [3]. If the SERVER_PROTOCOL is "HTTP/1.1", then the status codes defined in the HTTP/1.1 specification [8] may be used. If the script does not return a "Status" header field, then "200 OK" SHOULD be assumed by the server.
If a script is being used to handle a particular error or condition encountered by the server, such as a '404 Not Found' error, the script SHOULD use the "Status" CGI header field to propagate the error condition back to the client. E.g., in the example mentioned it SHOULD include a "Status: 404 Not Found" in the header data returned to the server.

7.2.1.4. Extension header fields

Scripts MAY include in their CGI response header additional fields not defined in this or the HTTP specification. These are called "extension" fields, and have the syntax of a generic-field as defined in section 7.2.1. The name of an extension field MUST NOT conflict with a field name defined in this or any other specification; extension field names SHOULD begin with "X-CGI-" to ensure uniqueness.

7.2.2. HTTP header fields

The script MAY return any other header fields defined by the specification for the SERVER_PROTOCOL (HTTP/1.0 [3] or HTTP/1.1 [8]). Servers MUST resolve conflicts between CGI header and HTTP header formats or names (see section 8).

8. Server Implementation

Coar, et al. INTERNET-DRAFT [Page 18]

CGI/1.1 Expires: 14 May, 1999

This section defines the requirements that must be met by HTTP servers in order to provide a coherent and correct CGI/1.1 environment in which scripts may function. It is intended primarily for server implementors, but it is useful for script authors to be familiar with the information as well.

8.1. Requirements for Servers

Servers MUST support the standard mechanism (described below) which allows script authors 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 meta-variables. This mechanism is as follows:

The server MUST translate the header data from the CGI header field syntax to the HTTP header field 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 header fields returned by the script and header fields that it would otherwise send itself. The fields affected and the resolution method used SHOULD be documented as part of the server implementation.

8.2. Recommendations for Servers

Servers SHOULD provide the "query" component of the script-URI as command-line arguments to scripts if it does not contain any unencoded '=' characters and the command-line arguments can be generated in an unambiguous manner. (See section 5.)

Servers SHOULD set the AUTH_TYPE meta-variable to the value of the 'auth-scheme' token of the "Authorization" field if it was supplied as part of the request header. (See section 6.1.1.)

Where applicable, servers SHOULD set the current working directory to the directory in which the script is located before invoking it.

Servers MAY reject with error '404 Not Found' 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:

1. define any restrictions on allowed characters, in particular whether ASCII NUL is permitted;
2. define any restrictions on allowed path segments, in particular whether non-terminal NULL segments are permitted;
3. 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];
4. define any limits of the implementation, including limits on path or search string lengths, and limits on the volume
of header data 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).

For non-parsed header (NPH) scripts (see section 7.1), servers 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.

8.3. Summary of Meta-Variables

Servers MUST provide the following meta-variables to scripts. See the individual descriptions for exceptions and semantics.

- CONTENT_LENGTH
- CONTENT_TYPE
- GATEWAY_INTERFACE
- PATH_INFO
- QUERY_STRING
- REQUEST_METHOD
- REMOTE_ADDR
- SCRIPT_NAME
- SERVER_NAME
- SERVER_PORT
- SERVER_PROTOCOL
- SERVER_SOFTWARE

Servers SHOULD define the following meta-variables for scripts. See the individual descriptions for exceptions and semantics.

- AUTH_TYPE
- REMOTE_HOST

In addition, servers SHOULD provide meta-variables for all fields present in the HTTP request header, with the exception of those involved with access control. Servers MAY at their discretion provide meta-variables for access control fields.

Servers MAY define the following meta-variables. See the individual descriptions for exceptions and semantics.

- PATH_TRANSLATED
- REMOTE_IDENT
Servers may at their discretion define additional implementation-specific extension meta-variables provided their names do not conflict with defined header field names. Implementation-specific meta-variable names SHOULD be prefixed with "X_" (e.g., "X_DBA") to avoid the potential for such conflicts.

9. Script Implementation

This section defines the requirements and recommendations for scripts that are intended to function in a CGI/1.1 environment. It is intended primarily as a reference for script authors, but server implementors should be familiar with these issues as well.

