Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-iptel-cpl-01.ps

IPTEL WG
Lennox/Schulzrinne
Columbia University

March 10, 2000
Expires: September 2000

CPL: A Language for User Control of Internet Telephony Services

Status of this Memo

Copyright Notice

Abstract

The Call Processing Language (CPL) is a language that can be used to describe and control Internet telephony services. It is designed to be implementable on either network servers or user agent servers. It is meant to be simple, extensible, easily edited by graphical clients, and independent of operating system or signalling protocol. It is suitable for running on a server where users may not be allowed to execute arbitrary programs, as it has no variables, loops, or ability to run external programs.

This document is a product of the IP Telephony (IPTEL) working group of the Internet Engineering Task Force. Comments are solicited and should be addressed to the working group's mailing list at iptel@lists.research.bell-labs.com and/or the authors.

Contents

1 Introduction

The Call Processing Language (CPL) is a language that can be used to describe and control Internet telephony services. It is not tied to any particular signalling architecture or protocol; it is anticipated that it will be used with both SIP [1] and H.323 [2].

The CPL is powerful enough to describe a large number of services and features, but it is limited in power so that it can run safely in Internet telephony servers. The intention is to make it impossible for users to do anything more complex (and dangerous) than describing Internet telephony services. The language is not Turing-complete, and provides no way to write loops or recursion.

The CPL is also designed to be easily created and edited by graphical tools. It is based on XML [3], so parsing it is easy and many parsers for it are publicly available. The structure of the language maps closely to its behavior, so an editor can understand any valid script, even ones written by hand. The language is also designed so that a server can easily confirm scripts' validity at the time they are delivered to it, rather that discovering them while a call is being processed.

Implementations of the CPL are expected to take place both in Internet telephony servers and in advanced clients; both can usefully process and direct users' calls. In the former case, a mechanism will be needed to transport scripts between clients and servers; this document does not describe such a mechanism, but related documents will.

The framework and requirements for the CPL architecture are described in the document "Call Processing Language Framework and Requirements," which will be an Informational RFC; it is currently available as the Internet-Draft `draft-ietf-iptel-cpl-framework-02` [4].

1.1 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 [5] and indicate requirement levels for compliant CPL implementations.

In examples, non-XML strings such as `-action1-`, `-action2-`, and so forth, are sometimes used. These represent further parts of the script which are not relevant to the example in question.

> Some paragraphs are indented, like this; they give motivations of design choices, or questions for future discussion in the development of the CPL, and are not essential to the specification of the language.

2 Structure of CPL scripts

2.1 High-level structure

A CPL script consists of two types of information: *ancillary information* about the script, and *call processing actions.*

A call processing action is a structured tree that describes the decisions and actions a telephony signalling server performs on a call set-up event. There are two types of call processing actions: *top-level actions* are actions that are triggered by signalling events that arrive at the server. Two top-level action names are defined: incoming, the action performed when a call arrives whose destination is the owner of the script; and outgoing, the action performed when a call arrives whose originator is the owner of the script. *Sub-actions* are actions which can be called from other actions. The CPL forbids sub-actions from being called recursively: see section 8.

Note: The names “action,” “sub-action,” and “top-level action” are probably not ideal. Suggestions for better names for these concepts are welcomed.

Ancillary information is information which is necessary for a server to correctly process a script, but which does not directly describe any actions. Currently, the only type of ancillary information defined is timezone definitions; see section 9.

2.2 Abstract structure of a call processing action

Abstractly, a call processing action is described by a collection of nodes, which describe actions that can be performed or choices which can be made. A node may have several parameters, which specify the precise behavior of the node; they usually also have outputs, which depend on the result of the condition or action.

For a graphical representation of a CPL action, see figure 1. Nodes and outputs can be thought of informally as boxes and arrows; the CPL is designed so that actions can be conveniently edited graphically using this representation. Nodes are arranged in a tree, starting at a single root node; outputs of nodes are connected to additional nodes. When an action is run, the action or condition described by the top-level node is performed; based on the result of that node, the server follows one of the node’s outputs, and that action or condition is performed; this process continues until a node with no specified outputs is reached. Because the graph is acyclic, this will occur after a bounded and predictable number of nodes are visited.

If an output to a node is not specified, it indicates that the CPL server should perform a node- or protocol-specific action. Some nodes have specific default actions associated with them; for others, the default action is implicit in the underlying signalling protocol, or can be configured by the administrator of the server. For further details on this, see section 10.

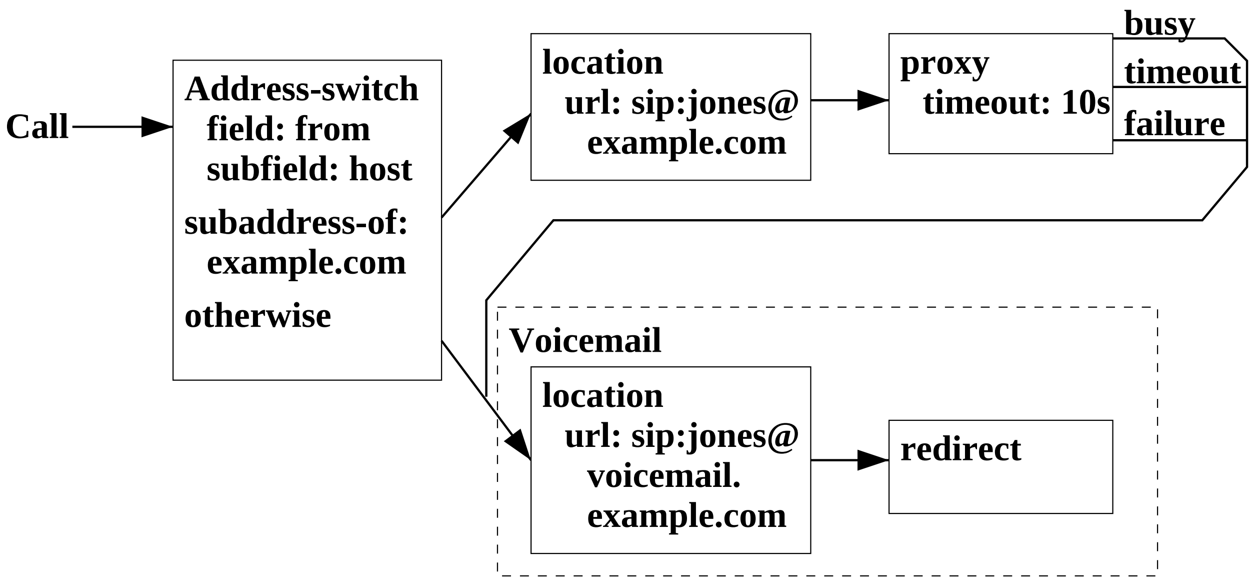

Figure 1: Sample CPL Action: Graphical Version

2.3 Location model

For flexibility, one piece of information necessary for the function of a CPL is not given as node parameters: the set of locations to which a call is to be directed. Instead, this set of locations is stored as an implicit global variable throughout the execution of a processing action (and its sub-actions). This allows locations to be retrieved from external sources, filtered, and so forth, without requiring general language support for such actions (which could harm the simplicity and tractability of understanding the language). The specific actions which add, retrieve, or filter location sets are given in section 5.

