Internet Engineering Task Force
INTERNET-DRAFT
draft-lennox-sip-cgi-01.ps

Individual Submission
J. Lennox, J. Rosenberg, H. Schulzrinne
Columbia U./Bell Labs
May 21, 1999
Expires: November 1999

Common Gateway Interface for SIP

Status of this Memo

Copyright Notice

Abstract

In Internet telephony, there must be a means by which new services are created and deployed rapidly. In the World Wide Web, the Common Gateway Interface (CGI) has served as popular means towards programming web services. Due to the similarities between the Session Initiation Protocol (SIP) and the Hyper Text Transfer Protocol (HTTP), CGI seems a good candidate for service creation in a SIP environment. This draft proposes a SIP-CGI interface for providing SIP services on a SIP server.

Contents

1 Introduction

In Internet telephony, there must be a means by which new services are created and deployed rapidly. In traditional telephony networks, this was accomplished through IN service creation environments, which provided an interface for creating new services, often using GUI based tools.

The WWW has evolved with its own set of tools for service creation. Originally, web servers simply translated URL's into filenames stored on a local system, and returned the file content. Over time, servers evolved to provide dynamic content, and forms provided a means for soliciting user input. In essence, what evolved was a means for service creation in a web environment. There are now many means for creation of dynamic web content, including server side JavaScript, servlets, and the common gateway interface (CGI) [1].

Multimedia communications, including Internet telephony, will also require a mechanism for creating services. This mechanism is strongly tied to the features provided by the signaling protocols. The Session Initiation Protocol (SIP) [2] has been developed for initiation and termination of multimedia sessions. SIP borrows heavily from HTTP, inheriting its client-server interaction and much of its syntax and semantics. For this reason, the web service creation environments, and CGI in particular, seem attractive as starting points for developing SIP based service creation environments.

2 Motivations

CGI has a number of strengths which make it attractive as an environment for creating SIP services:

Language independence: CGI works with perl, C, VisualBasic, tcl, and many other languages, as long as they support access to environment variables.

Exposes all headers: CGI exposes the content of all the headers in an HTTP request to the CGI application. An application can make use of these as it sees fit, and ignore those it doesn't care about. This allows all aspects of an HTTP request to be considered for creation of content. In a SIP environment, headers have greater importance than in HTTP. They carry critical information about the transaction, including caller and callee, subject, contact addresses, organizations, extension names, registration parameters and expirations, call status, and call routes, to name a few. It is therefore critical for SIP services to have as much access to these headers as possible. For this reason, CGI is very attractive.

Creation of Responses: CGI is advantageous in that it can create all parts of a response, including headers, status codes and reason phrases, in addition to message bodies. This is not the case for other mechanisms, such as Java servlets, which are focused primarily on the body. In a SIP environment, it is critical to be able to generate all aspects of a response (and, all aspects of new or proxied requests), since the body is usually not of central importance in SIP service creation.

Component Reuse: Many of the CGI utilities allow for easy reading of environment variables, parsing of form data, and often parsing and generation of header fields. Since SIP reuses the basic RFC822 [3] syntax of HTTP, many of these tools are applicable to SIP CGI.

Familiar Environment: Many web programmers are familiar with CGI.

Ease of extensibility: Since CGI is an interface and not a language, it becomes easy to extend and reapply to other protocols, such as SIP.

The generality, extensibility, and detailed control and access to information provided by CGI, coupled with the range of tools that exist for it, which can be immediately applied to SIP, make it a good mechanism for SIP service creation.

3 Differences from HTTP-CGI

While SIP and HTTP share a basic syntax and a request-response model, there are important differences. Proxies play a critical role in services for SIP, while they are less important for HTTP. SIP servers can fork requests (proxying multiple requests when a single request is received), an important capability absent from HTTP. SIP supports additional features, such as registrations, which are absent from HTTP. These differences are reflected in the differences between SIP CGI and HTTP CGI. SIP CGI runs primarily on proxy, redirect, and registrar servers, rather than user agent servers (which are the equivalent of origin servers in HTTP). SIP CGI allows the script to perform specific messaging functions not supported in HTTP CGI (such as proxying requests), and SIP CGI introduces a persistence model that allow a script to maintain control through multiple message exchanges. HTTP CGI has no persistence for scripts.

3.1 Basic Model

The basic model for HTTP-CGI is depicted in figure 1.

A client issues an HTTP request, which is passed either directly to the origin server (as shown), or is forwarded through a proxy server. The origin server executes a CGI script, and the CGI script returns a response, which is passed back to the client. The main job of the script is to generate the body for the response. Only origin servers execute CGI scripts, not proxy servers.

In a SIP server, the model is different, and is depicted in Figure 2.

