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

Individual Submission
Lennox/Schulzrinne
Columbia University

October 31, 2000
Expires: April 2001

Transporting User Control Information in SIP REGISTER Payloads

Status of this Memo

Copyright Notice

Abstract

Several newly developed languages and interfaces, such as the CPL and SIP CGI, allow users or administrators to specify how a SIP proxy and redirect server should process calls. This document defines how SIP REGISTER requests and responses can be used to transport scripts between user agents and SIP proxy and redirect servers.

Contents

1 Introduction **2**

2 Conventions Of This Document **2**

3 Header Field Definitions **3**
3.1 Content-Disposition 3
3.2 Accept-Disposition 3
3.3 If-Unmodified-Since 4

4 Transport Details **4**
4.1 Script Upload and Removal 4
4.2 Server Response 6

5 Persistence Model **7**

6 Examples **7**

7 Usage notes **9**

8 Security Considerations **9**

9 IANA Considerations **10**

10 Changes from earlier versions **10**
10.1 Changes from draft -00 10
10.2 Changes from IPTel draft -00 10

11 Authors' Addresses **11**

1 Introduction

Several newly developed languages and interfaces, such as the CPL [1] and SIP CGI [2] allow users or administrators to specify how Internet telephony servers should process calls. Scripts typically can be created on a client, but executed on an Internet telephony server.

There therefore needs to be a method of transporting these scripts from a client to a server, and of retrieving them from the server so the client can know the current status or modify the script. This method should integrate cleanly with the existing infrastructure of Internet telephony, without requiring significant additional protocol traffic or complexity in either a client or a server.

This document defines how the payload of SIP [3] REGISTER messages, and their responses, can be used to transport these scripts to SIP registration servers alongside the user's registration. Since clients typically will need to register anyway, and servers will need to have registrars to process the clients' registrations, this technique does not impose much additional overhead on servers and clients.

This technique is not appropriate for all environments — most obviously, it is not useful for H.323 [4] servers — and we do not anticipate that it will be the only such transport mechanism developed. Other protocols considered have included transporting scripts over LDAP [5], ACAP [6], or HTTP file upload [7], or transport mechanisms developed from scratch.

The advantages of this technique, over these other possible methods for transfering scripts to a registration server, are twofold. First of all, a SIP client already needs to know the registrar and addresses to use in order to register a Contact location. Re-using this registration infrastructure makes it trivial to know the location to which a script should be sent. Other methods would require some correlation mechanism, or additional configuration options — a client would need to be told what HTTP server (for example) to use, in addition to knowing its SIP registrar.

Additionally, using this mechanism small SIP end systems can send and retrieve scripts without needing to implement additional protocols. Small embedded end systems are common for SIP; whereas parallel protocols would impose significant additional complexity in these devices, the mechanism described in this document requires very little of these devices over and above the base SIP specification.

2 Conventions Of This Document

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 [8] and indicate requirement levels for compliant implementations of SIP register payload script uploading.

Some paragraphs are indented, like this; they give motivations of design choices, or questions for future discussion in the development of the specification. They are not normative to the specification of the protocol.

3 Header Field Definitions

Script uploading borrows a number of header fields from HTTP and related MIME protocols. This section defines these extended headers.

3.1 Content-Disposition

The Content-Disposition header field is defined in RFC 2183 [9]. It has also been added to SIP in the work-in-progress revised SIP specification “2543bis” [10]. In REGISTER requests or responses, this header field is used to describe the intended purpose of a message body. The grammar of this header field is as follows:

```
Content-Disposition         =  “Content-Disposition” “:”
                               disposition-type *( “;” disposition-param )
disposition-type            =  “script” | “sip-cgi” | token
disposition-param           =  action-param
                            |  modification-date-param
                            |  generic-param
action-param                =  “action” “=” action-value
action-value                =  “store” | “remove” | token
modification-date-param     =  “modification-date” “=” quoted-date-time
quoted-date-time            =  <"> SIP-date <">
```

The grammar symbols “token” and “generic-param” are defined in RFC 2543 [3].