For the incoming top-level processing action, the location set is initialized to the empty set. For the outgoing action, it is initialized to the destination address of the call.

2.4 XML structure

Syntactically, CPL scripts are represented by XML documents. XML is thoroughly specified by [3], and implementors of this specification should be familiar with that document, but as a brief overview, XML consists of a hierarchical structure of tags; each tag can have a number of attributes. It is visually and structurally very similar to HTML [6], as both languages are simplifications of the earlier and larger standard SGML [7].

See figure 2 for the XML document corresponding to the graphical representation of the CPL script in figure 1. Both nodes and outputs in the CPL are represented by XML tags; parameters are represented by XML tag attributes. Typically, node tags contain output tags, and vice-versa (with one exception; see section 2.3).

The connection between the output of a node and another node is represented by enclosing the tag representing the pointed-to node inside the tag for the outer node's output. Convergence (several outputs pointing to a single node) is represented by sub-actions, discussed further in section 8.

The higher-level structure of a CPL script is represented by tags corresponding to each piece of meta-information, sub-actions, and top-level actions, in order. This higher-level information is all enclosed in a special tag cpl, the outermost tag of the XML document.

A complete Document Type Declaration for the CPL is provided in Appendix A. The remainder of the main sections of this document describe the semantics of the CPL, while giving its syntax informally. For the formal syntax, please see the appendix.

3 Script structure: overview

As mentioned, a CPL script consists of ancillary information, subactions, and top-level actions. The full syntax of the cpl node is given in figure 3.

Call processing actions, both top-level actions and sub-actions, consist of nodes and outputs. Nodes and outputs are both described by XML tags. There are four categories of CPL nodes: *switches*, *location modifiers*, *signalling actions*, and *non-signalling actions*.

4 Switches

Switches represent choices a CPL script can make, based on either attributes of the original call request or items independent of the call.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl SYSTEM "cpl.dtd">

<cpl>
  <subaction id="voicemail">
    <location url="sip:jones@voicemail.example.com">
      <redirect />
    </location>
  </subaction>

