Internet Engineering Task Force
INTERNET-DRAFT
draft-lennox-sip-cgi-00.ps
J. Lennox, J.Rosenberg, H.Schulzrinne
Columbia U./Bell Labs
October 16, 1998
Expires: April 1999

Common Gateway Interface for SIP

Status of this Memo

Copyright Notice

Abstract

In an Internet telephony environment, it is critical to have a means by which new services are created and deployed rapidly and efficiently. In the web world, the Common Gateway Interface (CGI) has served as popular means towards programming web services. Due to the similarities between SIP and 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 an Internet telephony environment, it is critical to have a means by which new services are created and deployed rapidly and efficiently. In traditional telephony networks, this was accomplished through IN service creation environments, which provided an interface for manipulating objects and relationships related to telephony.

The Internet has provided new fora for creation of services. The web has become one of the most important applications on the Internet. Originally, web servers simply translated URL's into files stored on a local system, and returned the file as 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. Each of these have different strengths and weaknesses for creation of dynamic content.

Internet telephony lies at the crossroads of telephony and the web, and presents unique challenges for service creation. Creation of services in Internet telephony is strongly tied to the services provided by the signaling protocol. The Session Initiation Protocol (SIP) [1] has been developed for initiation and termination of multimedia sessions, including Internet telephony. SIP borrows heavily from HTTP, inheriting its client-server interation 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.

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 [2] syntax of HTTP, all of these tools are immediately 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 an good mechanism for SIP service creation.

3 Differences from HTTP-CGI

Certainly, SIP is different from HTTP. A SIP server does different things than a web server. As such, SIP-CGI must build upon the basic HTTP-CGI.

3.1 Basic Model

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

```
 ~~~~~~~~   req  -------
|        |------|       |
| client | resp | server|
|        |------|       |
 ~~~~~~~~        -------
                  |   | CGI
                  |   |
                 -------
                |       |
                |  CGI  |
                | prog. |
                |       |
                 -------
```

Figure 1: HTTP CGI Model

A client issues an HTTP request, the server executes a CGI script, and the CGI script returns a response, which is forwarded to the client. The main job of the script is to generate the body for the response.

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

```
 ~~~~~~~~   req  -------   req   -------     req   ~~~~~~~~
|        |------|       |-------|       |---------|        |
| client | resp | server| resp  | server| resp    | client |
|        |------|       |-------|       |---------|        |
 ~~~~~~~~        -------         -------           --------
                  |   | CGI
                  |   |
                 -------
                |       |
                |  CGI  |
                | prog. |
                |       |
                 -------
```

Figure 2: SIP CGI Model

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 server, and eventually to a client, 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 will require the server to generate new requests of its own, and for it to modify and generate content in responses.

So, in HTTP, the server is mainly concerned about generation of responses, a SIP server is mainly concerned about proxying of requests, proxying of responses, generation of responses, and generation of requests. Furthermore, a single request may generate, simultaneously, multiple responses, proxied requests, and new requests. This implies that SIP-CGI must encompass a greater set of functions than in HTTP-CGI. When a request arrives at a server, the CGI script is executed, and must be able to simultaneously cause requests to be created and proxied, and responses to be created and forwarded upstream. These functions are a super-set of the simple end-server request/response model, which means SIP-CGI may be designed as a backward-compatible extension of HTTP-CGI.

3.2 Time of Execution

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. It then terminates. The response is sent upstream, 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.

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.

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.

3.3 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.

3.4 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.

3.5 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 SIP CGI Specification

4.1 Introduction

4.1.1 Relationship with HTTP CGI

This SIP CGI specification is based on work-in-progress revision 1.1 of the HTTP CGI standard [3]. 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.

4.1.2 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.

4.1.3 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].

4.1.4 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.

4.2 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.

4.3 Message Metadata (Meta-Variables)

Each SIP-CGI 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.

(This description of meta-variables is taken verbatim from [HTTP-CGI:4].)

The meta-variables from HTTP-CGI are:

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

The new variables introduced in SIP-CGI are:

```
REQUEST_URI
RESPONSE_STATUS
RESPONSE_REASON
SCRIPT_COOKIE
```

The HTTP-CGI 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 any additional meta-variables it chooses.

4.3.1 AUTH_TYPE

See [HTTP-CGI:4.1]. The auth-scheme token can also be digest or pgp corresponding to the authentication methods 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.

