Requirements for SIP Servers and User Agents*

Henning Schulzrinne
Columbia University

November 16, 1998

1 Introduction

This document describes requirements for first-generation Internet SIP-based telephony servers and user agents.

Servers can operate in either redirect or proxy mode. In redirect mode, a server receives a request and then informs the client to contact some other server, the "next hop". The redirect server does not maintain any call state. The proxy server also acts as a client, issuing one or more outgoing requests triggered by each incoming request. It maintains state while it is contacting other servers or user agents. Note that a server or user agent does not need to know whether it is communicating with another server or a user agent.

Servers SHOULD be able to operate in either mode, preferably configurable on per-user user or domain basis.

We distinguish two kinds of user agents:

Telephony: The telephony client supports audio only, but has sophisticated call control capabilities, including

- call forwarding,
- blind and supervised call transfer,
- consultation,
- adding users to an existing conversation
- full-mesh, multicast and MCU-based multiparty conferences

Conferencing: The conferencing client supports multiple media and multi-user conferences.

The server is the same for both applications. Servers can forward and reject calls.

In the long run, it may well be desirable to merge the two kinds of user agents. The requirements for the two do not contradict each other. The multimedia conferencing application more heavily focuses on the support of multiple media, while the telephony application emphasizes sophisticated call control. [Adding multiple media to the telephony client doesn't seem to increase its signaling complexity any.]

*MCI has contributed to the support of this work.

1.1 Terminology

This specification uses the same words as RFC 1123 [1] to define the significance of each particular requirement. These are:

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

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

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

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

2 Basic Requirements

2.1 SIP Protocol Support

Clients and server MUST implement SIP as described in Internet draft `draft-ietf-mmusic-sip-05.txt` [2] and call control services as described in `draft-ietf-mmusic-sip-cc-00.txt` [3]. A technical report [4] provides additional motivation and descriptions of more advanced services. Nothing in this requirements document should contradict the specification, but this document describes SIP servers and user agents focused on implementation, not protocol elements.

While the SIP protocol does not prescribe any maximum line length for requests, implementations should fold lines longer than 72 characters for ease of debugging. Implementations should operate correctly with any line length.

Support of the UTF-8 character set is mandatory; it is acceptable [TBD] to only display the 8859-1 (Latin) subset to the user and only allow 8859-1 user input. User agents and logging subsystems should be careful to filter or escape characters according to local conventions.

2.2 SIP Requests

Implementations shall support INVITE, ACK, BYE, OPTIONS and REGISTER.

2.3 SIP Responses

Any non-successful ($\geq$ 300) SIP response may contain either a text/plain or text/html body for display to the client. Succesful responses always contain a session description. Informational responses contain no message body.

method, status	Location	Also	Replaces
INVITE	–	√	√
ACK	–	–	–
BYE	blind transfer	√	√
OPTIONS	–	√	√
REGISTER	user location	–	–
1xx	forwarding location	–	–
2xx	contact directly	√	√
3xx	redirect location	–	–
4xx	–	–	–
5xx	–	–	–
6xx	–	–	–

2.4 Session Description

SIP implementations MUST support the SDP [5] session description format.

3 User Agent Requirements

3.1 Network

A user agent MUST support unicast UDP and SHOULD support multicast UDP and TCP. It first tries to bind to the SIP port (5060) on startup; if that fails because some other SIP user agent (or errant application) is already using that port, it binds to a suitable random port.

Unless configured with a fixed SIP server, the user agent sends a REGISTER message with ttl equal to 1 to the "all SIP servers" reserved multicast address (224.0.1.75). It includes Location headers for both UDP and TCP. A user agent should be prepared to receive several responses from different local servers. It may then store these addresses and use one as the primary server, the others as fallback servers.

By default, user agents SHOULD try to locate a local SIP server first. If a server cannot be found, they MAY try to contact callees directly. (Some implementations may disable direct calls for greater administrative control.) User agents MAY be configurable to contact certain domains directly. (This approach is similar to the one used for configuring proxies for web browsers, except that SIP allows to discover the proxy automatically.)

If the local server were to announce the domains it wants and does not want to handle, the last step could be automated.

3.2 User Interface and Architecture Considerations

User agents may either integrate call processing and media handling into one application or split the two functionalities. In the latter case, a number of arrangements are possible:

1. Each host has a single priviledged SIP "daemon" that runs permanently in the background. This SIP daemon receives incoming calls and starts up a per-user, per-call SIP daemon, proxying or redirecting calls as needed.