The client generates a request, which is forwarded to a server. The server may generate a response (such as an error or redirect response). Or, if the server is a proxy server, the request is proxied to another

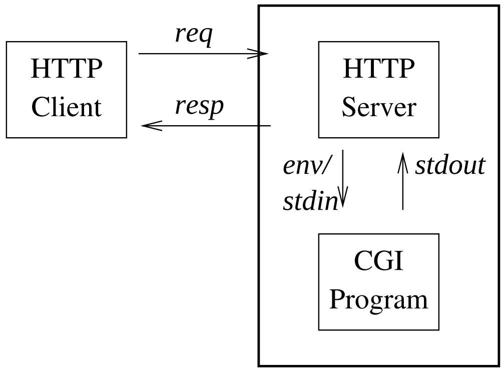

Figure 1: HTTP CGI Model

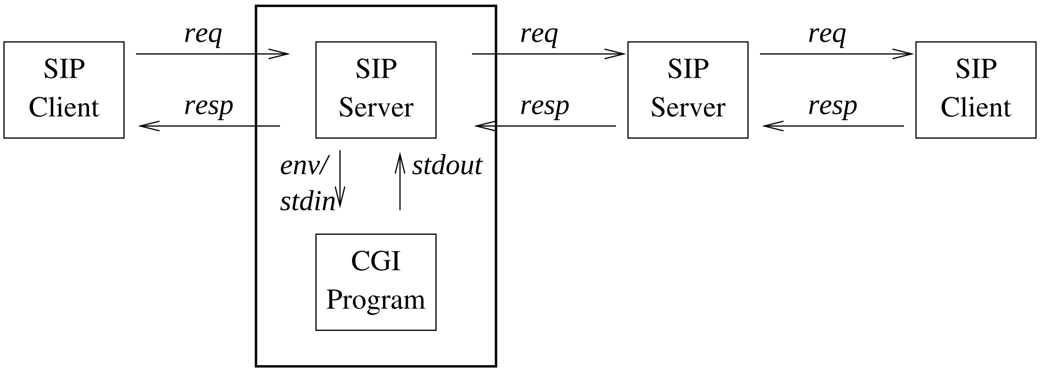

Figure 2: SIP CGI Model

server, and eventually to a user agent, and the response is passed back upstream, through the server, and back towards the client. A SIP proxy server may additionally fork requests, generating multiple requests in response to a received request. Generally, a proxy server will not generate the content in responses. These contain session descriptions created by user agents. Services, such as call forward and mobility services, are based on the decisions the server makes about (1) when, to where, and how many requests to proxy downstream, and (2) when to send a response back upstream. Creation of services such as ad-hoc bridging (where the server acts as a media mixer in a multiparty call, without being asked to do so by the end users) will require the server to generate new requests of its own, and for it to modify and generate the body in responses.

An HTTP server is mainly concerned about generation of responses. A SIP server is generally concerned about performing four basic operations:

Proxying of Requests: Receiving a request, adding or modifying any of the headers, deciding on a set of servers to forward the request to, and forwarding it to them.

Returning Responses: Receiving a response, adding or modifying any of the headers, and passing the response towards the client.

Generating Requests: Creating a new request, originating at the server, placing headers and a body into the message, and sending it to a server.

Generation of Responses: Receiving a request, generating a response to it, and sending it back to the client.

When a request is received, one or more of the above operations may occur at once. For example, a SIP server may generate a provisional response, generate a new request, and proxy the original request to two servers. This implies that SIP-CGI must encompass a greater set of functions than HTTP-CGI. These functions are a super-set of the simple end-server request/response model.

3.2 Persistence Model

In HTTP-CGI, a script is executed once for each request. It generates the response, and then terminates. There is no state maintained across requests from the same user, as a general rule (although this can be done — and is — for more complex services such as database accesses, which essentially encapsulate state in client-side cookies or dynamically-generated URLs). A transaction is just a single request, and a response.

In SIP-CGI, since a request can generate many new and proxied requests, these themselves will generate responses. A service will often require these responses to be processed, and additional requests of responses to be generated. As a result, whereas an HTTP-CGI script executes once per transaction, a SIP-CGI script must maintain control somehow over numerous events.

In order to enable this, and to stay with the original CGI model, we mandate that a SIP CGI script executes when a message arrives, and after generating output (in the form of additional messages), terminate. State is maintained by allowing the CGI to return an opaque token to the server. When the CGI script is called again for the same transaction, this token is passed back to the CGI script. When called for a new transaction, no token is passed.

For example, consider a request which arrives at a SIP server. The server calls a CGI script, which generates a provisional response and a proxied request. It also returns a token to the server, and then terminates. The response is returned upstream towards the client, and the request is proxied. When the response to the proxied request arrives, the script is executed again. The environment variables are set based on the content of the new response. The script is also passed back the token. Using the token as its state, the script decides to proxy the request to a different location. It therefore returns a proxied request, and another token. The server forwards this new request, and when the response comes, calls the CGI script once more, and passes back the token. This time, the script generates a final response, and passes this back to the server. The server sends the response to the client, destroys the token, and the transaction is complete.

3.3 SIP CGI Triggers

In many cases, calling the CGI script on the reception of every message is inefficient. CGI scripts come at the cost of significant overhead since they generally require creation of a new process. Therefore, it is important in SIP-CGI for a script to indicate, after it is called the first time, under what conditions it will be called for the remainder of the transaction. If the script is not called, the server will take the "default" action, as specified in this document. This allows an application designer to trade off flexibility for computational resources. Making an analogy to the Intelligent Network (IN) - a script is able to define the triggers for its future execution.