  <incoming>
    <address-switch field="origin" subfield="host">
      <address subdomain-of="example.com">
        <location url="sip:jones@example.com">
          <proxy>
            <busy> <sub ref="voicemail" /> </busy>
            <noanswer> <sub ref="voicemail" /> </noanswer>
            <failure> <sub ref="voicemail" /> </failure>
          </proxy>
        </location>
      </address>
      <otherwise>
        <sub ref="voicemail" />
      </otherwise>
    </address-switch>
  </incoming>
</cpl>
```

Figure 2: Sample CPL Script: XML Version

Node:	cpl	
Parameters:	none	
Outputs:	timezone	See section 9
	subaction	See section 8
	outgoing	Top-level actions to take on this user’s outgoing calls
	incoming	Top-level actions to take on this user’s incoming calls
Output:	outgoing	
Parameters:	none	
Output:	incoming	
Parameters:	none	

Figure 3: Syntax of the top-level cpl tag

All switches are arranged as a list of conditions that can match a variable. Each condition corresponds to a node output; the output points to the next node to execute if the condition was true. The conditions are tried in the order they are presented in the script; the output corresponding to the first node to match is taken.

There are two special switch outputs that apply to every switch type. The output not-present is true if the variable the switch was to match was not present in the original call. The output otherwise, which MUST be the last output specified, matches if no other condition matched.

If no condition matches and no otherwise output was present in the script, the default script action is taken. See section 10 for more information on this.

4.1 Address switches

Address switches allow a CPL script to make decisions based on one of the addresses present in the original call request. They are summarized in figure 4.

Node:	address-switch	
Outputs:	address	Specific addresses to match
Parameters:	field	origin, destination, or original-destination
	subfield	address-type, user, host, port, tel, display, password, or asn1
Output:	address	
Parameters:	is	exact match
	contains	substring match (for display only)
	subdomain-of	sub-domain match (for host, tel only)

Figure 4: Syntax of the address-switch node

Address switches have two node parameters: field, and subfield. The mandatory field parameter allows the script to specify which address is to be considered for the switch: either the call's origin address (field "origin"), its current destination address (field "destination"), or its original destination (field "original-destination"), the destination the call had before any earlier forwarding was invoked. Servers MAY define additional subfield values.

The optional subfield specifies what part of the address is to be considered. The possible subfield values are: address-type, user, host, port, tel, and display. Additional subfield values MAY be defined: two additional ones, password and asn1 are defined specifically for SIP and H.323 respectively, in sections 4.1.1 and 4.1.2 below. If no subfield is specified, the "entire" address is matched; the precise meaning of this is defined for each underlying signalling protocol.

The subfields are defined as follows:

address-type This indicates the type of the underlying address; i.e., the URI scheme, if the address can be represented by the URI. The types specifically discussed by this document are sip, tel, and h323. The address type is not case-sensitive; it is always present if the address is present.

user This subfield of the address indicates, for e-mail style addresses, the user part of the address. For telephone number style address, it includes the subscriber number. This subfield is case-sensitive; it may be not present.

host This subfield of the address indicates the Internet host name or IP address corresponding to the address, in host name, IPv4, or IPv6 format. For host names only, subdomain matching is supported with the subdomain-of match operator. It is not case sensitive, and may be not present.

port This subfield indicates the TCP or UDP port number of the address, numerically in decimal format. It is not case sensitive, as it MUST only contain decimal digits. It may be not present; however, for address types with default ports, an absent port matches the default port number.

tel This subfield indicates a telephone subscriber number, if the address contains such a number. It is not case sensitive (the telephone numbers may contain the symbols 'A' 'B' 'C' and 'D'), and may be not present. It may be matched using the subdomain-of match operator. Punctuation and separator characters in telephone numbers are discarded.

display This subfield indicates a "display name" or user-visible name corresponding to an address. It is a Unicode string, and is matched using the case-insensitive algorithm described in section 4.2. The contains operator may be applied to it. It may be not present.

For any completely unknown subfield, the server MAY reject the script at the time it is submitted with an indication of the problem; if a script with an unknown subfield is executed, the server MUST consider the not-present output to be the valid one.

The address output tag may take exactly one of three possible parameters, indicating the kind of matching allowed.

is An output with this match operator is followed if the subfield being matched in the address-switch exactly matches the argument of the operator. It may be used for any subfield, or for the entire address if no subfield was specified.

subdomain-of This match operator applies only for the subfields host and tel. In the former case, it matches if the hostname being matched is a subdomain of the domain given in the argument of the match operator; thus, `match="example.com"` would match the hostnames "example.com", "research.example.com", and "zaphod.sales.internal.example.com". IP addresses may be given as arguments to this operator; however, they only match exactly. In the case of the tel subfield, the output matches if the telephone number being matched has a prefix that matches the argument of the match operator; match="1212555" would match the telephone number "1 212 555 1212."

contains This match operator applies only for the subfield display. The output matches if the display name being matched contains the argument of the match as a substring.

4.1.1 Address switch mapping for SIP

For SIP, the origin address corresponds to the address in the From header; destination corresponds to the Request-URI; and original-destination corresponds to the To header.

The display subfield of an address is the display-name part of the address, if it is present. Because of SIP's syntax, the destination address field will never have a display subfield.

The address-type subfield of an address is the URI scheme of that address. Other address fields depend on that address-type.

For sip URLs, the user, host, and port subfields correspond to the "user," "host," and "port" elements of the URI syntax. The tel subfield is defined to be the "user" part of the URI if and only if the "user=phone" parameter is given to the URI. An additional subfield, password is defined to correspond to the "password" element of the SIP URI; however, use of this field is NOT RECOMMENDED for general security reasons.

For tel URLs, the tel and user subfields are the subscriber name; in the former case, "noise" characters are stripped. the host and port subfields are both not present.

For other URI schemes, only the address-type subfield is defined by this specification; servers MAY set others of the pre-defined subfields, or MAY support additional subfields.

If no subfield is specified for addresses in SIP messages, the string matched is the URI part of the address, with all parameters stripped.

4.1.2 Address switch mapping for H.323

For H.323, the origin address corresponds to the address in the sourceAddress field; both destination and original-destination correspond to the destinationAddress field, as H.323 has no indication of original destination.

For all addresses in H.323 messages, the value of the address-type field is h323. The tel tag is set to the AliasAddress, if its type is e164. The user tag is set to h323-ID; host is set to transportID/TransportAddress/ipAddress, translated to a dotted-quad; port is set to transportID/TransportAddress/ipAddress/port. The display tag is not present. An additional subfield, asn1, is defined as the textually-encoded ASN.1 of the address. The matching if no subfield is specified is undefined at this time.

TODO: Have this looked over by an H.323 expert for accuracy/completeness. Once an h323 URL scheme is defined, it should be used for the whole-address matching.

4.2 String switches

String switches allow a CPL script to make decisions based on free-form Unicode strings present in a call request. They are summarized in figure 5.

Node:	string-switch	
Outputs:	string	Specific string to match
Parameters:	field	subject, organization, or user-agent
Output:	string	
Parameters:	is	exact match
	contains	substring match

Figure 5: Syntax of the string-switch node

String switches have one node parameter: field. The mandatory field parameter specifies which string is to be matched. Currently three fields are defined: subject, indicating the subject of the call; organization, indicating the originator's organization; and user-agent, indicating the program or device with which the call request was made. All these fields correspond to SIP strings.

TODO: Need H.323 free-form strings. "Data"?

Strings are matched as case-insensitive Unicode strings, in the following manner. First, strings are canonicalized to the "Compatibility Composition" (KC) form, as specified in Unicode Technical Report 15 [8]. Then, strings are compared using locale-insensitive caseless mapping, as specified in Unicode Technical Report 21 [9].

> Code to perform the first step, in Java and Perl, is available; see the links from Annex E of UTR 15 [8]. The case-insensitive string comparison in the Java standard class libraries already performs the second step; other Unicode-aware libraries should be similar.

The output tags of string matching are named string, and have a mandatory argument, one of is or contains, indicating whole-string match or substring match, respectively.

4.3 Time switches

Time switches allow a CPL script to make decisions based the time and/or date the script is being executed. They are summarized in figure 6.

Node:	time-switch	
Outputs:	time	Specific time to match
Parameters:	timezone	local, utc, or other (see section 9)
Output:	time	
Parameters:	year	Years to match
	month	Months to match
	date	Days of month to match
	weekday	Days of week to match
	timeofday	Times of day to match

Figure 6: Syntax of the time-switch node

Time switches take one optional parameter, timezone, which specifies the time zone in which matching is to take place. Two values of this are predefined: local indicates the time zone in which the server is located, and utc indicates Universal Coordinated Time. Timezones may also be specified in the ancillary information; see section 9.

The time outputs can take the following optional parameters: year, month, date, day, and timeofday. Each argument is syntactically expressed as a list of numeric ranges. Ranges are delimited as value-value; lists elements are separated by commas. Months are specified in the range 1-12; date as 1-31, day as 0-6 (where 0 is Sunday), and times of day as 24-hour times in the range 0000-2359; years are unlimited in range, though only positive values are allowed.

An output node matches if the time the triggering call was placed falls within one of specified the ranges in each of the specified parameters.

The following examples show sample time nodes, and descriptions of the corresponding time periods they indicate:

```
<time month="12" date="25" year="1999">
```

December 25th, 1999, all day

`<time month="5" date="4">`
May 4th, every year, all day

`<time day="1-5" timeofday="0900-1700">`
9 AM – 5 PM, Monday through Friday, every week

`<time timeofday="1310-1425,1440-1555,1610-1725" day="2,4">`
1:10 – 2:25 PM, 2:40 – 3:55 PM, and 4:10 – 5:25 PM, Tuesdays and Thursdays, every week

`<time date="1-7" day="1">`
The first Monday of every month, all day

If more complicated time ranges need to be specified, they SHOULD be broken down into component ranges specifiable in this syntax, and their outputs connected the outputs to the same subsequent node with subactions (see section 8).

The not-present output is never true for a time switch.

Note: XML schemas [10] define their own "time instant" and "time duration" syntax. Would it be better to base this syntax on that? It doesn't seem to be quite as powerful.

Note: the question of whether the week should start at Sunday or Monday, and of whether numbering starts at 0 or 1, was a matter of some dispute. In the absence of any convincing argument in favor of any one proposal, the current choice (Sunday is 0) was chosen semi-arbitrarily, because it corresponds to the `tm_wday` field of C's `struct tm`.

Note: the way of specifying "first Monday of month" and "last Monday of month" is awfully hackish. Would it be worthwhile to add a week parameter, which could optionally be negative to count from the end of the month?

4.4 Priority switches

Priority switches allow a CPL script to make decisions based on the priority specified for the original call. They are summarized in figure 7.

Node:	priority-switch	
Outputs:	priority	Specific priority to match
Parameters:	none	
Output:	priority	
Parameters:	less	Match if priority is less than specified
	greater	Match if priority is greater than specified
	equal	Match if priority is equal to specified

Figure 7: Syntax of the priority-switch node

Priority switches take no parameters.

The priority tags take one of the three parameters greater, less, and equal. The values of these tags are the priorities specified in SIP [1]: in decreasing order, emergency, urgent, normal, and non-urgent. These values are matched in a case-insensitive manner. Outputs with the less parameter are taken if the priority of the message is less than the priority given in the argument; and so forth.

If no priority header is specified in a message, the priority is considered to be normal. If an unknown priority is given, the priority is considered to be equivalent to normal for the purposes of greater and less comparisons, but it is compared literally for equal comparisons.

Since every message has a priority, the not-present output is never true for a priority switch.

5 Location modifiers

The abstract location model of the CPL is described in section 2.3. The behavior of several of the signalling actions (defined in section 6) is dependent on the current location set specified. Location nodes add to or remove locations from the location set.

There are three types of location nodes defined. *Explicit locations* add literally-specified locations to the current location set; *location lookups* obtain locations from some outside source; and *location filters* remove locations from the set, based on some specified criteria.

5.1 Explicit location

Explicit location nodes specify a location literally. Their syntax is described in figure 8.

Node:	location	
Outputs:	any node	
Parameters:	url	URL of address to add to location set

Figure 8: Syntax of the location node

Explicit location nodes have one node parameter: url, whose value is the URL of the address to add to the location set. Only one address may be specified per location node; multiple locations may be specified by cascading these nodes.

Basic location nodes have only one possible output, since there is no way that they can fail. (If a basic location node specifies a location which isn't supported by the underlying signalling protocol, the script server SHOULD detect this and report it to the user at the time the script is submitted.) Therefore, its XML representation does not have explicit output nodes; the `<location>` tag directly contains another node tag.

5.2 Location lookup

Locations can also be specified up through external means, through the use of location lookups. The syntax of these tags is given in figure 9.

Location lookup nodes have one mandatory parameter, and three optional parameters. The mandatory parameter is source, the source of the lookup. This can either be a URL, or a non-URL value. If the value of source is a URL, it indicates a location which returns the `application/url` media type. The server adds the locations returned by the URL to the location set.

Non-URL sources indicate a source not specified by a URL which the server can query for addresses to add to the location set. The only non-URL source currently defined is registration, which specifies all the locations currently registered with the server, using SIP REGISTER or H.323 RAS messages.

Node:	lookup	
Outputs:	success	Action if lookup was successful
	notfound	Action if lookup found no addresses
	failure	Action if lookup failed
Parameters:	source	Source of the lookup
	timeout	Time to try before giving up on the lookup
	use	Caller preferences fields to use
	ignore	Caller preferences fields to ignore

Output:	success
Parameters:	none

Output:	notfound
Parameters:	none

Output:	failure
Parameters:	none

Figure 9: Syntax of the lookup node

The lookup node also has an three optional parameters. The timeout parameter which specifies the time, in seconds, the script is willing to wait for the lookup to be performed. If this is not specified, its default value is 30.

The other two optional parameters affect the interworking of the CPL script with caller preferences and caller capabilities. These are defined in the Internet-Draft "SIP Caller Preferences and Callee Capabilities" [11]. By default, a CPL server SHOULD invoke caller preferences filtering when performing a lookup action. The two parameters use and ignore allow the script to modify how the script applies caller preferences filtering. The use and ignore parameters both take as their arguments comma-separated lists of caller preferences parameters. If use is given, the server applies the caller preferences resolution algorithm only to those preference parameters given in the use parameter, and ignores all others; if the ignore parameter is given, the server ignores the specified parameters, and uses all the others. Only one of use and ignore can be specified. The addr-spec part of the caller preferences is always applied, and the script cannot modify it.

Note: this is very SIP-specific. Does H.323 have a similar endpoint-capabilities and requested-capabilities mechanism?

TODO: Add examples. This is confusing.

Lookup has three outputs: success, notfound, and failure. Notfound is taken if the lookup process succeeded but did not find any locations; failure is taken if the lookup failed for some reason, including that specified timeout was exceeded. If failure is not specified, the action corresponding to notfound is taken; if notfound is not specified, the success output is taken, but the current location set is not modified. The success output MUST be included.

Clients SHOULD specify the three outputs success, notfound, and failure in that order, so their script complies with the DTD given in Appendix A, but servers SHOULD accept them in any order.

5.3 Location filtering

A CPL script can also filter addresses out of the address set, through the use of a mechanism very similar to caller preferences: the remove-location node. The syntax of these nodes is defined in figure 10.

Node:	remove-location	
Outputs:	any node	
Parameters:	param	Caller preference parameter to apply
	value	Value of caller preference parameter
	location	Caller preference location to apply

Figure 10: Syntax of the remove-location node

A remove-location node has the same effect on the location set as a Reject-Contact header in caller preferences [11]. The value of the location parameter is treated as though it were the addr-spec field of a Reject-Contact header; an absent header is equivalent to an addr-spec of "*" in that specification. If param and value are present, their values are comma-separated lists of caller preferences parameters and corresponding values, respectively. There MUST be the same number of parameters as values specified. These are treated, for location filtering purposes, as though they appeared in the params field of a Reject-Location header, as "; param=value" for each one.

Note: do we want to be able to switch based on whether there are any locations left in the set after a lookup?

Note: this is also very SIP-specific. Does H.323 have a similar endpoint-capabilities mechanism?

TODO: Add examples. This is also confusing.

6 Signalling actions

Signalling action nodes cause signalling events in the underlying signalling protocol. Three signalling actions are defined: "proxy," "redirect," and "reject."

6.1 Proxy

Proxy causes the triggering call to be forwarded on to the currently specified set of locations. The syntax of the proxy node is given in figure 11.

After a proxy action has completed, the CPL server chooses the "best" response to the call attempt, as defined by the signalling protocol or the server's administrative configuration rules.

If the call attempt was successful, or if a redirection response was the "best" response and recurse was not specified, CPL execution terminates and the best response is forwarded back upstream to the originator. Otherwise, one of the three outputs busy, noanswer, or failure is taken.

Note: future extension of the CPL to allow in-call or end-of-call actions will require success outputs to be added.

Question: should an explicit redirection output be added for the case when recurse was false? How should it interact with the location set?

Node:	lookup	
Outputs:	busy	Action if call attempt returned "busy"