2. Each user runs a single SIP-capable application in the background. When a call comes in, it "pops up" a call handling window through which the callee determines what to do with a call: accept, reject or forward. If the callee accepts, he can choose the subset of media from the ones indicated in the

caller's session description. Appropriate media agents are then either started up or, if already running and capable of handling the additional session, configured for the network parameters of the caller. Media agents are then responsible for making any resource reservations. This solution is simpler and avoids having to run a priviledged daemon, but forces users to leave a SIP process running at all times.

If the user agent is not itself capable of rendering HTML, it SHOULD be able to send requests to a local web browser to view the URL. This external browser should then also be used to display `text/html` SIP message bodies.

It may simplify the design of media agents if the SIP user agent also displays the participants for a call. A design choice to be considered is how to handle several concurrent calls, e.g., as separate windows or as separate applications.

It is recommended that user agents can invoke other tools for non-SIP URLs such as "rtsp" and "mailto".

3.3 Incoming Calls

For incoming calls, users SHOULD be shown the content of the To, From, Organization, Priority and Subject fields. User agents are free to map this information to a more user-friendly rendition, e.g., by mapping different priority levels to different ringing sounds or From header values to "real" names through a local database lookup. User agents SHOULD indicate whether and how an incoming call has been authenticated.

User agents MUST check for mistaken To and request-URI and reject calls not intended for it with status code 404 (Not Found).

User agents SHOULD automatically accept invitations for an existing Call-ID and should allow to change allowable media types and/or encodings during a call.

A user agents receiving a call proceeds as follows:

1. Invoke any per-user script (Section 7.1).

   > Should we have different scripts for user agent and server? (Server script may do its own user location, which would not make sense at the user agent.)

2. The script may either refuse, automatically accept or defer decision on the call. If a decision is deferred, the user agent SHOULD return a "1xx" response. (A script defers by returning a status of 1xx.)

3. If the script does not decide, the user is asked via a user interface element how to handle the call. Note that it is possible that a second call arrives while the first one is still "ringing". It should be up to the user to decide which of the pending calls to accept.

4. The user agent SHOULD be configurable with a timeout interval. If the user does not answer the call within a preset time interval, the user agent, if thus configured, logs the call to a file and/or sends an email message to the owner. It then returns status 408 (Request Timeout). A timeout interval avoids that the user finds a slew of abandoned call attempts on her screen when returning from a prolonged absence.

3.4 Outgoing Calls

For each outgoing call, the caller MUST be able to choose whether to initiate a call that uses multicast or unicast for media distribution, regardless of the number of (immediate) callees. The user SHOULD also be able to indicate whether it wants to reach the first callee, in case the caller address (To) expands to a list, or reach all of them. (Call-Disposition: all). The user SHOULD be able to choose whether the call is allowed to be forwarded (Call-Disposition: do-not-forward).

The user SHOULD (at least optionally) be advised of call progress (1xx responses).

An INVITE may return a list of Location values containing SIP and non-SIP (“mailto”, “telephone”, . . .) URLs. The following behavior is suggested:

1. Remove any locations that the user has defined as being undesirable. For example, a user might configure the user agent to ignore RTSP and HTTP URLs or to ignore URLs marked as “business”.
2. Remove any URLs that require a priority higher than the current call priority.
3. If the user has indicated she wants to confirm redirected calls, she should be presented with an ordered list of URLs and be allowed to remove any of the locations from further consideration.
4. Then, SIP URLs are tried in order or, if they have the same precedence, in parallel.
5. For non-SIP URLs, the user is presented with a pop-up stating that the callee was not reachable via an Internet call, but left alternate instructions. It is probably easiest to present all such options together and let the user choose which ones to exercise.

Lists are ordered by “q” value if given, by order of appearance if not.

User agents SHOULD support basic and digest authentication for both end systems and proxies. User agents SHOULD [when this is fully specified] support PGP-based authentication of request and MAY support encryption of requests. [TBD; Currently, the details of how to do such authentication are not fully specified.]

It is recommended that the user start up the audio media agent when sending the INVITE request rather than waiting for an ACK. This is necessary so that the user can receive any in-band audio announcements (“The number you have dialed . . . ”, “All circuits are busy”) that may be generated by an Internet-to-GSTN gateway. (Due to proxying, the caller would in general not know whether a call is being connected, e.g., through a PBX.)