So, in summary, whereas an HTTP-CGI script executes once during a transaction, a single SIP-CGI script may execute many times during a transaction, and may specify at which points it would like to have control for the remainder of the transaction.

3.4 Naming

In HTTP-CGI, the CGI script itself is generally the resource named in the Request URI of the HTTP request. This is not so in SIP. In general, the request URI names a user to be called. The mapping to a script to be executed may depend on other SIP headers, including To and From fields, the SIP method, status codes, and reason phrases. As such, the mapping of a message to a CGI script is purely a matter of local policy administration at a server. A server may have a single script which always executes, or it may have multiple scripts, and the target is selected by some parts of the header.

3.5 Environment Variables

In HTTP-CGI, environment variables are set with the values of the paths and other aspects of the request. As there is no notion of a path in SIP, some of these environment variables do not make sense.

3.6 Timers

In SIP, certain services require that the script gets called not only when a message arrives, but when some timer expires. The classic example of this is "call-forward no answer." To be implemented with SIP-CGI, the first time the script is executed, it must generate a proxied request, and also indicate a time at which to be called again if no response comes. This kind of feature is not present in HTTP-CGI, and some rudimentary support for it is needed in SIP-CGI.

4 Overview of SIP CGI

When a request arrives at a SIP server, initiating a new transaction, the server will set a number of environment variables, and call a CGI script. The script is passed the body of the request through stdin.

The script returns, on stdout, a set of SIP action lines, each of which may be modified by CGI and/or SIP headers. This set is delimited by the same rules which delimit multiple SIP messages in a single UDP request - generally through the use of two carriage returns. The action lines allow the script to specify any of the four operations defined above, in addition to the default operation. Initiating a new request or generating a response is done by copying the request line of the request or the status line of the response, respectively, into an action line of the CGI output. For example, the following will create two new requests, in addition to responding with a 200 OK to the original request:

```
SIP/2.0 200 OK

INVITE sip:joe@sales.com SIP/2.0
Call-ID: 309hjfa0hs@122.8.34.66
CSeq: 0 INVITE
To: sip:joe@sales.com
From: sip:bob@sales.com

REGISTER sip:engineering.org SIP/2.0
Call-ID: 08h08gfa76g@111.22.33.4
```

```
CSeq: 1 REGISTER
To: sip:bill@engineering.org
From: sip:bill@engineering.org
Contact: mailto:bill@ieee.org
```

The operation of proxying a request is supported by the CGI-PROXY-REQUEST CGI action, which takes the URL to proxy to as an argument. For example, to proxy a request to dante@inferno.com:

```
CGI-PROXY-REQUEST-TO sip:dante@inferno.com SIP/2.0
Contact: sip:server1@company.com
```

In this example, the server will take the original request, and modify any header fields normally changed during the proxy operation (such as decrementing MaxForwards, and adding a Via field). This message is then "merged" with the output of the CGI script - SIP headers specified below the action line in the CGI output will be added to the outbound request. In the above example, the Contact header will be added. Note that the action line looks like the request line of a SIP request message. This is done in order to simplify parsing.

Returning of responses is more complex. A server may receive multiple responses as the result of forking a request. The script should be able to ask the server to return any of the responses it had received previously. To support this, the server will pass an opaque token to the script through environment variables, unique for each response received. To return a response, a CGI script needs to indicate which response is to be returned. For example, to return a response named with the token abcdefghij, the following output is generated:

```
CGI-FORWARD-RESPONSE abcdefghij SIP/2.0
```

Finally, the default action CGI action tells the server to do what it would normally do when receiving the message that triggered the script:

```
CGI-DEFAULT-ACTION dummy SIP/2.0
```

The SIP CGI script is not only executed when the original request arrives, but on the arrival of any responses due to proxied or new requests. A CGI script can change this default behavior through the CGI CGI-Reexecute-On header. Like SIP headers, CGI headers are written underneath the action line. They are extracted by the SIP server, and used to provide the server with additional guidance. CGI headers always begin with CGI to differentiate them from SIP headers. In this case, the header tells the server when to re-execute the script. Allowed values are never, always, and final-responses.

When the script is re-executed, it may need access to some state in order to continue processing. A script can generate a piece of state, called a cookie, for any new request or proxied request. It is passed to the server through the CGI-SET-COOKIE action. The action contains a token, which is the cookie itself. The server does not examine or parse the cookie. It is simply stored. When the script is re-executed, the cookie is passed back to the script through an environment variable.

```
CGI-SET-COOKIE khsihppii8asdl SIP/2.0
```

Finally, when the script causes the server to proxy a request, responses to these requests will arrive. To ease matching of responses to requests, the script can place a request token in the CGI CGI-Request-Token header. This header is removed by the server when the request is proxied. Any responses received to this request will have the token passed in an environment variable.