9.1. Requirements for Scripts

Scripts using the parsed-header method to communicate with servers MUST supply a response header to the server. (See section 7.)

Scripts using the NPH method to communicate with servers MUST provide complete HTTP responses, and MUST use the value of the SERVER_PROTOCOL meta-variable to determine the appropriate format. (See section 7.1.)

Scripts MUST check the value of the REQUEST_METHOD meta-variable in order to provide an appropriate response. (See section 6.1.13.)

Scripts MUST be prepared to handle URL-encoded values in meta-variables. In addition, they MUST recognise both "+" and "%20" in URL-encoded quantities as representing the space character. (See section 3.1.)

Scripts MUST ignore leading zeros in the major and minor version numbers in the GATEWAY_INTERFACE meta-variable value. (See section 6.1.4.)

When processing requests that include a content-body, scripts MUST NOT read more than CONTENT_LENGTH bytes from the input stream. (See sections 6.1.2 and 6.2.)

9.2. Recommendations for Scripts

Servers may interrupt or terminate script execution at any
time and without warning, so scripts SHOULD be prepared to
deal with abnormal termination.

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 a script is processing the output of a form , it SHOULD
verify that the CONTENT_TYPE is
"application/x-www-form-urlencoded" [2] or whatever other
media type is expected.

Scripts parsing PATH_INFO, PATH_TRANSLATED, or SCRIPT_NAME
SHOULD 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'.

As it is impossible for scripts to determine the client URI
that initiated a 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 header fields, scripts SHOULD try to send the
CGI header fields (see section 7.2) as soon as possible, and
preferably before any HTTP header fields. This may help reduce
the server's memory requirements.

10. System Specifications

10.1. AmigaDOS

The implementation of the CGI on an AmigaDOS operating system
platform SHOULD use environment variables as the mechanism of
providing request metadata to CGI scripts.

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 variable names and header field names;
the newline (NL) sequence is LF; servers SHOULD also
accept CR LF as a newline.

10.2. Unix

The implementation of the CGI on a UNIX operating system
platform SHOULD use environment variables as the mechanism of
providing request metadata to CGI scripts.

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 have any characters which are
'active' in the Bourne shell escaped with a backslash.
If the value of the QUERY_STRING meta-variable contains
an unencoded equals-sign '='; then the command line
SHOULD NOT be used by the script.

The current working directory
The current working directory for the script SHOULD be
set to the directory containing the script.

Character set
The US-ASCII character set is used for the definition
of environment variable names and header field names;
the newline (NL) sequence is LF; servers SHOULD also
accept CR LF as a newline.

11. Security Considerations

11.1. Safe Methods
As discussed in the security considerations of the HTTP specifications [3],[8], 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.

11.2. HTTP Header Fields Containing Sensitive Information

Some HTTP header fields 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 field containing a username and password. If the server, rather than the script, validates this information then the password SHOULD NOT be passed on to the script via the HTTP_AUTHORIZATION meta-variable without careful consideration. This also applies to the Proxy-Authorization header field and the corresponding HTTP_PROXY_AUTHORIZATION meta-variable.

11.3. 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.

12. Acknowledgements

This work is based on a draft published in 1997 by David R. Robinson in 1997, which in turn was 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 by Chris Adie, Dave Kristol, Mike Meyer, David Morris, and Harald Alvestrand.

13. References


CGI/1.1 Expires: 14 May, 1999


[10]

[11]

[12]

14. Authors' Addresses

Ken A L Coar
MeepZor Consulting
7824 Mayfair Crest Lane, Suite 202
Raleigh, NC 27615-4875
U.S.A.
Tel: +1 (919) 254.4237
Fax: +1 (919) 254.5250
Email: Ken.Coar@Golux.Com

David Robinson
Electronic Share Information Ltd
Mount Pleasant House
2 Mount Pleasant
Huntingdon Road
Cambridge CB3 0RN
UK
Tel: +44 (1223) 566926
Fax: +44 (1223) 506288
Email: drtr@esi.co.uk