3.5 Call Services

User agents SHOULD be able to act on the Also and Replaces headers as described in the SIP specification.

User agents should be able to initiate the following services:

Call forwarding unconditional: Based on the script interface (Section 7.1), user agents SHOULD be able to forward all incoming calls to another address by returning a 301 (Moved permanently) or 302 (Moved temporarily).

Call forwarding conditional: Same as before, except that forwarding is based on local conditions.

Call forwarding busy: A user should be able to configure the user agent to automatically reject or forward calls if the user is already in another call. It is recommended that the user can configure this for an on-going call, allowin him to disable “call waiting” for important calls.

If the user accepts several concurrent calls, it should be possible to force the media agents to generate media for one or more of the current calls, subject to resource availability. For users with low-bandwidth connectivity, the media agents should be configured to only generate a single stream.

Distinctive ringing: A user agent may have a configurable mechanism to map the From field to different alerting signals. Alternatively, the user agent may be tied to a text-to-speech system that announces the caller’s name and other call information.

Do not disturb: It is recommended that user agents feature a control that allows to temporarily reject or forward calls below a certain priority level. Incoming calls should still be signaled visually, with the indication that do-not-disturb mode is in effect.

Speed dialing: It is recommended that the user interface contains one-click mechanisms to reach commonly called numbers and maintains a list of recently callers and callees.

Multi-party calls: If the current call is a unicast call, any party in a call SHOULD be able to add additional parties by sending the new party in the call an INVITE request for the current Call-ID containing the list of current members in the Also header. Note that such meshes have poor scaling properties and SHOULD be limited to calls of no more than four parties. For larger groups, the call should transition to a multicast or MCU-based call.

If the call is a multicast call, it suffices to send an invitation with that multicast address to the party to be added to the call.

See [4] for additional details on MCU-based calls and the transition from unicast to multicast calls.

Unsupervised call transfer: A user should be able to transfer an existing call to any address. Unsupervised call transfer is implemented by sending a BYE request to the calling party that contains a Location header pointing to the transfer destination.

Supervised call transfer: See [4].

3.6 Media Encodings

Audio and video media agents SHOULD implement the revised RTP specification [6] and current profile. Audio agents SHOULD at least support the following encodings:

encoding	channels	sampling rate (Hz)
G.711 (μ-law)	1	8,000
DVI4	1	8,000
GSM	1	8,000
G.723	1	8,000
Voxware	1	8,000
Digitalk	1	8,000

Video agents SHOULD at least support the following encoding:

encoding	image size
H.261 with conditional replenishment	QCIF, CIF

The media agents MUST be able to map any RTP PT to any encoding, that is, support dynamic payload types.

User agents SHOULD allow the user to specify a prefered ordering of payload types, with payload types of higher preference listed first in the SDP "m" parameter.

4 Server Requirements

A server should be able to operate in either redirect or proxy mode. It may be appropriate to proxy some users (e.g., those local to the server), while redirecting other users (e.g., those that are only guests or whose location is outside the local domain).

4.1 Call Processing

Servers process incoming calls (INVITE requests) in several stages:

1. If the request is INVITE or OPTIONS, check if the name represents a local alias for either a single user or a mailing list; if so, skip the next step. If the name is a list, the call handling depends on the value of the Call-Disposition header: If the header contains the parameter all, the server returns the list to the requestor as Also headers. If all is not present, the server SHOULD either proxy or redirect the call, depending on the configuration (Section 5), with redirection being the default action. Proxying is described in the chapter "Behavior of SIP Servers" in the SIP specification [2]. For redirection, the server returns a list of Location headers to the caller.
2. Map the name in Request-URI to a local user id as described in (Section 4.2).
3. If the request is for an existing Call-ID, handle it as designated for the call and skip the next step.
4. Check for global handling directives which are specific to the caller or callee (Section 5).
5. If the call has not been handled by the previous step or the user location has been left for the server to determine, find the location of the user by invoking a default user location script[1]. Section 6 describes some approaches to user location. If there is no user location script, a server MAY also multicast the INVITE request to a multicast address which is either specific to a particular user or shared by several users.
6. If the script or general server configuration called for proxying, issue the necessary commands.
7. If the user does not respond, the server MAY generate a MIME email message with the call request.