5 SIP CGI Specification

5.1 Introduction

5.1.1 Relationship with HTTP CGI

This SIP CGI specification is based on work-in-progress revision 1.1 of the HTTP CGI standard [1]. This document is a product of the informal CGI-WG, which is not an official IETF working group at this time. CGI-WG's homepage is located at the URL `http://Web.Golux.Com/coar/cgi/`, and the most recent versions of the CGI specification are available there. A number of sections of this document will refer to sections from the HTTP-CGI specification, as [HTTP-CGI:xx], rather than repeat information from that document verbatim.

5.1.2 Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [4] and indicate requirement levels for compliant SIP CGI implementations.

5.1.3 Specifications

The terms "system defined" and "implementation defined" are used to refer to functions and features of SIP CGI which are not defined in the main part of this specification. The definitions of these can be found in [HTTP-CGI:1.3].

5.1.4 Terminology

The terms "meta-variable," "script," and "server" are defined in [HTTP-CGI:1.4]. A "message" is a SIP request or response, typically either the one that triggered the invocation of the CGI script, or one that the CGI script caused to be sent.

5.2 Notational Conventions and Generic Grammar

In this specification we use the Augmented Backus-Naur Form notation described in RFC 2234 [5]. The basic rules described in [HTTP-CGI:2.2] are used to describe basic parsing constructs.

5.3 Invoking the script

The script is invoked in a system defined manner. Unless specified otherwise, the file containing the script will be invoked as an executable program.

Only one CGI script at a time may be outstanding for a SIP transaction. If subsequently arriving responses would cause a CGI script to be invoked, handling of them is deferred, except for ACK, until CGI scripts for previous messages in the transaction terminate. Messages are processed in the order they are received.

5.4 The SIP-CGI Script Command Line

The server SHOULD NOT provide any command line arguments to the script.

Command line arguments are used for indexed queries in HTTP CGI; HTTP indexed queries do not have an equivalent in SIP.

5.5 Data Input to the SIP-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.

5.5.1 Message Metadata (Meta-Variables)

Each SIP-CGI server implementation MUST define a mechanism to pass data about the message 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 or 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.

(This description of meta-variables is taken verbatim from [HTTP-CGI:6.1].)

The canonical meta-variables defined by this specification are:

Meta-variables also in HTTP-CGI:

```
AUTH_TYPE
CONTENT_LENGTH
CONTENT_TYPE
GATEWAY_INTERFACE
REMOTE_ADDR
REMOTE_HOST
REMOTE_IDENT
REMOTE_USER
REQUEST_METHOD
SERVER_NAME
SERVER_PORT
SERVER_PROTOCOL
SERVER_SOFTWARE
```

New meta-variables introduced in SIP-CGI:

```
REQUEST_URI
```

```
RESPONSE_STATUS
RESPONSE_REASON
RESPONSE_TOKEN
SCRIPT_COOKIE
REQUEST_TOKEN
```

Meta-variables with names beginning with the protocol name (e.g., "SIP_ACCEPT") are also canonical in their description of message header fields. The number and meaning of these fields may change independently of this specification. (See also section 5.5.1.5.)

The HTTP-CGI meta-variables PATH_INFO, PATH_TRANSLATED, QUERY_STRING, and SCRIPT_NAME are not meaningful in the SIP-CGI context, and are omitted from this specification.

A server MAY also set additional non-canonical meta-variables.

5.5.1.1 AUTH_TYPE See [HTTP-CGI:6.1.1]. The auth-scheme token can also be "PGP" corresponding to the authentication method detailed in the SIP specification.

For the complex authentication schemes, the server SHOULD perform the authentication checking itself. If the authentication failed, this meta-variable SHOULD NOT be set.

5.5.1.2 CONTENT_LENGTH See [HTTP-CGI:6.1.2]. Servers MUST provide this meta-variable to scripts if the message was a accompanied by a content-body entity, even if the message did not include a Content-Length header field.