The Content-Disposition header field serves to describe the purpose of the message body. The disposition type describes the purpose of the material contained in the body of the message. Currently, two disposition types are defined. The type “script” indicates a CPL script or a similar scripting environment whose use in a SIP server can be uniquely determined by its media type. The type “sip-cgi” refers to SIP CGI scripts, which can be any media type executable on the server platform. Additional types can be registered with IANA through the procedure defined in RFC 2183 [9].

> The Content-Type of the uploaded payload is not sufficient to describe the purpose of the payload to the server. A script with a given purpose could, conceivably, be of any of a large number of media types, particularly for SIP CGI.

The action parameter to the header field is used when uploading scripts, to specify what the server should do with the script uploaded. If a non-zero-length script is specified, the action “store” MUST be given. The action “remove” MUST only be used when accompanied by a zero-length body.

The modification-date parameter is used to indicate the time when the script was last modified. This is used for versioning, to prevent potential race conditions (see Sections 4 and 7).

If multipart MIME types [11] are used to indicate several distinct scripts (see Section 4.2), this header field MUST be included in the MIME part header, not in the general SIP header.

3.2 Accept-Disposition

The Accept-Disposition header field is used to indicate what content disposition types are acceptable to a client or server.

```
Accept-Disposition   =   “Accept-Disposition” “:”
                         #( (disposition-type | “*”) *( “;” generic-param ) )
```

The special meta-type “*” matches every content disposition type, and indicates that any content disposition is acceptable.

The action and modification-date parameters are not meaningful for Accept-Disposition.

> The Accept-Disposition header field is not currently defined by any other published document as far as we know, but it is a natural counterpart to Content-Disposition. (In general, most Content-* header fields have corresponding Accept-* fields.)

3.3 If-Unmodified-Since

The If-Unmodified-Since request header field is defined in section 14.28 of the HTTP/1.1 specification, RFC 2616 [12]. It is used to make a request conditional. If the requested resource has not been modified since the time specified in this field, the server SHOULD perform the requested operation as if the If-Unmodified-Since header field were not present.

If the requested resource has been modified since the specified time, the server MUST NOT perform the requested operation, and MUST return 412 Precondition Failed.

Specifically, for register bodies, this header field is used to indicate that the client does not want the provided content to be stored if the corresponding content has been modified on the server since the given time.

The syntax of this header field is as follows:

```
If-Unmodified-Since   =   “If-Unmodified-Since” “:” SIP-date
```

An example of this field is:

```
If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT
```

If the request normally (i.e., without the If-Unmodified-Since header) would result in anything other than a 2xx or 412 status, the If-Unmodified-Since header field SHOULD be ignored.

If the specified date is invalid, the header field is ignored.

4 Transport Details

This section describes the procedures by which scripts are uploaded to a server, and retrieved from it.

4.1 Script Upload and Removal

To upload a script, the registration client places the script in the body of the SIP REGISTER request. Bodies of SIP requests are described in [3]. The Content-Type header field is set to the media type of the submitted script. The MIME type `application/cpl+xml` designates CPL scripts. Clients SHOULD upload SIP CGI scripts as an appropriate media type for the language the script is written in (for example, `application/x-perl`), or `application/octet-stream` if no such media type exists or is known. Registrars MAY perform validation on the media types if they know certain types of scripts cannot be executed on their servers, but SHOULD be permissive about unknown or ambiguous media types for SIP CGI scripts.

Which types of SIP CGI scripts can be successfully run on a server depends on the server's environment, including which scripting languages are installed on it. It is possible that the user has more knowledge of this environment than the server.

Script uploads MUST also be accompanied by a Content-Disposition header field (see Section 3.1) describing the purpose of the message body. This header field MUST have an action parameter indicating whether the script is to be stored or removed, and MAY have a modification-date parameter giving a timestamp for the script.