4.2 Name mapping

Servers SHOULD support the mapping of user names to a canonical name. For example, the subscriber John Q. Public with id "jqp" may be matched by the following user names in the Request-URI:

Name in request-URI	type of match
jqp	U
John	F
Public	L
J.Q.Public	FL
John.Q.Public	FL
John_Public	FL
JohnPublic	FL
JPublic	FL
JohnQPublic	FL

The type of match is indicating by the letters U, F, L, or FL, indicating a match on user, first, last name or first and last name. Matches of names and domains are case-insensitive and should be performed after expanding URL-escaped characters.

Names may, for example, be mapped based on the user registry, e.g., the Unix `/etc/passwd` file.

Servers should allow the administrator to configure

[1] The per-user script is invoked first since it makes no sense to find a user that does not want to accept calls.

- whether names are mapped at all and by what script;
- whether multiple responses are
  - returned in Location headers via a 300 (Multiple Choices) response;
  - refused via 404 (Not Found).

The interface to the name mapper service will be system-dependent. The service could be integrated into the server, called via a remote procedure call or invoked as an external process. For an external process, it is suggested that the namemapper process be called with the name part of the Request-URI and return its result via stdout.

5 Global Call Handling Directives

It should be possible to configure the server to perform different actions based on the To and From headers in INVITE, OPTIONS and REGISTER. Note that there is no need for separate directives for ACK and BYE as they are treated as the INVITE, OPTIONS or BYE request for that Call-ID. If there is no Call-ID for a given ACK or BYE, it is treated like an INVITE request. (Note that BYEs are only used after INVITE or REGISTER.)

There are many different approaches to configuring call handling. We describe, as an example only, an approach that is simpler to implement than a full-fledged language, yet probably sufficient for most cases. (Note: As an alternative, it is probably pretty trivial to simply assign headers to Tcl variables, create actions and write a small Tcl script. No reason to re-invent language elements and expression parsing.)

In our example, each line of the configuration file contains one rule with four elements, separated by white space:

1. The name of the header field to be checked.
2. The name of the method (INVITE, OPTIONS, REGISTER) this rule applies to.
3. The domain, as a "glob" string, this rule applies to. (Glob strings are simpler and faster than regular expressions; they contain "*" or "?" to designate a wildcard string or character.) Strings are matched from the end, so that "columbia.edu" matches "erlang.cs.columbia.edu", but "@columbia.edu" only matches "foo@columbia.edu".
4. The action to be taken:
   - A number that indicates a status code (e.g., 300). A 3xx status code is followed by a list of space-separated URLs to be returned in the Location header. If that list is empty, the server tries to locate the user according to Section 6. No further rules are processed.
   - The action "proxy" followed by a list of domain names asks the server to send proxy requests to the domains listed. If not explicitly specified, the server tries to locate the user (Section 6). No further rules are processed.
   - The action "authenticate" requires that the call has to be authenticated with a strength equal to the argument. Authentication options are, in order of increasing strength, "basic", "digest" and "pgp". If the call is already authenticated with sufficient strength, this rule is ignored. For basic and digest authentication, the caller has to appear in the AuthUserFile and provided the correct password.

- The action "script" invokes a cgi script. The name of the script follows. If no name is given, the user-specific script is used. If the script returns a final response, processing of the rule file is terminated.
- The action "log" logs the call. The first argument gives the log file format directive (Section 6.5), the second the file name or command (if preceded by a '|') for the log file. If no file name is given, actions are logged to the default file given via the RequestLog directive (Section 6.4). Processing of the rule file continues.
- If the server supports "voice mail" (i.e., multimedia recording), the action "record" initiates an RTSP or /request to the server listed in the argument.

For example, a SIP server for the domain cs.columbia.edu may have the following table:

```
# We log all calls, but not registrations.
From * INVITE log |/usr/local/bin/logger

# We allow registration only from local users
From REGISTER cs.columbia.edu 200
From REGISTER * 404

# We forward calls to some alumni.
To   INVITE  graduated@cs.columbia.edu 301 sip://vp@example.com

# Calls to *-voicemail get voicemail (probably by indirection).
To   *-voicemail record voice-mail.cs.columbia.edu

# All other calls are subject to callee-written (user-specific) scripts.
To INVITE cs.columbia.edu /usr/local/bin/user_script

# We are a redirect server; the server figures out the location.
To   INVITE   cs.columbia.edu 302

# We require that AOL users are authenticated.
From INVITE   aol.com authenticate

# We don't handle any other domains.
To   INVITE   * 404
```