5.5.1.3 CONTENT_TYPE See [HTTP-CGI:6.1.3].

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

```
GATEWAY_INTERFACE  =  "SIP-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 SIP-CGI/2.4 is a lower version than SIP-CGI/2.13 which in turn is lower than SIP-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 SIP-CGI interface ("SIP-CGI/1.1").

Servers MUST provide this meta-variable to scripts.

For maximal compatibility with existing HTTP-CGI libraries, we want to keep this as similar as possible to the syntax of CGI 1.1. However, we **do** want it to be clear that this is indeed SIP-CGI. Making HTTP-CGI's version identifier a substring of the SIP-CGI identifier seemed like a reasonable compromise. (The existing CGI libraries we checked do not seem to check the version.)

5.5.1.5 Protocol-Specific Metavariables These metavariables are specific to the protocol *via* which the method is sent. Interpretation of these variables depends on the value of the `SERVER_PROTOCOL` meta-variable (see section 5.5.1.19).

Meta-variables with names beginning with "SIP_" contain values from the message header, if the protocol used was SIP. They are constructed in the same manner (except for the meta-variable prefix) as HTTP-CGI's "HTTP_" headers; see [HTTP-CGI:6.1.5].

Servers are not required to create meta-variables for all the message header fields they receive; however, because of the relatively greater importance of headers in SIP, the server SHOULD provide all headers which are not either potentially sensitive authorization information, such as Authorization, or which are available via other SIP CGI variables, such as Content-Length and Content-Type.

> Note: these meta-variables' names were changed from `HTTP_*` to `SIP_*` since the first draft of this specification. The intention had been to make it easier to use existing CGI libraries unmodified, but this convenience was felt to be over-weighed by the confusion this introduced.

5.5.1.6 REMOTE_ADDR This is the IP address of the host sending the message to this server; see [HTTP-CGI:6.1.9]. This is not necessarily that of the originating client or user agent server.

For locally generated responses (see section 5.8), this should be the loopback address (e.g. 127.0.0.1 for IPv4).

5.5.1.7 REMOTE_HOST This is the fully qualified domain name of the host sending the message to this server, if available, otherwise NULL. See [HTTP-CGI:6.1.10].

5.5.1.8 REMOTE_IDENT The identity information supported about the connection by a RFC 1413 [6] request, if available; see [HTTP-CGI:6.1.11].

The server MAY choose not to support this feature, and it is anticipated that not many implementations will, as the information is not particularly useful in the presence of complex proxy paths.

5.5.1.9 REMOTE_USER If `AUTH_TYPE` was specified, this specifies the identity specified by that authorization information. See [HTTP-CGI:6.1.12].

> TBD: specify the syntax of this field for digest and pgp authentication.

5.5.1.10 REQUEST_METHOD If the message triggering the script was a request, the `REQUEST_METHOD` meta-variable is set to the method with which the request was made, as described in section 4.2 of the SIP/2.0 specification [2]; otherwise NULL.

```
REQUEST_METHOD    =  sip-method
sip-method        =  "INVITE" | "BYE" | "OPTIONS" | "CANCEL"
                     | "REGISTER" | "ACK"
                     | extension-method
extension-method  =  token
```

Note that ACK is usually not appropriate for the SIP-CGI/1.1 environment; however, see section 5.11. The implications of REGISTER in the CGI context are discussed in section 5.9, and CANCEL is discussed in section 5.10. A SIP-CGI/1.1 server MAY choose to process some methods directly rather than passing them to scripts.

Servers MUST provide this meta-variable to scripts if the triggering message was a request.

5.5.1.11 REQUEST_TOKEN

```
REQUEST_TOKEN = token
```

If the script specified a request token in a proxied request, this token is returned to the server in responses to that request. Note that this token is chosen by the script, not by the server. Each response to a proxied request contains the same value for this token.

5.5.1.12 REQUEST_URI

This meta-variable is specific to requests made with SIP.

```
REQUEST_URI = SIP-URL ; SIP-URL is defined in
                      ; section 2 of [2].
```

If the message triggering the script was a request, this variable indicates the URI specified with the request method. This meta-variable is only present if `REQUEST_METHOD` is non-NULL; in that case, servers MUST provide it to scripts.

> This meta-variable fills the roles of HTTP-CGI's `SCRIPT_NAME`, `PATH_INFO`, and `QUERY_STRING`.

5.5.1.13 RESPONSE_STATUS

```
RESPONSE_STATUS = Status-Code ; Status-Code is defined in
                              ; section 5.1.1 of [2].
```

If the message triggering the script was a response, this variable indicates the numeric code specified in the response; otherwise it is NULL. In the former case, servers MUST provide this meta-variable to scripts.

5.5.1.14 RESPONSE_REASON

```
RESPONSE_REASON = Reason-Phrase ; Reason-Phrase is defined in
                                ; section 5.1.1 of [2].
```

If the message triggering the script was a response, this variable indicates the textual string specified in the response.

5.5.1.15 RESPONSE_TOKEN