A script registrar's normal behavior is to enter the script in its database, as specified in section 5, associated with the user in the To field of the REGISTER message. However, if a zero-length script is submitted with the action remove, any existing script of the user's with the given disposition type is instead deleted from the database.

Note that having a zero-length script, and not having any script, are quite distinct conditions, and both are legal.

A script registrar MAY choose to add contents with an unknown disposition type and an action=store parameter to its script database. (In this case it SHOULD include a "*" field in Accept-Disposition headers that it sends — see Section 4.2.) It MAY alternately reject a script with an unknown disposition type with a 4xx response.

We anticipate that the mechanism described in this specification can also be used for purposes such as users' speed-dial lists or device configuration files, and that new disposition types would be registered for these.

To delete a script, a client sends a REGISTER message with its Content-Disposition header field with an the appropriate type and a action parameter of "remove", and a Content-Length header field of 0. If there is no script defined with the specified purpose, this message does nothing. When a script is deleted, the server SHOULD return to its default behavior, just as if no script had ever been uploaded.

A client MAY include an If-Unmodified-Since header field (see Section 3.3) to indicate that the uploaded script should only be accepted if the script on the server has not been modified since the given date. This is typically used if a client has downloaded a script (see Section 4.2), modified it, and wishes to upload the modified script; the If-Unmodified-Since header field guarantees that the script has not in the meantime been modified by any other client.

The server MAY perform syntactic and semantic validation on scripts at the time they are uploaded to the server. If the script is not valid, the server SHOULD return a 400-class error to the registration request indicating the problem. It MAY include in the body of the response an explanation of why the script was considered invalid, if the registration included an Accept header field with an appropriate media type for such an explanation (such as `text/html` or `text/plain`).

When a script with the same disposition type as an existing script is successfully uploaded for a given user, the previous script is replaced in the server. Scripts with different disposition types are stored and deleted independently.

How scripts interact with calls on the server is not defined by this document. In particular, which script applies to calls in progress at the time a script is added, changed or deleted is not defined by this document, but MAY be defined by specifications of script languages. However, if a current or new script affects the handling of REGISTER requests, the upload process SHOULD be handled entirely by the existing script; the new script does not take effect until the upload process has completed.

The entity which executes the user's script — i.e., a proxy or redirect server — needs to have access to the uploaded scripts. This document does not specify how this is done; typically, the registrar and proxy server are co-located. There normally will be a way for a registrar to pass information to an appropriate proxy server; normal SIP information such as registered Contact locations needs to be passed in order for a registrar to be useful.

Though the storage of scripts with different disposition types is independent, a server MAY choose not to execute some scripts if scripts with another disposition are present, for instance only executing one of a CPL and SIP CGI script.

If a script upload fails for any reason (including a validation failure, or an unsatisfied If-Unmodified-Since header), the script server MUST NOT perform any other actions associated with a successful REGISTER request, such as entering Contact headers in the registration database.

A client MAY also include in the upload request an Accept-Disposition header field listing disposition types it wants to receive in its response. (See Section 4.2.)

4.2 Server Response

In a successful response to any REGISTER request, whether or not a script payload was included, the server SHOULD return the currently stored scripts in the body of the response, unless the client requested otherwise.

If the request contained an Accept header, the server SHOULD NOT return any scripts whose media types do not match that header. Similarly, if the request contained an Accept-Disposition header, the server SHOULD NOT return scripts whose disposition types do not match that header.

Empty headers are legal for both the Accept and Accept-Disposition headers. (Grammatically, they are “#”, not “1#”.) If a client does not want to receive any scripts in response to a registration, it SHOULD include an empty Accept-Disposition header field in its REGISTER request. Servers SHOULD correctly honor empty Accept headers as well, but these are less likely to be useful for clients.