For a Tcl-ish solution, a subset of the above might look like:

```
if {[glob $from *] && $method == "INVITE"} {
  log
}
if {$method == "INVITE" && $to == "graduated@cs.columbia.edu"} {
  301 vp@example.com
}
```

6 User Location

The server locates users by one or more of the following algorithms, in rough order of precedence and desirability:

- Internal database built from REGISTER requests.
- Sending out a multicast INVITE request to the local network. The multicast should be sent either with a TTL of 1 (local network) or an administratively scoped address.
- Using a file-system based registration mechanism, where the user agent writes its location at a specified location in a shared file system.
- Contacting an external database.
- Consulting the user's `.forward` or similar mail forwarding file or registry information.
- Recursive "finger".

A server MAY rely on REGISTER, but servers MAY first attempt to reach a user at the registered location and then, should that fail, attempt the other location methods.

6.1 Voice Mail

A server MAY also support RTSP-based voice mail services. There are several different ways to implement such a service [4]. A server-based approach that also works for calls connected through a GSTN gateway proceeds as follows:

1. The SIP server establishes an RTSP session with a designated RTSP-capable voice mail server and obtains the server's transport address.
2. The SIP server accepts the call and returns an SDP description containing the address obtained in the previous step.
3. The caller now sends media data to the RTSP server, which records it.
4. When the SIP server receives a BYE request, it sends a SHUTDOWN request to the RTSP voice mail server.
5. The SIP server then delivers the RTSP URL for the voice mail message to the called user via email.

6.2 Per-User Configuration and Scripts

6.3 Configuration

Each local user should be able to configure the processing of requests, via either simple fixed forwarding or the invocation of a script.

Note: This document describes only scripts that govern the response to INVITE and OPTIONS requests. These scripts cannot generate other requests or maintain per-call state. Such functionality is being planned for a call processing language to be developed within the IETF IPtel working group.

The server invokes the user script when receiving the first instance of an INVITE or OPTIONS request for a particular Call-ID and stores the value returned by the script for at least 5 minutes. Subsequent requests for the same Call-ID return the same response. The server also returns this response for BYE and ACK requests.

Per-user configuration may be kept in either the Windows registry (for Windows NT and Windows'95), a standard (SQL) database or a per-user file. Per-user files are only appropriate for servers which share a file system with user accounts, such as a departmental SIP server. They have the advantage that access is fast and users can make changes without requiring elaborate per-user access control in databases.

Per-user configuration files are kept in the `.siprc` file in the user's home directory for Unix operating systems and in an appropriate per-user file or database entry for other operating systems. This is the rough equivalent of the `.forward` file used by the sendmail Unix mail delivery daemon.

The per-user configuration may either include a list of addresses that are returned in 303 (Moved permanently) Location responses or the name of a script or program starting with the | character.

For example,

```
|/usr/local/sipd/vacation js
```

would invoke a SIP "vacation" program to return the appropriate response. A simple static forwarding rule would be written as

```
J.Public@new_address.com, J.Public@ieee.org
```

For security reasons, the server SHOULD check that the user's home directory and the `~/.siprc` file are not writable by anybody but the owner.

The per-user simple forwarding entry cannot indicate the status code in responses, but it is convenient for common forwarding tasks. The script interface described below differs, however, from the way that sendmail passes the whole message for delivery to "vacation" programs, so that a direct re-use of this mechanism is not possible.

6.4 Server Configuration

6.4.1 Global Configuration

Servers MAY be configured similar to the httpd and Apache web servers, that is, using a text file. The following configuration statements make sense for SIP servers. There should be one server-wide configuration file and, in addition, per-user files that can override certain behavior.

> In Windows environments, this information might be specified through the registry, although I find it easier to edit, document and distribute configuration files.

Configuration files may contain comments starting with a # sign in the first non-blank column.

> It has been suggested to use the RFC822 format instead. That way, the server could use the same parser as for messages.

AuthName: This directive sets the name of the authorization realm for a directory. This realm is given to the client so that the user knows which username and password to send.

AuthType: This directive selects the type of user authentication for the server. Only `Basic` is currently implemented.

AuthGroupFile: The AuthGroupFile directive sets the name of a textual file containing the list of user groups for user authentication.