	noanswer	Action if call attempt was not answered before timeout
	failure	Action if call attempt failed
Parameters:	timeout	Time to try before giving up on the call attempt
	recurse	Whether to recursively look up redirections
	ordering	What order to try the location set in.
Output:	busy	
Parameters:	none	
Output:	noanswer	
Parameters:	none	
Output:	failure	
Parameters:	none	

Figure 11: Syntax of the proxy node

Proxy has three optional parameters. The timeout parameter specifies the time, in seconds, to wait for the call to be completed or rejected; after this time has elapsed, the call attempt is terminated and the noanswer branch is taken. If this parameter is not specified, the default value is 20 seconds if the proxy node has a no-answer output specified; otherwise the server SHOULD allow the call to ring for an arbitrarily long period of time.

Question: is 20 seconds a good value? How should such a value be chosen?

The second optional parameter is recurse, which can take two values, yes or no. This specifies whether the server should automatically attempt to place further call attempts to telephony addresses in redirection responses that were returned from the initial server.

The third optional parameter is ordering. This can have three possible values: parallel, sequential, and first-only. This parameter specifies in what order the locations of the location set should be tried. Parallel asks that they all be tried simultaneously; sequential asks that the first one be tried first, the second second, and so forth, until one succeeds or the set is exhausted; first-only instructs the server to try only the first address in the set, and then follow one of the outputs. The default value of this parameter is parallel.

Once a proxy action completes, if control is passed on to other actions, all locations which have been used are cleared from the location set. That is, the location set is emptied if ordering was parallel or sequential; the first item in the set is removed from the set if ordering was first-only.

For the proper actions when outputs are unspecified, see section 10.

6.2 Redirect

Redirect causes the server to direct the calling party to attempt to place its call to the currently specified set of locations. The syntax of this node is specified in figure 12.

Redirect immediately terminates execution of the CPL script, so this node has no outputs. It also takes no arguments.

Node:	redirect	
Outputs:	none	
Parameters:	none	

Figure 12: Syntax of the redirect node

Question: should there be some way of distinguishing between “moved temporarily” and “moved permanently” (SIP 301 and 302) redirections?

6.3 Reject

Reject nodes cause the server to reject the call attempt. Their syntax is given in figure 13.

Node:	reject	
Outputs:	none	
Parameters:	status	Status code to return
	reason	Reason phrase to return

Figure 13: Syntax of the reject node

This immediately terminates execution of the CPL script, so this node has no outputs.

This node has two arguments: status and reason. The status argument is required, and can take one of the values busy, notfound, reject, and error. Servers which implement SIP MAY also allow a numeric argument corresponding to a SIP status in the 4xx, 5xx, or 6xx range, but scripts SHOULD NOT use them if they wish to be portable.

The reason argument optionally allows the script to specify a reason for the rejection. CPL servers MAY ignore the reason, but ones that implement SIP SHOULD send them in the SIP reason phrase.

7 Other actions

In addition to the signalling actions, the CPL defines several actions which do not affect the telephony signalling protocol.

7.1 Mail

The mail node causes the server to notify a user of the status of the CPL script through electronic mail. Its syntax is given in figure 14.

Node:	mail	
Outputs:	any node	
Parameters:	url	Mailto url to which the mail should be sent

Figure 14: Syntax of the mail node

The mail node takes one argument: a `mailto` URL giving the address, and any additional desired parameters, of the mail to be sent. The server sends the message containing the content to the given url; it SHOULD also include other status information about the state of the call and the CPL script at the time of the notification.

Mail nodes have only one output, since failure of e-mail delivery cannot reliably be known in real-time. Therefore, its XML representation does not have explicit output nodes: the `<mail>` tag directly contains another node tag.

> Using a full `mailto` URL rather than just an e-mail address allows additional e-mail headers to be specified, such as `<mail url="mailto:jones@example.com;subject=lookup%20failed" />`.

7.2 Log

The Log node causes the server to log information about the call to non-volatile storage. Its syntax is specified in figure 15.

Node:	log	
Outputs:	any node	
Parameters:	name	Name of the log file to use
	comment	Comment to be placed in log file

Figure 15: Syntax of the log node

Log takes two arguments, both optional: name, which specifies the name of the log, and comment, which gives a comment about the information being logged. Servers SHOULD also include other information in the log, such as the time of the logged event, information that triggered the call to be logged, and so forth. Logs are specific to the owner of the script which log event. If the name parameter is not given, the event is logged to a standard, server-defined logfile for the script owner. This specification does not define how users may retrieve their logs from the server.

A correctly operating CPL server SHOULD NOT ever allow the log event to fail. As such, log nodes have only one output, and their XML representation does not have explicit output nodes. A CPL `<log>` tag directly contains another node tag.

8 Subactions

XML syntax defines a tree. To allow more general call flow diagrams, and to allow script re-use and modularity, we define subactions.

Two tags are defined for subactions: subaction definitions and subaction references. Their syntax is given in figure 16.

Subactions are defined through subaction tags. These tags are placed in the CPL after any ancillary information (see section 9) but before any top-level tags. They take one argument: id, a token indicating a script-chosen name for the subaction.

Subactions are called from sub tags. The sub tag is a "pseudo-node": it can be used anyplace in a CPL action that a true node could be used. It takes one parameter, ref, the name of the subaction to be called. The sub tag contains no outputs of its own; control instead passes to the subaction.

Tag:	subaction	
Subtags:	any node	
Parameters:	id	Name of this subaction
Pseudo-node:	sub	
Outputs:	none in XML tree	
Parameters:	ref	Name of subaction to execute

Figure 16: Syntax of subactions and sub pseudo-nodes

References to subactions MUST refer to subactions defined before the current action. A sub tag MUST NOT refer to the action which it appears in, or to any action defined later in the CPL script. Top-level actions cannot be called from sub tags, or through any other means. Script servers MUST verify at the time the script is submitted that no sub node refers to any sub-action which is not its proper predecessor.

Allowing only back-references of subs forbids any sort of recursion. Recursion would introduce the possibility of non-terminating or non-decidable CPL scripts, a possibility our requirements specifically excluded.

Every sub MUST refer to a subaction ID defined within the same CPL script. No external links are permitted.

If any subsequent version ever defines external linkages, it will use a different tag, perhaps XLink [12]. Ensuring termination in the presence of external links is a difficult problem.

9 Ancillary information

Only one sort of ancillary information is currently defined for CPL scripts: timezone information. The syntax of timezone specifications is given in figure 17.

Timezone specifications consist, conceptually, of three parts: the name of the timezone, as used by time switches in the script; the GMT offset and abbreviation of each offset used in the timezone; and the instants at which each offset takes effect.

The name of the timezone is given by the name parameter to the timezone tag. This is the name which time-switch tags can specify in their timezone parameter.

The timezone tag must contain at least one instance of the standard tag, which has mandatory arguments offset, giving the zone's offset in minutes from UTC, and abbr, giving the standard abbreviation of the timezone. If more than one time offset is in use in a timezone during a year, the timezone tag contains another tag, daylight, which takes the same parameters as standard; and each of standard and daylight has parameters, using the same syntax as time-switch tags (section 4.3, specifying a set of instants when the time zone rule takes effect, in the local time of the other offset.

Currently only two classes of offsets are supported. A timezone rule MAY contain several definitions each of standard and daylight if, for instance, different rules are in effect for different years.

Figure 18 shows the timezone specification for most of the eastern United States.

Figure 19 shows a simpler timezone rule for the state of Arizona, United States; most of Arizona does not observe daylight savings time.

Note: the syntax for specifying the first or last weekday of a month is very clumsy. A proper week parameter might be a good thing to add.

Tag:	timezone	
Parameters:	name	Name of this timezone
Outputs:	standard	Specification of standard time
	daylight	Specification of daylight (summer) time
Tag:	standard	
Parameters:	offset	UTC offset during standard time
	abbr	abbreviation of this timezone
	year	year that this timezone transition occurs
	month	month that this timezone transition occurs
	date	day of month that this timezone transition occurs
	weekday	weekday that this timezone transition occurs
	timeofday	time of day that this timezone transition occurs
Tag:	daylight	
Parameters:	. . .	same as for standard

Figure 17: Syntax of the timezone tag