4.3.2 CONTENT_LENGTH

See [HTTP-CGI:4.2]. If the message contains a body, this meta-variable MUST be set set even if a Content-Length header field was not included in the message.

4.3.3 CONTENT_TYPE

See [HTTP-CGI:4.3].

4.3.4 GATEWAY_INTERFACE

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

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

Note that the major and minor numbers are treated as separate integers and hence each may be incremented higher 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 MUST be ignored by scripts and SHOULD NOT be generated by servers.

This document defines the 1.1 version of the SIP-CGI interface.

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.)

XXX: check more than just Perl.

4.3.5 HTTP_*

These meta-variables encode the header data of the message; see [HTTP-CGI:4.5]. The server is not required to create meta-variables for all the header fields it receives; 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.

The variable names are specified as HTTP_* rather than SIP_* in order to make it easier to use existing CGI libraries unmodified.

4.3.6 REMOTE_ADDR

This is the IP address of the host sending the message to this server; see [HTTP-CGI:4.9]. This is not necessarily that of the originating client or user agent server.

For locally generated responses (see section 4.7), this should be the loopback address (i.e. 127.0.0.1 for IPv4).

4.3.7 REMOTE_HOST

This is the hostname of the host sending the message to this server. See [HTTP-CGI:4.10].

4.3.8 REMOTE_IDENT

The identity information supported about the connection by a RFC 1413 [6] request, if available; see [HTTP-CGI:4.11].

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.

4.3.9 REMOTE_USER

If AUTH_TYPE was specified, this specifies the identity specified by that authorization information. See [HTTP-CGI:4.12].

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

4.3.10 REQUEST_METHOD

If the message triggering the script was a request, the method with which the request was made, as described in section 4.2 of the SIP/2.0 specification [1]; otherwise NULL.

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

Note that ACK is not appropriate for the SIP-CGI/1.1 environment. The implications of REGISTER in the CGI context are discussed in section 4.8.

4.3.11 REQUEST_URI

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

REQUEST_URI = SIP-URL ; SIP-URL is defined in section 2 of [1].

If the message triggering the script was a request, this variable indicates the URI specified with the request method. This variable is only present if `REQUEST_METHOD` is non-NULL.

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

4.3.12 RESPONSE_STATUS

RESPONSE_STATUS = Status-Code ; Status-Code is defined in
; section 5.1.1 of [1].

If the message triggering the script was a response, this variable indicates the numeric code specified in the response.

4.3.13 RESPONSE_REASON

RESPONSE_REASON = Reason-Phrase ; Reason-Phrase is defined in
; section 5.1.1 of [1].

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

4.3.14 SCRIPT_COOKIE

SCRIPT_COOKIE = *qchar

This is the value the script passed to the server after an earlier message in this transaction in the optional CGI header Script-Cookie. See the description of that header in section 4.6.2 below.

4.3.15 SERVER_NAME

See [HTTP-CGI:4.15].

4.3.16 SERVER_PORT

See [HTTP-CGI:4.16].

4.3.17 SERVER_PROTOCOL

The name and revision of the protocol with which the message arrived; see [HTTP-CGI:4.17]. This will usually be "SIP/2.0".

4.3.18 SERVER_SOFTWARE

See [HTTP-CGI:4.16].

4.4 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.

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.

4.5 Data Input to the SIP-CGI Script

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

There MUST be at least `CONTENT_LENGTH` bytes available for the script to read if `CONTENT_LENGTH` is not NULL. The script is not obliged to read the data, but it MUST NOT attempt to read more than `CONTENT_LENGTH` bytes, even if more data are available.

4.6 Data Output from the SIP-CGI Script

A SIP CGI's output consists of any number of messages, each corresponding to actions which the script is requesting that the server perform. Messages can contain three kinds of header fields: primary CGI header fields, secondary CGI header fields, and SIP header fields. Primary CGI header fields determine the nature of the action performed, and are described in section 4.6.1. Secondary CGI header fields pass additional instructions or information to the server, and are described in section 4.6.2.

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

All header fields 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 response.

The messages sent by a script are delimited as follows:

1. 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.

2. 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.

3. 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.