AuthUserFile: The AuthUserFile directive sets the name of a textual file containing the list of users and passwords for user authentication. Filename is the path to the user file. Each line of the user file file contains a username followed by a colon, followed by the crypt() encrypted password. The behavior of multiple occurrences of the same user is undefined.

BindAddress: A server can either listen for connections to every IP address of the server machine, or just one IP address of the server machine.

CustomLog: The first argument is the filename to which log records should be written. This is used exactly like the argument to TransferLog; that is, it is either a full path or relative to the current server root. The format argument specifies a format for each line of the log file. The options available for the format are exactly the same as for the argument of the LogFormat directive. If the format includes any spaces (which it will do in almost all cases) it should be enclosed in double quotes.

ErrorDocument: In the event of a problem or error (status 4xx or 5xx), a SIP server can be configured to do one of two things:

1. output a simple hardcoded error message that simply echoes the status response line,
2. output a customized message stored in a local file (with Content-Type type text/plain or text/html[2]),

The first option is the default, while option 2 is configured using the ErrorDocument directive, which is followed by the SIP response code and a message or URL.

ErrorLog: The error log directive sets the name of the file to which the server will log any errors it encounters. If the filename does not begin with a slash (/) then it is assumed to be relative to the ServerRoot.

Group: The Group directive sets the user group under which the server will answer requests. In order to use this directive, the stand-alone server must be run initially as root. The user group determines which files and applications the server has access to. Typically, it is set to 'noone'.

HostNameLookups: This directive enables DNS lookups so that host names can be logged (and passed to CGIs in REMOTE_HOST). The value double refers to doing double-reverse DNS. That is, after a reverse lookup is performed, a forward lookup is then performed on that result. At least one of the ip addresses in the forward lookup must match the original address.

KeepAlive: Set to "On" to enable persistent connections, "Off" to disable.

KeepAliveTimeout; The number of seconds the server will wait for a subsequent request before closing the connection. Once a request has been received, the timeout value specified by the Timeout directive applies.

Listen: The Listen directive instructs the server to listen to more than one IP address, protocol or port; by default it responds to TCP requests on all IP interfaces, but only on the port given by the Port directive.

ListExpansion: If set, it determines the script to be run to map a user name to one or more addresses. The server passes the user name part of the request-URI to a script as a command line argument and expects to receive a list, one per line, of list members or aliases.

ListProxy: If set to "on", the server (if not overridden by domain-specific directives) will send proxy invitations if the list expansion (see ListExpansion) returned a list of names and if the call does **not** have a Call-Disposition: all.

[2]Servers MAY use the `mime.types` file to determine the file type of local files.

LogFormat: The format is described in Section 6.5.

LogLevel: LogLevel adjusts the verbosity of the messages recorded in the error logs. The following levels are available, in order of decreasing significance:

PidFile: The PidFile directive sets the file to which the server records the process id of the daemon. If the filename does not begin with a slash (/) then it is assumed to be relative to the ServerRoot. The PidFile is only used in standalone mode.

RequestLog: The RequestLog directive adds a log file in the format defined by the most recent LogFormat directive, or Common Log Format if no other default format has been specified. File-pipe is either a filename relative to the ServerRoot or '|' followed by a command, the program to receive the agent log information on its standard input.

ServerAdmin: The ServerAdmin sets the e-mail address that the server includes in any error messages it returns to the client.

TimeOut: The TimeOut directive currently defines the amount of time (in seconds) that the server will wait for requests on a TCP connection before closing it.

User: The User directive sets the operating system userid as which the server will answer requests. In order to use this directive, the standalone server must be run initially as root. This directive determines which files and applications the server has access to. It is typically set to 'noone'.

We want to allow per-domain handling, with wildcards.

Examples and more detailed definitions can be found in the Apache web server definition.

6.5 Request and Response Logging

It is suggested that servers allow the logging of calls in a format similar to the httpd server log format. This allows the immediate re-use of common log analysis tools.

The http log format is

host ident authuser date request status bytes

The log format for a SIP server is similar:

host From To *date request status*

where *date* is in "[day/month/year:hour:minute:second zone]" format. The *host* field names the host issuing the SIP request.