```
RESPONSE_TOKEN = token
```

If the message triggering the script was a response, the server MAY specify a token which subsequent invocations of the CGI script can use to identify this response. This string is chosen by the server and is opaque to the CGI script. See the discussion of CGI-FORWARD-RESPONSE in section 5.6.1 below.

5.5.1.16 SCRIPT_COOKIE

SCRIPT_COOKIE = token

This is the value an earlier invocation of this script for this transaction passed to the server in CGI action line CGI-SET-COOKIE. See the description of that action in section 5.6.1.5 below.

5.5.1.17 SERVER_NAME See [HTTP-CGI:6.1.15].

5.5.1.18 SERVER_PORT See [HTTP-CGI:6.1.16].

5.5.1.19 SERVER_PROTOCOL The SERVER_PROTOCOL meta-varaible is set to the name and revision of the protocol with which the message arrived; see [HTTP-CGI:6.1.17]. This will usually be “SIP/2.0”.

5.5.1.20 SERVER_SOFTWARE See [HTTP-CGI:6.1.18].

5.5.2 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 5.5.1.2) 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 not obligated to do so. Therefore, scripts MUST NOT attempt to read more than CONTENT_LENGTH bytes, even if more data are available.

5.6 Data Output from the SIP-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.

Servers MAY implement a timeout period within which data must be received from scripts, a maximum number of requests or responses that a particular CGI script can initiate, a maximum total number of requests or responses that can be sent by scripts over the lifetime of a transaction, or any other resource limitations it desires. If a script exceeds one of these limitations, the server MAY terminate the script process and SHOULD abort the transaction with either a ‘504 Gateway Timed Out’ or a ‘500 Internal Server Error’ response.

A SIP CGI script’s output consists of any number of messages, each corresponding to actions which the script is requesting that the server perform. Messages consist of an action line, whose syntax is specific to the type of action, followed by CGI header fields and SIP header fields. Action lines determine the nature of the action performed, and are described in section 5.6.1. CGI header fields pass additional instructions or information to the server, and are described in section 5.6.2.

A message MUST contain exactly one action line, and MAY also contain any number of CGI header fields and SIP header fields, and MAY contain a SIP body.

All header fields (both SIP and CGI) occurring in an output message MUST be specified one per line; SIP-CGI/1.1 makes no provision for continuation lines.

The generic syntax of CGI header fields is specified in [HTTP-CGI:8.2].

A server MAY choose to honor only some of the requests or responses; in particular, it SHOULD NOT accept any responses following a Status message which sends a definitive non-success response.

The messages sent by a script are delimited as follows:

1. A message begins with an action line.
2. If the message does not contain a Content-Type header field, or if it contains the header field `"Content-Length: 0"`, then it is terminated by a blank line.
3. If the message contains both Content-Type and Content-Length header fields, the message has a body consisting of the Content-Length octets following the blank line below the set. The next message begins after the body (and optionally some number of blank lines). If the script closes its output prematurely, the server SHOULD report a 500-class server error.
4. If the message contains Content-Type but not Content-Length, the message's body similarly begins with the blank line following the set; this body extends until the script closes its output. In this case, this is necessarily the last message the script can send. The server SHOULD insert a Content-Length header containing the amount of data read before the script closed its output.
5. If a message contains a non-zero Content-Length but does not contain a Content-Type, it is an error. The server SHOULD report a 500-class server error.

The output of a SIP-CGI script is intended to be syntactically identical to that of a UDP packet in which multiple requests or responses are sent, so that the same message parser may be used.

5.6.1 CGI Action Lines

5.6.1.1 Status

Status = SIP-Version 3*digit SP reason-phrase NL

This action line causes the server to generate a SIP response and relay it upstream towards the client. The server MUST copy the To, From, Call-ID, and CSeq headers from the original request into the response if these headers are not specified in the script output. The server SHOULD copy any other headers from the request which would normally be copied in the response if these are not specified in the script output.

For compatibility with HTTP-CGI, a server MAY interpret a message containing a Content-Type header field and no action line as though it contained "`SIP/2.0 200 OK`". This usage is deprecated.

5.6.1.2 Proxy Request

Proxy-Request-To = "CGI-PROXY-REQUEST" SIP-URI SIP-Version

This action line causes the server to forward the given request to the specified SIP URI. It may be sent either by a script triggered by a request, or by a script triggered by a response on a server which is running statefully and remembers the original request.

Any SIP header field MAY be specified below the action line. Specified SIP headers replace all those in the original message in their entirety; if a script wants to preserve header elements from the original message as well as adding new ones, it can concatenate them by the usual rules of header concatenation, and place the result in the script output. New header fields are added to the message after any Via headers but before any other headers.

Any headers from the original request which are not generated by the CGI script are copied into the proxied request, after modifications normally performed by a proxy server. In particular, the server MUST append a Via field and decrement MaxForwards. A server MAY perform additional modifications as it sees fit, such as adding a Record-Route header. A server SHOULD NOT append these headers if they are specified in the script output.

A script MAY specify that a SIP header is to be deleted from the message by specifying a field name without a field body, as in