If multiple scripts are registered and match the Accept and Accept-Disposition headers, the server SHOULD return all of them in a multipart content if and only if the client included an appropriate `multipart/*` media type in its Accept header. Otherwise, the server MAY select any of the matching registered scripts to return. A client which cannot accept a multipart media type SHOULD NOT include multiple Accept-Disposition headers in its request.

Each returned script MUST have its media type specified by a Content-Type header, and its disposition type by a Content-Disposition header. The Content-Disposition header field SHOULD include the modification-date parameter indicating the time the script was modified. This header field SHOULD NOT include an action parameter, as the server is not requesting that the client perform any actions.

The server SHOULD NOT return any registered scripts if the response to the registration request was an error condition.

To inform a client of what types of scripts it supports, a server SHOULD include Accept and Accept-Disposition headers in any response to a registration, response to an OPTIONS request directed at the registrar request, and any response which rejected a registration on the grounds of an unsupported disposition or media type. A server which accepts arbitrary disposition types SHOULD include the wildcard disposition pattern “*” in its Accept-Disposition header.

Note: Including Accept headers in arbitrary REGISTER responses is against the strict wording of RFC 2543 [3], which says that Accept headers are only allowed in requests or 415 (Unsupported Media Type) responses. However, it is always legal to include a header field in any request or response, as clients which do not understand it in a given context simply ignore it. The work-in-progress revised SIP specification [10] allows this usage.

5 Persistence Model

Registrations in SIP are normally transient — the data in the Contact header fields last only for the length of time specified in the registration's Expires header, and clients must refresh their registrations periodically.

In contrast, scripts sent to registration servers using the method described in this document are persistent — they remain in the server until replaced or deleted, and they do not need to be refreshed. Servers SHOULD therefore store uploaded scripts in non-volatile storage so they persist through server restarts or failures. Clients SHOULD only upload scripts when they are explicitly requested to, and SHOULD NOT transmit their scripts in every registration request.

The model of standard SIP registrations is that each client registers itself; if a location changes or hosts die, old registrations naturally time out. Since a user can be simultaneously registered from many locations, several clients re-registering periodically present no conflicts.

The model of scripts is quite different. A user only has one script (or at least only of a given type) at a time, so if clients periodically re-uploaded scripts, two clients with different specified scripts would cause "script flapping," as the behavior specified in the server changed frequently, with unpredictable and probably surprising behavior. Moreover, one of the most important purposes of scripts is to control the processing of a user's requests when he or she is *not* registered from any location; if scripts timed out and had to be refreshed, this goal could not be accomplished.

6 Examples

The first example shows a user uploading a simple call-filtering SIP CGI script written in Perl to his server. Note that he is transmitting both a contact address, which persists only for 30 minutes, the time specified by the Expires header, and a script, which persists indefinitely. This allows him subsequently to register new contact addresses and have his script apply equally to them. (See [2] for an explanation of SIP CGI as used in the script.)

The use of Basic authorization here is for the purposes of the example only; in actual practice much more robust authentication SHOULD be used. See section 8.

```
REGISTER sip:sip.example.com SIP/2.0
From: Joe User <sip:joe@example.com>
To: Joe User <sip:joe@example.com>
CSeq: 18 REGISTER
Expires: 1800
Call-ID: 39485832@joespc.example.com
Contact: sip:joe@joespc.example.com
Accept: application/x-perl, application/sdp, text/html
Accept-Disposition: sip-cgi
Authorization: Basic am9lOnBhc3N3b3JkAFBX
Content-Type: application/x-perl
Content-Length: 137
Content-Disposition: sip-cgi; action=store

#!/usr/bin/perl
if ($ENV{HTTP_FROM} =~ /telemarketers.com/) {
    print "SIP/2.0 603 Go away\n"
} else {
    exit(0); # Default action
}
```

In the second example, a few minutes later, the user registers a new contact address, but does not change his script. In the response to the registration, the server reminds him of his contact addresses and his current script.