```
<timezone name="US/Eastern">
  <standard offset="-0500" abbr="EST" month="10" date="25-31"
        day="0" timeofday="0200" />
  <!-- 2 AM, last Sunday in October -->
  <daylight offset="-0400" abbr="EDT" month="4" date="1-7"
        day="0" timeofday="0200" />
  <!-- 2 AM, first Sunday in April -->
</timezone>
```

Figure 18: Timezone rule for the eastern United States.

10 Default actions

When a CPL action reaches an unspecified output, the action it takes is dependent on the current state of script execution. This section gives the actions that should be taken in each case.

no location or signalling actions performed, location set empty: Look up the user's location through whatever mechanism the server would use if no CPL script were in effect. Proxy, redirect, or send a

```
<timezone name="US/Arizona">
  <standard offset="-0700" abbr="MST" />
</timezone>
```

Figure 19: Timezone rule for Arizona, United States.

rejection message, using whatever policy the server would use in the absence of a CPL script.

no location or signalling actions performed, location set non-empty: (This can only happen for outgoing calls.) Proxy the call to the addresses in the location set.

location actions performed, no signalling actions: Proxy or redirect the call, whichever is the server's standard policy, to the addresses in the current location set. If the location set is empty, return not-found rejection.

noanswer output of proxy, no timeout given: (This is a special case.) If the noanswer output of a proxy node is unspecified, and no timeout parameter was given to the proxy node, the call should be allowed to ring for the maximum length of time allowed by the server (or the request, if the request specified a timeout).

proxy action previously taken: Return whatever the "best" response is of all accumulated responses to the call to this point, according to the rules of the underlying signalling protocol.

11 Examples

TODO: these examples don't illustrate many of the new features added to the CPL in draft -01. Add these.

11.1 Example: Call Redirect Unconditional

The script in figure 20 is a simple script which redirects all calls to a single fixed location.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl SYSTEM "cpl.dtd">

<cpl>
  <incoming>
    <location url="sip:smith@phone.example.com">
     <redirect />
    </location>
  </incoming>
</cpl>
```

Figure 20: Example Script: Call Redirect Unconditional

11.2 Example: Call Forward Busy/No Answer

The script in figure 21 illustrates some more complex behavior. We see an initial proxy attempt to one address, with further actions if that fails. We also see how several outputs take the same action, through the use of subactions.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl SYSTEM "cpl.dtd">

<cpl>
  <subaction id="voicemail">
    <location url="sip:jones@voicemail.example.com" >
      <proxy />
    </location>
  </subaction>

  <incoming>
    <location url="sip:jones@jonespc.example.com">
       <proxy timeout="8s">
         <busy>
         </busy>
         <noanswer>
           <sub ref="voicemail" />
         </noanswer>
       </proxy>
    </location>
  </incoming>
</cpl>
```

Figure 21: Example Script: Call Forward Busy/No Answer

11.3 Example: Call Screening

The script in figure 22 illustrates address switches and call rejection, in the form of a call screening script. Note also that because the address-switch lacks an otherwise clause, if the initial pattern did not match, the script does not define any action. The server therefore proceeds with its default action, which would presumably be to contact the user.

11.4 Example: Time-of-day Routing

Figure 23 illustrates time-based conditions and timezones.

11.5 Example: Non-call Actions

Figure 24 illustrates non-call actions; in particular, alerting a user by electronic mail if the lookup server failed. The primary reason for the mail node is to allow this sort of out-of-band notification of error conditions, as the user might otherwise be unaware of any problem.

11.6 Example: A Complex Example

Finally, figure 25 is a complex example which shows the sort of sophisticated behavior which can be achieved by combining CPL nodes. In this case, the user attempts to have his calls reach his desk; if he

```
<?xml version="1.0" ?>
<!DOCTYPE cpl SYSTEM "cpl.dtd">

<cpl>
  <incoming>
    <address-switch field="origin" subfield="user">
      <address is="anonymous">
         <reject status="reject"
            reason="I don’t accept anonymous calls" />
      </address>
    </address-switch>
  </incoming>
</cpl>
```

Figure 22: Example Script: Call Screening

does not answer within a small amount of time, calls from his boss are forwarded to his celphone, and all other calls are directed to voicemail.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl SYSTEM "cpl.dtd">

<cpl>
  <timezone name="US/Eastern">
    <standard offset="-0500" abbr="EST" month="10" date="25-31"
          day="0" timeofday="0200" />
    <!-- 2 AM, last Sunday in October -->
    <daylight offset="-0400" abbr="EDT" month="4" date="1-7"
          day="0" timeofday="0200" />
    <!-- 2 AM, first Sunday in April -->
  </timezone>

  <incoming>
    <time-switch timezone="US/Eastern">
      <time day="1-5" timeofday="0900-1700">
        <lookup source="registration">
          <success>
            <proxy />
          </success>
        </lookup>
      </time>
      <otherwise>
        <location url="sip:jones@voicemail.example.com">
          <proxy />
        </location>
      </otherwise>
    </time-switch>
  </incoming>
</cpl>
```