```
Subject:
```

If the message does not specify a body, the body from the initial request is used. A message with "`Content-Length: 0`" is specifying an empty body; this causes the body to be deleted from the message.

If the initial request was authenticated by any means other than 'basic,' the script SHOULD NOT add, change, or remove any end-to-end headers, as this would break the authentication.

5.6.1.3 Forward Response

Forward-Response	=	"CGI-FORWARD-RESPONSE" Response-Name SIP-Version
Response-Name	=	response-token | "this"

This action line causes the server to forward a response on to its appropriate final destination. The same rules apply for accompanying SIP headers and message bodies as for CGI-PROXY-REQUEST.

The specified response name may either be a response token the server previously submitted in a RESPONSE_TOKEN meta-variable, or the string "this." The string "this" may only be sent if the message which triggered this CGI script was a response; it indicates that this triggering response should be forwarded.

5.6.1.4 Initiate Request

Initiate-Request = sip-method sip-uri SIP-Version

This action line causes the server to initiate a new SIP request, with the specified method, to the specified URI. Any SIP header MAY be included in the request. The script MUST place those headers in the message which are mandatory according to the rules in [2].

The SIP server SHOULD only allow appropriate SIP methods to be sent. In particular, unknown SIP methods and script-generated ACK messages SHOULD NOT be sent.

5.6.1.5 Script Cookie

Script-Cookie = "CGI-SET-COOKIE" token SIP-Version

This action line causes the server to store a script cookie, passed as a token in the action line. Subsequent messages received by the server which cause script execution carry the token in a meta-header. This includes responses to proxied requests and new requests initiated by the script. The script can alter the value of the cookie by subsequent script cookie actions. This alteration will take affect for all subsequent script invocations.

5.6.1.6 Default Action

Default-Action = "CGI-DEFAULT-ACTION" "dummy" SIP-Version

This action line tells the server to execute the default action at this point in the transaction. The default actions depend on the event which triggered the script:

Request received: When the request is first received, the default action of the server is to check the registration database against the request, and either proxy or redirect the request based on the action specified by the user agent in the registation.

Proxied response received: If a response is received to a proxied request, the server forwards the response towards the caller if the response was a 200 or 600 class response, and sends a CANCEL on all pending branches. If the response was informational, the state machinery for that branch is updated, and the response is not proxied upstream towards the caller. For 300, 400, and 500 class responses, an ACK is sent, and the response is forwarded upstream towards the caller if all other branches have terminated, and the response is the best received so far. If not all branches have terminated, the server does nothing. If all branches have terminated, but this response is not the best, the best is forwarded upstream. This is the basic algorithm outlined in the SIP specification.

Generated Response Received: If the original CGI script generated its own request, and a response arrives, the default action is to ACK the response if it is INVITE, otherwise nothing is done.

This action MUST NOT be combined with any SIP headers, or any CGI headers except Request-Token. If the default action is specified in the script output, the script will not be re-executed on any subsequent responses.

Open issue: CGI script does proxy encription, Max-Forwards, etc., but doesn't want to handle the contact database. Should this be SIP headers accompanying this CGI request?

If a SIP CGI script produces no output or no definitive response before closing its communication channel, the server SHOULD assume this action.

5.6.2 CGI Header fields

CGI header fields syntactially resemble SIP header fields, but their names all begin with the string "CGI-". The SIP server MUST strip all CGI header fields from any request before sending it, including those it does not recognize.

5.6.2.1 Reexecute-On