His client sends this request:

```
REGISTER sip:sip.example.com SIP/2.0
From: Joe User <sip:joe@example.com>
To: Joe User <sip:joe@example.com>
CSeq: 19 REGISTER
Expires: 1800
Call-ID: 39485832@joespc.example.com
Contact: sip:joe@joeshome.example.com
Accept: application/x-perl, application/sdp, text/html
Accept-Disposition: sip-cgi
Authorization: Basic am9lOnBhc3N3b3JkAFBX
Content-Length: 0
```

And the server replies with this response:

```
SIP/2.0 200 OK
From: Joe User <sip:joe@example.com>
To: Joe User <sip:joe@example.com>
CSeq: 19 REGISTER
Contact: sip:joe@joespc.example.com
Contact: sip:joe@joeshome.example.com
Accept: application/cpl+xml, */*
Accept-Disposition: script, sip-cgi
Content-Type: application/x-perl
Content-Disposition: sip-cgi;
    modification-date="Wed, 25 Oct 2000 21:21:54 GMT"
Content-Length: 137

#!/usr/bin/perl
if ($ENV{HTTP_FROM} =˜ /telemarketers.com/) {
    print "SIP/2.0 603 Go away\n"
} else {
    exit(0); # Default action
}
```

Finally, the user decides to eliminate his script, and the server responds in the same manner as it would respond to an ordinary registration, as though no script had ever been uploaded:

```
REGISTER sip:sip.example.com SIP/2.0
From: Joe User <sip:joe@example.com>
To: Joe User <sip:joe@example.com>
CSeq: 20 REGISTER
Call-ID: 39485832@joespc.example.com
Contact: sip:joe@joeshome.example.com
Authorization: Basic am9lOnBhc3N3b3JkAFBX
Accept: application/x-perl, application/sdp, text/html
Content-Length: 0
Content-Disposition: sip-cgi; action=remove

SIP/2.0 200 OK
From: Joe User <sip:joe@example.com>
To: Joe User <sip:joe@example.com>
CSeq: 20 REGISTER
Contact: sip:joe@joespc.example.com
Contact: sip:joe@joeshome.example.com
Accept: application/cpl+xml, */*
Accept-Disposition: script, sip-cgi
Content-Length: 0
```

7 Usage Notes

Because scripts can be long, clients which upload scripts, or which present an Accept header field which could cause scripts to be returned, SHOULD send their REGISTER messages over TCP rather than UDP.

A user agent which downloads a script to allow a user to edit it, and then re-uploads the script once the editing is complete, SHOULD include a If-Unmodified-Since header field in the re-uploading process with a value equal to the downloaded script's modification-date. This guarantees that the script has not been modified by any other user agent since it was downloaded.

8 Security Considerations

Scripts transported by this mechanism can control how a server processes private information intended for a user. Therefore, a server MUST reject all un-authenticated attempts to submit, alter, or delete a script. It is very strongly RECOMMENDED that that the server require an authentication method which verifies the integrity of the submitted script, to prevent an attacker from replaying a script submission with a different script body. Examples of such authentication methods are Digest authentication [13] with the quality of protection "auth-int", and SIP's PGP authentication. Alternately, transport or network-layer authentication and integrity verification (TLS [14] or IPSec [15]) can be used between the client and server.

It is also RECOMMENDED that a server authenticate and provide integrity verification of the scripts it returns. PGP, transport-layer and network-layer authentication accomplish this as well.

It may be possible to use Digest authentication for server-to-client authentication, but it is not clear how nonce handling would work.

9 IANA Considerations

The process for registering new Content-Disposition values and parameters is given in RFC 2183 [9].

This document defines two new Content-Disposition values, "script" and "sip-cgi", and one new parameter, "action", which can have the value "store" or "remove".

10 Changes From Earlier Versions

10.1 Changes From Draft -00

The changebars in the Postscript and PDF versions of this document indicate significant changes from this version.