Figure 23: Example Script: Time-of-day Routing

```
<?xml version="1.0" ?>
<!DOCTYPE cpl SYSTEM "cpl.dtd">

<cpl>
  <incoming>
    <lookup source="http://www.example.com/cgi-bin/locate.cgi?user=jones"
            timeout="8">
      <success>
        <proxy />
      </success>
      <failure>
        <mail url="mailto:jones@example.com;subject=lookup%20failed" />
      </failure>
    </lookup>
  </incoming>
</cpl>
```

Figure 24: Example Script: Non-call Actions

```
<?xml version="1.0" ?>
<!DOCTYPE cpl SYSTEM "cpl.dtd">

<cpl>
  <subaction id="voicemail">
    <location url="sip:jones@voicemail.example.com">
      <redirect />
    </location>
  </subaction>

  <incoming>
    <location url="sip:jones@phone.example.com">
      <proxy timeout="8s">
        <busy>
          <sub ref="voicemail" />
        </busy>
        <noanswer>
          <address-switch field="origin">
            <address contains="boss@example.com">
              <location url="tel:+19175551212">
                <proxy />
              </location>
            </address>
            <otherwise>
              <sub ref="voicemail" />
            </otherwise>
          </address-switch>
        </noanswer>
      </proxy>
    </location>
  </incoming>
</cpl>
```

Figure 25: Example Script: A Complex Example

12 Security considerations

The CPL is designed to allow services to be specified in a manner which prevents potentially hostile or mis-configured scripts from launching security attacks, including denial-of-service attacks. Because script runtime is strictly bounded by acyclicity, and because the number of possible script actions are strictly limited, scripts should not be able to inflict damage upon a CPL server.

Because scripts can direct users' telephone calls, the method by which scripts are transmitted from a client to a server MUST be strongly authenticated. Such a method is not specified in this document.

Script servers SHOULD allow server administrators to control the details of what CPL actions are permitted.

13 Acknowledgments

We would like to thank Tom La Porta and Jonathan Rosenberg for their contributions and suggestions.

We drew a good deal of inspiration, notably the language's lack of Turing-completeness and the syntax of string matching, from the specification of Sieve [13], a language for user filtering of electronic mail messages.

A The XML DTD for CPL

This section includes a full DTD describing the XML syntax of the CPL. Every script submitted to a CPL server SHOULD comply with this DTD; however, CPL servers SHOULD allow minor variations from it, particularly in the ordering of output branches of nodes. Note that compliance with this DTD is not a sufficient condition for correctness of a CPL script, as many of the conditions described above are not expressible in DTD syntax.

```
<?xml version="1.0" encoding="US-ASCII" ?>

<!--
    Draft DTD for CPL, corresponding to
    draft-ietf-iptel-cpl-01.
-->

<!-- Top-level tags of the CPL -->
<!-- Ancillary information -->
<!ENTITY % Ancillary 'timezone' >

<!-- Subactions -->
<!ENTITY % Subactions 'subaction' >

<!-- Top-level actions -->
<!ENTITY % TopLevelAction 'incoming|outgoing' >


<!-- Nodes. -->
```

```
<!-- Switch nodes -->
<!ENTITY % Switch ’address-switch|string-switch|time-switch|
                  priority-switch’ >

<!-- Location nodes -->
<!ENTITY % Location ’location|lookup|remove-location’ >

<!-- Signalling action nodes -->
<!ENTITY % SignallingAction ’proxy|redirect|reject’ >

<!-- Other actions -->
<!ENTITY % OtherAction ’mail|log’ >

<!-- Links to subactions -->
<!ENTITY % Sub ’sub’ >

<!-- Nodes are one of the above four categories, or a subaction.
     This entity (macro) describes the contents of an output.
     Note that a node can be empty, implying default action. -->
<!ENTITY % Node     ’(%Location;|%Switch;|%SignallingAction;|
                     %OtherAction;|%Sub;)?’ >


<!-- Switches: choices a CPL script can make. -->

<!-- All switches can have an ’otherwise’ output. -->
<!ELEMENT otherwise ( %Node; ) >

<!-- All switches can have a ’not-present’ output. -->
<!ELEMENT not-present ( %Node; ) >

<!-- Address-switch makes choices based on addresses. -->
<!ELEMENT address-switch ( (address|not-present)+, otherwise? ) >
<!ATTLIST address-switch
   field         CDATA    #REQUIRED
   subfield      CDATA    #IMPLIED
>

<!ELEMENT address ( %Node; ) >
<!ATTLIST address
   is            CDATA    #IMPLIED
   contains      CDATA    #IMPLIED
   subdomain-of  CDATA    #IMPLIED
>
```

```
<!-- String-switch makes choices based on strings. -->

<!ELEMENT string-switch ( (string|not-present)+, otherwise? ) >
<!ATTLIST string-switch
   field         CDATA    #REQUIRED
>

<!ELEMENT string ( %Node; ) >
<!ATTLIST string
   is            CDATA    #IMPLIED
   contains      CDATA    #IMPLIED
>

<!-- Time-switch makes choices based on the current time. -->

<!ELEMENT time-switch ( (time|not-present)+, otherwise? ) >
<!ATTLIST time-switch
   timezone      CDATA    #IMPLIED
>

<!ELEMENT time ( %Node; ) >
<!ATTLIST time
   year          CDATA  #IMPLIED
   month         CDATA  #IMPLIED
   date          CDATA  #IMPLIED
   day           CDATA  #IMPLIED
   timeofday     CDATA  #IMPLIED
>


<!-- Priority-switch makes choices based on message priority. -->

<!ELEMENT priority-switch ( (priority|not-present)+, otherwise? ) >

<!ENTITY % PriorityVal '(emergency|urgent|normal|non-urgent)' >

<!ELEMENT priority ( %Node; ) >
<!ATTLIST priority
   less          %PriorityVal;  #IMPLIED
   greater       %PriorityVal;  #IMPLIED
   equal         CDATA          #IMPLIED
>
```

```
<!-- Locations: ways to specify the location a subsequent action
     (proxy, redirect) will attempt to contact. -->

<!ENTITY % Clear  'clear (yes|no) "no"' >

<!ELEMENT location ( %Node; ) >
<!ATTLIST location
   url           CDATA    #REQUIRED
   %Clear;
>

<!ELEMENT lookup ( success,notfound?,failure? ) >
<!ATTLIST lookup
  source         CDATA     #REQUIRED
  timeout        CDATA     "30"
  use            CDATA     #IMPLIED
  ignore         CDATA     #IMPLIED
  %Clear;
>

<!ELEMENT success  ( %Node; ) >
<!ELEMENT notfound ( %Node; ) >
<!ELEMENT failure ( %Node; ) >

<!ELEMENT remove-location ( %Node; ) >
<!ATTLIST remove-location
   param         CDATA    #IMPLIED
   value         CDATA    #IMPLIED
   location      CDATA    #IMPLIED
>


<!-- Signalling Actions: call-signalling actions the script can
     take. -->

<!ELEMENT proxy ( busy?,noanswer?,failure? ) >
<!ATTLIST proxy
   timeout       CDATA    "20"
   recurse       (yes|no) "yes"
   ordering      CDATA    "parallel"
>

<!ELEMENT busy ( %Node; ) >
<!ELEMENT noanswer ( %Node; ) >
<!-- "failure" repeats from lookup above.  XXX? -->
```

```
<!ELEMENT redirect EMPTY >


<!-- Statuses we can return -->

<!ELEMENT reject EMPTY >
<!ATTLIST reject
   status        CDATA    #REQUIRED
   reason        CDATA    #IMPLIED
>

<!-- Non-signalling actions: actions that don’t affect the call -->

<!ELEMENT mail ( %Node; ) >
<!ATTLIST mail
   url           CDATA    #REQUIRED
>

<!ELEMENT log ( success,failure? ) >
<!ATTLIST log
   name          CDATA    #IMPLIED
   comment       CDATA    #IMPLIED
>


<!-- Calls to subactions. -->

<!ELEMENT sub EMPTY >
<!ATTLIST sub
   ref           IDREF    #REQUIRED
>



<!-- Ancillary data -->
<!-- Timezone information -->
<!ELEMENT timezone ( standard,daylight? ) >
<!ATTLIST timezone
   name          CDATA    #REQUIRED
>

<!ENTITY % ZoneParams
’  offset        CDATA  #REQUIRED
   abbr          CDATA  #REQUIRED
   year          CDATA  #IMPLIED
   month         CDATA  #IMPLIED
   date          CDATA  #IMPLIED
   day           CDATA  #IMPLIED
   timeofday     CDATA  #IMPLIED' >

<!ELEMENT standard EMPTY>
<!ATTLIST standard
  %ZoneParams;
>

<!ELEMENT daylight EMPTY>
<!ATTLIST daylight
  %ZoneParams;
>


<!-- Top-level action nodes -->
<!ELEMENT subaction ( %Node; )>
<!ATTLIST subaction
   id            ID       #REQUIRED
>

<!ELEMENT outgoing ( %Node; )>

<!ELEMENT incoming ( %Node; )>


<!-- The top-level element of the script. -->

<!ELEMENT cpl  ( timezone*,subaction*,outgoing?,incoming? ) >
```