An example for a server at `example.org`, broken across two lines for readability:

```
erlang.cs.columbia.edu hgs@cs.columbia.edu john.q.public@example.org
  [11/Feb/1998:03:18:05 -0500] "INVITE sip://jqp@example.org SIP/2.0"
  301
```

Instead, a server MAY offer a configurable log file format, similar to the Apache configurable log file format:

%...h	Remote host
%...{Foobar}i	The contents of Foobar header line(s) in the request sent to the server.
%...{Foobar}o	The contents of Foobar header line(s) in the reply.
%...p	The port the request was served to.
%...P	The process ID of the child that serviced the request.
%...r	First line of request.
%...s	Status.
%...t	Time, in common log format time format.
%...{format}t	The time, in the form given by format, which should be in strftime(3) format.
%...T	The time taken to serve the request, in seconds and fraction.
%...u	Remote user (from authentication; may be bogus if return status (%s) is 401).
%...U	The URL path listed in the request line.
%...v	The name of the server (i.e., which virtual host)

The '...' can be nothing at all (e.g. "%h %u %r %s %b"), or it can indicate conditions for inclusion of the item (which will cause it to be replaced with '–' if the condition is not met). Note that there is no escaping performed on the strings from "%r", "%...i" and "%...o".

The forms of condition are a list of SIP status codes, which may or may not be preceded by "!". Thus, "%400,501{User-agent}i" logs User-agent on 400 errors and 501 errors (Bad Request, Not Implemented) only; "%!200,304,302{From}i" logs From on all requests which did not return some sort of normal status.

Note that the common log format is defined by the string

```
%h %u %t \"%r\" %s %b
```

which can be used as the basis for extending for format if desired (e.g., to add extra fields at the end). NCSA's extended/combined log format would be

```
%h %l %u %t \"%r\" %s %b \"%{Referer}i\" \"%{User-agent}i\"
```

100-class responses are not logged.

7 API for SIP servers

While easy to generate and parse from within an application, it is sometimes of advantage to leave SIP processing to a lower-layer library or daemon.

In many cases, SIP will be part of a generic telephony interface such as TAPI [7] or JTAPI [8]. However, these typically do not support the full range of SIP functionality (such as third-party call control and multicast signaling) and parameters (such as Priority and Subject or Location parameters).

We can distinguish a protocol-oriented from a service-oriented API. A protocol-oriented API hides the details of the SIP and SDP message formats, transport issues and the like. A transport API should support the following basic structure for outgoing and incoming calls:

- create a call;
- set generic or call-specific parameters such as Accept-Language or acceptable media encodings;
- set parameters for a call;
- send an invitation or OPTIONS for a call;
- accept or redirect an incoming call;

- terminate a call.

The following events may occur:

- incoming call, automatically creating a call identifier;
- progress of out-going call

For a C API, it appears easiest to have the application register a callback function that is invoked upon receiving events.

7.1 Script Interface

The script ("cgi-bin") interface for SIP servers is documented in the Internet draft *Common Gateway Interface for SIP*, `draft-lennox-sip-cgi-00`.

8 Acknowledgements

The material in this document has benefitted from discussions with Jonathan Rosenberg, Jonathan Lennox and the research staff of MCI.

References

[1] B. Braden, "Requirements for internet hosts - application and support," Request for Comments (Standard) STD 3, 1123, Internet Engineering Task Force, Oct. 1989. (Updated by RFC2181).

[2] M. Handley, H. Schulzrinne, and E. Schooler, "SIP: Session initiation protocol," Internet Draft, Internet Engineering Task Force, Feb. 1998. Work in progress.

[3] H. Schulzrinne and J. Rosenberg, "SIP call control services," Internet Draft, Internet Engineering Task Force, Feb. 1998. Work in progress.

[4] H. Schulzrinne and J. Rosenberg, "Signaling for internet telephony," Technical Report CUCS-005-98, Columbia University, New York, New York, Feb. 1998.

[5] V. Jacobson and M. Handley, "SDP: session description protocol," Internet Draft, Internet Engineering Task Force, Jan. 1998. Work in progress.

[6] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," Internet Draft, Internet Engineering Task Force, Jan. 1998. Work in progress.

[7] Microsoft Corporation, "Tapi: Windows telephony application programming interface," 1998.

[8] Sun Microsystems, "The java telephony API," Feb. 1997.

[9] D. Robinson and K. Coar, "The WWW common gateway interface: Version 1.2," Internet Draft, Internet Engineering Task Force, Feb. 1998. Work in progress.