```
Reexecute-On          =  "CGI-Reexecute-On" ":" reexecute-condition
reexecute-condition   =  "all" | "final-responses" | "never"
                         | token
```

This CGI header allows the script to inform the server of the conditions upon which the server should re-execute the script or take the default action. The keyword "all" means that the script would like to be re-executed when any response for the proxied request or new request, to which the CGI Header is attached, is received. The keyword "final-responses" means that the script would like to be invoked upon any final responses (all but 1xx) to the proxied request or new request to which the CGI header is attached. The keywork "never" means the script should not be executed again. If there are multiple messages in the script output, and any one of them contains the keywork "never", the script will never be executed again for any responses from any proxied or new request. When subsequent responses arrive, the default action is always taken.

A script is never reexecuted when a retransmitted request or retransmitted response is received. Note that since any request for the same transaction is by definition a retransmission, a script is called for the first time due to a request, but always by a response for subsequent invocations. A script is never executed when an ACK is received.

5.6.2.2 Request-Token

Request-Token = "CGI-Request-Token" ":" token

To assist in matching responses to requests, the script can place a CGI-Request-Token CGI header in a CGI-PROXY-REQUEST or new request. This header contains a token, opaque to the server. When a response to this request arrives, the token is passed back to the script as a meta-header.

> This allows scripts to "fork" a proxy request, and correlate which response corresponds to which branch of the request.

5.7 Local expiration handling

If a CGI script specifies an Expires header field along with a new request or CGI-PROXY-REQUEST, the SIP server SHOULD track the expiration timeout locally as well as sending the message to the remote server. When the timeout expires, the server SHOULD generate a "408 Request Timeout" response. The timeout response SHOULD be handled as specified in section 5.8. At the time the request is timed out, the server SHOULD also transmit CANCEL messages for the request.

> This allows a SIP-CGI script in a proxy server to implement services like "Call Forward No Answer" to trigger after a user-determined time, even if the remote user-agent server is not responding or does not properly handle the Expires header field.

> It might be better to separate this functionality with a CGI-Expires or CGI-Timeout CGI header field.

5.8 Locally generated responses

In a proxy environment, locally generated responses such as "408 Request Timeout" SHOULD be sent to the CGI script in the same manner as received messages are. However, messages which merely report a problem with a message, such as "400 Bad Request", SHOULD NOT be.

> This is the other half of the requirements for the implementation of the "Call Forward No Answer" service, along with the local handling of the Expires header.

5.9 SIP-CGI and REGISTER

The specific semantics of a SIP-CGI script which is triggered by a REGISTER request are somewhat different than that of those triggered by call-related requests; however, allowing user control of registration may in some cases be useful. The two specific actions for REGISTER that need to be discussed are the response “200” and CGI-DEFAULT-ACTION. In the former case, the server SHOULD assume that the CGI script is handling the registration internally, and SHOULD NOT add the registration to its internal registration database; in the latter case, the server SHOULD add the registration to its own database. The server also SHOULD NOT add the registration if a 3xx, 4xx, 5xx, or 6xx status was returned, or if the registration request was proxied to another location.

5.10 SIP-CGI and CANCEL

SIP-CGI servers SHOULD execute scripts when a CANCEL message is received. The script SHOULD clean up any state it has for the transaction, and return an appropriate response (normally 200 or CGI-DEFAULT-ACTION) as quickly as possible.

Question: should a CANCEL also terminate currently-executing scripts?

5.11 SIP-CGI and ACK

5.11.1 Receiving ACK’s

Under normal circumstances, if the server receives an ACK, the script is not re-executed. If the ACK is destined for the proxy (acknowledging a 300, 400, 500, or 600 response), the ACK causes response retransmissions to cease. If the ACK is for a 200 response forwarded from a downstream server, the ACK is proxied downstream.

However, if the script generated its own 200 response to an INVITE request, the script SHOULD be re-executed with the ACK message. This is necessary in cases where the script is causing the proxy to act as a UAS. ACK messages can contain bodies, and would therefore be useful to the script.

5.11.2 Sending ACK’s

When the server receives a non-200 final response to an INVITE request, it SHOULD generate an ACK on its own, and not depend on the script to do so. There is no way in SIP CGI 1.1 to override this behavior. However, since the server will not generate an ACK for 200 responses to INVITE, a script causing the server to act as a UAC MUST generate ACK’s for them.

6 Security Considerations

6.1 Request initiation

CGI scripts are able to initiate arbitrary SIP transactions, or to produce spoofed responses of any sort. This protocol does not attempt to restrict the actions CGI scripts can take. Server administrators MUST consider CGI scripts to be as security-sensitive as their SIP server itself, and perform equivalent levels of security review before installing them.

6.2 Authenticated and encrypted messages

CGI scripts must be careful not to interfere with authentication. In particular, adding or removing header fields that are below the Authorization header will cause the message to fail authentication at the user agent.

When a SIP request is encrypted, the headers which are in the clear are passed to the server according to this specification. The encrypted portion of the request is not passed to the script. Any SIP headers output by the script will be added to the message. However, scripts should be aware that these may be discarded if they also exist within the encrypted portion.

6.3 SIP header fields containing sensitive information

Some SIP 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.

6.4 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.

6.5 Data Length and Buffering Considerations

This specification places no limits on the length of entity bodies presented to the script. Scripts SHOULD NOT assume that statically allocated buffers of any size are sufficient to contain the entire submission at one time. Use of a fixed length buffer without careful overflow checking may result in an attacker exploiting 'stack-smashing' or 'stack-overflow' vulnerabilities of the operating system. Scripts may spool large submissions to disk or other buffering media, but a rapid succession of large submissions may result in denial of service conditions. If the CONTENT_LENGTH of an entity-body is larger than resource considerations allow, scripts SHOULD respond with '413 Request Entity Too Large.'

7 Changes from earier versions

7.1 Changes from draft 00

- All HTTP-CGI references were updated to refer to the current version of that specification (draft-coar-cgi-v11-03). Also, some sections were reorganized to mirror the ordering of that draft, and some wording which had been taken largely verbatim from that draft was updated to reflect updates to it.
- The protocol-specific meta-variables were renamed from HTTP_* to SIP_*.
- MUST stipulations were added to some required meta-variables.
- The grammar of RESPONSE_TOKEN and SCRIPT_COOKIE was simplified to just "token", rather than "*qchar".

- Script cookies were changed from a CGI header to an action. There can only be one script cookie outstanding for a script at any time.
- Request tokens were added to let scripts track script branching.
- Discussions of CGI handling of ACK and CANCEL requests were added.
- All references to draft-ietf-mmusic-sip were changed to references to RFC 2543.
- More security considerations were added.

8 Acknowledgements

This work draws extremely heavily upon the HTTP-CGI [1] specification, including directly copying wording, with minimal editing, in a number of places.

9 Full Copyright Statement

10 Authors' Addresses

Jonathan Lennox
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA
electronic mail: lennox@cs.columbia.edu

Jonathan Rosenberg
Rm. 4C-526
Bell Laboratories, Lucent Technologies
101 Crawfords Corner Rd.
Holmdel, NJ 07733
USA
electronic mail: jdrosen@bell-labs.com

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA
electronic mail: schulzrinne@cs.columbia.edu

References

[1] K. Coar and D. Robinson, "The WWW common gateway interface version 1.1," Internet Draft, Internet Engineering Task Force, Apr. 1999. Work in progress.

[2] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments (Proposed Standard) 2543, Internet Engineering Task Force, Mar. 1999.

[3] D. Crocker, "Standard for the format of ARPA internet text messages," Request for Comments (Historic) 822, Internet Engineering Task Force, Aug. 1982.

[4] S. Bradner, "Key words for use in RFCs to indicate requirement levels," Request for Comments (Best Current Practice) 2119, Internet Engineering Task Force, Mar. 1997.

[5] D. Crocker, Ed., and P. Overell, "Augmented BNF for syntax specifications: ABNF," Request for Comments (Proposed Standard) 2234, Internet Engineering Task Force, Nov. 1997.

[6] M. StJohns, "Identification protocol," Request for Comments (Proposed Standard) 1413, Internet Engineering Task Force, Jan. 1993.