4. If a message contains a non-zero Content-Length but does not contain a Content-Type, it is an error.

4.6.1 Primary CGI Headers

Status

Status = "Status" ":" 3*digit SP reason-phrase NL

This primary CGI header causes the server to generate a SIP response and relay it in the "appropriate" direction: either responding to a request, or forwarding on in place of the triggering response. If SIP headers are specified in the associated message, they replace whatever default header the server would have otherwise generated.

For compatibility with HTTP-CGI, a server SHOULD interpret a header set containing a Content-Type header field and no primary CGI header field as though it contained `"Status: 200 OK".` This usage is deprecated.

Proxy-Request-To

Proxy-Request-To = "Proxy-Request-To" ":" SIP-URI

This primary CGI header 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 in the associated message. Specified SIP headers replace all those in the original message in their entirety; if a script wants to preserve headers from the original message as well as adding new ones, it can concatenate them by the usual rules of header concatenation. New header fields are added to the message after any Path headers but before any other headers.

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.

Forward-Response

Forward-Response = "Forward-Response" ":" "yes"

This primary CGI header causes the server to forward the triggering response on to its appropriate final destination. It may only be sent by a script triggered by a response. The same rules apply for accompanying SIP headers and message bodies as for Proxy-Request-To.

No other value than "yes" is allowed for this field. A value is only included for syntactic consistency.

Initiate-Request

Initiate-Request = "Initiate-Request" ":" sip-method sip-uri

This primary CGI header 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.

Default-Action

Default-Action = ”Default-Action” ”:” ”yes”

This primary CGI header tells the server to execute the default action at this point in the transaction. The default actions are:

Request received: When the request is first received, the default action of the server is 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 header MUST NOT be combined with any other headers except Script-Cookie.

If a SIP CGI script produces no output at all before closing its communication channel, the server SHOULD assume this primary action.

4.6.2 Secondary CGI Headers

Script-Cookie

Script-Cookie = ”Script-Cookie” ”:” $<$ ” $>$ *qchar $<$ ” $>$
| ”Script-Cookie” ”:”

This secondary CGI header allows the script to pass a quoted-string to the server. If the header had a value, subsequent requests on this transaction branch will have the SCRIPT_COOKIE meta-variable set. Sending this secondary header without a value will cause subsequent transactions not to have the variable set; this is useful to clear a script cookie from a transaction.

This allows a SIP CGI script to retain state across multiple invocations in a complex transaction.

ReExecute-On

ReExecute-On = ”ReExecute-On” ”:” (”all” | ”final-responses” | ”never”)

This secondary 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 transaction is received. The keyword “final-responses” means that the script would like to be invoked upon any final responses (all but 1xx). The keywork “never” means the script should not be executed again. In cases where the script is not executed, the default action is taken.

Timeout

Timeout = "Timeout" ":" (HTTP-date | delta-seconds)

This secondary CGI header allows a script to request that the server lower its timeout threshold for a Proxy-Request-To or New-Request message. Servers SHOULD, however, disregard any attempt to raise the timeout limit.

This sets the threshhold only for the immediate timeout of the request. Any subsequent timeouts have their default value unless this header is also specified in subsequent script invocations. The default SIP timeouts are given in section 10 of [1].

When a timeout expires, the CGI script is invoked with the locally-generated “408 Request Timeout” response. This occurs whether the timeout was explicitly set or had its default value. See section 4.7.

This CGI header should not be confused with the SIP header Expires.

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.

4.7 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 Timeout header.

4.8 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 primary header return values for REGISTER that need to be discussed are “Status: 200” and “Default-Action: yes”. 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 Security Considerations

TODO

6 Full Copyright Statement

7 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] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Internet Draft, Internet Engineering Task Force, Sept. 1998. Work in progress.

[2] D. Crocker, "Standard for the format of ARPA internet text messages," Request for Comments (Standard) STD 11, 822, Internet Engineering Task Force, Aug. 1982. (Obsoletes RFC733); (Updated by RFC987); (Updated by RFC1327).

[3] D. Robinson and K. Coar, "The WWW common gateway interface version 1.1," Internet Draft, Internet Engineering Task Force, May 1998. Work in progress.

[4] S. Bradner, "Key words for use in RFCs to indicate requirement levels," BC 2119, Internet Engineering Task Force, Mar. 1997.

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

[6] M. S. Johns, "Identification protocol," Request for Comments (Proposed Standard) 1413, Internet Engineering Task Force, Feb. 1993. (Obsoletes RFC931).