B TODO

See also the TODO notes in in motivation comments scattered throughout the document.

- Investigate XML Schemas as an alternative to DTDs: they may be more flexible and/or powerful.
- Determine proper system and public identifiers for the DTD.
- Register `application/cpl` as a MIME media type.

C Changes from earlier versions

C.1 Changes from draft -00

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

- Added high-level structure; script doesn't just start at a first action.
- Added a section giving a high-level explanation of the location model.
- Added informal syntax specifications for each tag so people don't have to try to understand a DTD to figure out the syntax.
- Added subactions, replacing the old link tags. Links were far too reminiscent of gotos for everyone's taste.
- Added ancillary information section, and timezone support.
- Added not-present switch output.
- Added address switches.
- Made case-insensitive string matching locale-independent.
- Added priority switch.
- Deleted "Other switches" section. None seem to be needed.
- Unified url and source parameters of lookup.
- Added caller prefs to lookup.
- Added location filtering.
- Eliminated "clear" parameter of location setting. Instead, proxy "eats" locations it has used.
- Added recurse and ordering parameters to proxy.
- Added default value of timeout for proxy.
- Renamed response to reject.
- Changed notify to mail, and simplified it.
- Simplified log, eliminating its failure output.
- Added description of default actions at various times during script processing.
- Updated examples for these changes.
- Updated DTD to reflect new syntax.

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

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

[3] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, "Extensible markup language (XML) 1.0," W3C Recommendation REC-xml-19980210, World Wide Web Consortium (W3C), Feb. 1998. Available at http://www.w3.org/TR/REC-xml.

[4] J. Lennox and H. Schulzrinne, "Call processing language framework and requirements," Internet Draft, Internet Engineering Task Force, July 1999. Work in progress.

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

[6] D. Raggett, A. L. Hors, and I. Jacobs, "HTML 4.0 specification," W3C Recommendation REC-html40-19980424, World Wide Web Consortium (W3C), Apr. 1998. Available at http://www.w3.org/TR/REC-html40/.

[7] ISO (International Organization for Standardization), "Information processing — text and office systems — standard generalized markup language (SGML)," ISO Standard ISO 8879:1986(E), International Organization for Standardization, Geneva, Switzerland, Oct. 1986.

[8] M. Davis and M. Dürst, "Unicode normalization forms," Unicode Technical Report 15, Unicode Consortium, Nov. 1999. Revision 18.0. Available at http://www.unicode.org/unicode/reports/tr15/.

[9] M. Davis, “Case mapping,” Unicode Technical Report 21, Unicode Consortium, Nov. 1999. Revision 3.0. Available at http://www.unicode.org/unicode/reports/tr21/.

[10] D. C. Fallside, “XML schema part 0: Primer,” Working Draft WD-xmlschema-0-20000225, World Wide Web Consortium (W3C), Feb. 2000. Available at http://www.w3.org/TR/xmlschema-0/.

[11] H. Schulzrinne and J. Rosenberg, “SIP caller preferences and callee capabilities,” Internet Draft, Internet Engineering Task Force, Mar. 2000. Work in progress.

[12] S. DeRose, E. Maler, D. Orchard, and B. Trafford, “XML linking language (XLink),” Working Draft WD-xlink-20000221, World Wide Web Consortium (W3C), Feb. 2000. Available at http://www.w3.org/TR/xlink/.

[13] T. Showalter, “Sieve: A mail filtering language,” Internet Draft, Internet Engineering Task Force, Mar. 1999. Work in progress.

Full Copyright Statement