- Improved wording in abstract and introduction: this is a specification, not a proposal. It also applies only to SIP.
- Added wording to the introduction motivating the use of this specification rather than the other possitiblities.
- Changed to using the Content-Disposition header, to be in line with rfc2543bis and HTTP. Eliminated Content-Purpose and Content-Action in favor of the new header.
- Added a paragraph motivating the separation of Content-Disposition types from media types.
- Added Accept-Disposition header.
- Added If-Unmodified-Since header.
- Separated description of header syntax from upload and download procedures.
- Clarified that scripts are per-user, and associated with the user in the To header.
- Clarified that the model is that proxy servers have access to the registered scripts.
- Added usage note that user agents which support download-edit-upload functionality should use the If-Unmodified-Since header to prevent race conditions.
- Expanded upon the security considerations. Added mention of qop=auth-int.

10.2 Changes From IPTel Draft -00

This document was originally published as `draft-iptel-sip-reg-payload-00`, but the consensus of the IPTel working group was that this should not be a work item of that group.

- Added Content-Purpose and Content-Action headers.
- Changed the procedure by which scripts are deleted.
- Eliminated the pseudo-media-type `application/sip-cgi`, as it is counter to the spirit of MIME. Instead, established that SIP CGI scripts can be any media type.

- Added “Conventions,” “Usage Notes,” and “IANA Considerations” sections.
- Updated examples to use the syntax of the current version of SIP CGI.
- Updated references to refer to the latest versions of all documents.

11 Authors’ Addresses

Jonathan Lennox
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
USA
electronic mail: lennox@cs.columbia.edu

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
USA
electronic mail: schulzrinne@cs.columbia.edu

References

[1] J. Lennox and H. Schulzrinne, “CPL: a language for user control of internet telephony services,” Internet Draft, Internet Engineering Task Force, Mar. 1999. Work in progress.

[2] J. Lennox, J. Rosenberg, and H. Schulzrinne, “Common gateway interface for SIP,” Internet Draft, Internet Engineering Task Force, June 2000. Work in progress.

[3] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, “SIP: session initiation protocol,” Request for Comments 2543, Internet Engineering Task Force, Mar. 1999.

[4] International Telecommunication Union, “Packet based multimedia communication systems,” Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[5] M. Wahl, T. Howes, and S. Kille, “Lightweight directory access protocol (v3),” Request for Comments 2251, Internet Engineering Task Force, Dec. 1997.

[6] C. Newman and J. G. Myers, “ACAP – application configuration access protocol,” Request for Comments 2244, Internet Engineering Task Force, Nov. 1997.

[7] E. Nebel and L. Masinter, “Form-based file upload in HTML,” Request for Comments 1867, Internet Engineering Task Force, Nov. 1995.

[8] S. Bradner, “Key words for use in RFCs to indicate requirement levels,” Request for Comments 2119, Internet Engineering Task Force, Mar. 1997.

[9] R. Troost, S. Dorner, and K. Moore, “Communicating presentation information in internet messages: The content-disposition header field,” Request for Comments 2183, Internet Engineering Task Force, Aug. 1997.

[10] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, “SIP: Session initiation protocol,” Internet Draft, Internet Engineering Task Force, Aug. 2000. Work in progress.

[11] N. Freed and N. Borenstein, “Multipurpose internet mail extensions (MIME) part two: Media types,” Request for Comments 2046, Internet Engineering Task Force, Nov. 1996.

[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext transfer protocol – HTTP/1.1,” Request for Comments 2616, Internet Engineering Task Force, June 1999.

[13] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, “HTTP authentication: Basic and digest access authentication,” Request for Comments 2617, Internet Engineering Task Force, June 1999.

[14] T. Dierks and C. Allen, “The TLS protocol version 1.0,” Request for Comments 2246, Internet Engineering Task Force, Jan. 1999.

[15] S. Kent and R. Atkinson, “Security architecture for the internet protocol,” Request for Comments 2401, Internet Engineering Task Force, Nov. 1998.

Full Copyright Statement