Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-iptel-cpl-00.ps

IPTEL WG
Lennox/Schulzrinne
Columbia University
February 26, 1999
Expires: September 1999

CPL: A Language for User Control of Internet Telephony Services

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To view the list Internet-Draft Shadow Directories, see http://www.ietf.org/shadow.html.

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 a loop or a function.

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.

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 [4] 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 Abstract structure

Abstractly, a CPL script 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 script, see figure 1. Nodes and outputs can be thought of informally as boxes and arrows; the CPL is designed so that it can be conveniently edited graphically using this representation. Nodes are arranged in a directed acyclic graph, starting at a single root node; outputs of nodes are connected to additional nodes. When a CPL script is run, the action or condition described by the root 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.

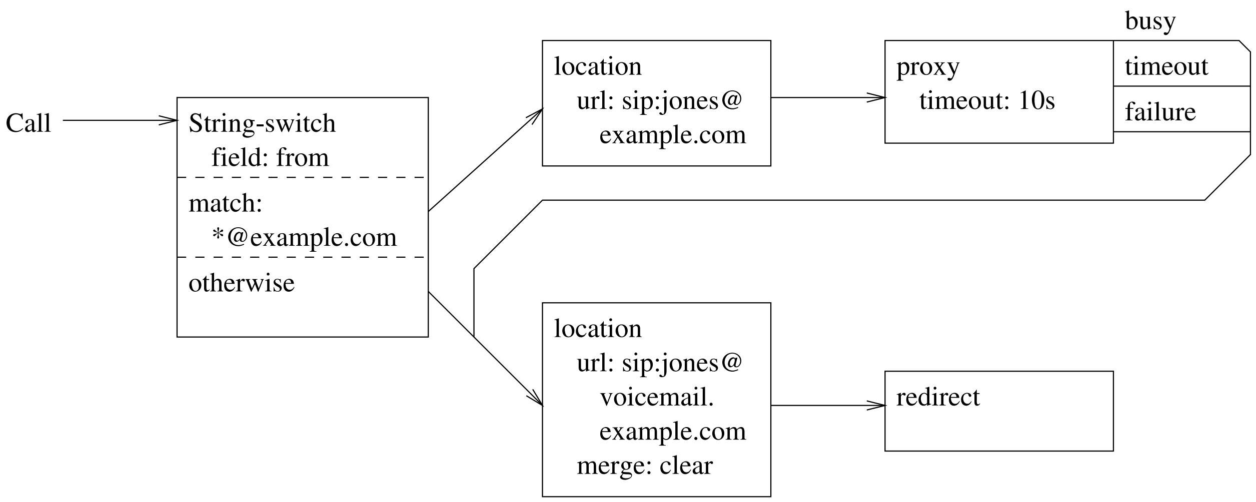

Figure 1: Sample CPL Script: Graphical Version

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

<call>
  <string-switch field="from">
    <string matches="*@example.com">
      <location url="sip:jones@example.com">
        <proxy>
  <busy> <link ref="voicemail" /> </busy>
  <noanswer> <link ref="voicemail" /> </noanswer>
  <failure> <link ref="voicemail" /> </failure>
</proxy>
      </location>
    </string>
    <otherwise>
      <location url="sip:jones@voicemail.example.com"
        merge="clear" id="voicemail">
        <redirect />
      </location>
    </otherwise>
  </string-switch>
</call>
```

Figure 2: Sample CPL Script: XML Version

2.2 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 [5], as both languages are simplifications of the earlier and larger standard SGML [6].

See figure 2 for the XML document corresponding to the graphical representation of a 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 4.1).

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 links, discussed further in section 7. The top-level node is enclosed in the special tag call; this is therefore the outermost tag of the XML.

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; for its syntax, please see the appendix.

3 Switches

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

All switches are arranged as a list of conditions that can match a variable, each with one output pointing to the next node to execute if that condition is matched. The conditions specified are tried in the order they are presented in the script; the output corresponding to the first node to match is taken. Switches also have an optional otherwise output, following all the other outputs, that matches if no previous node matched. If a switch does not have an otherwise output, and no condition matched, the server should take a default action, just as for any other un-attached node output, as discussed in section 2.1.

The variable to match is specified in the initial switch tag, as a field parameter. What variables are legal depends on which switch type is specified; some variables are optional, and CPL servers MAY define additional variables for each switch type. Because some variables may not be supported by a server, CPL servers SHOULD verify, at the time a script is submitted, that they support all the variables specified in the script.

3.1 String Switch

String-switch is a condition which allows string matching on a string variable. The node tag is named string-switch, and takes one argument, field, as discussed above. The output tags are named string, and take one mandatory and one optional argument. The mandatory argument's name is one of is, contains, or matches, indicating exact string match, substring match, or glob match of the variable respectively.

The optional argument of string output tags is comparator, which allows for internationalization of string matching. Strings to be matched are always considered as strings of UTF-8 characters. CPL servers MUST support the two comparators i;octet, indicating literal comparison of UTF-8 octets, and i;ascii-casemap, which indicates that alphabetic characters in the US-ASCII range should have their upper and lower cases compared the same. If no comparitor is specified, i;ascii-casemap is assumed. Comparators are defined by ACAP [7]; for more information, see that specification. CPL servers SHOULD verify at script submission time that all requested comparators are supported by the server.

> The naming scheme of comparators is as defined by ACAP; the motivation of the "i;" prefix of the comparators is unclear, but it seems to be some sort of namespace for future use.

> Question: should comparator be an attribute of the whole string-switch as opposed to an attribute of each comparison? There are arguments for either behavior.

All CPL servers MUST define the fields to and from for string matching, containing URIs referring to the called and calling addresses, respectively. CPL servers which run on SIP SHOULD also define request-uri, subject, organization, priority, containing the contents of the equivalent SIP headers, if present, and also display-to, and display-from, containing the display names corresponding to the called and calling addresses. CPL servers which run on H.323 SHOULD define XXX.

> Question: what are the appropriate string fields for H.323?

In this example, action1 is performed if the URL representation of the caller's address exactly matches "sip:lennox@cs.columbia.edu," action2 is performed for any string which matches any user at any host in the cs.columbia.edu domain, and action3 is taken in all other cases.

```
<string-switch field="from">
```

```
  <string is="sip:lennox@cs.columbia.edu">
     -action1-
  </string>
  <string matches="*@*cs.columbia.edu">
     -action2-
  </string>
  <otherwise>
     -action3-
  </otherwise>
</string-match>
```

3.2 Time Switch

Time-switch is a condition which allows matching on the time and/or date the triggering call was placed. Times are matched in the server's time zone. The node tag is named time-switch, and takes no arguments; the output tags are named time.

> Note: while it would be nice to allow clients to specify their own time zone, there doesn't currently appear to be any standard registry of time zone names, and we don't want to have to define one just for the CPL. Leveraging off of the iCalendar standard [8] would be nice, but their time zone specification seems excessively heavyweight — it defines time zone rules explicitly (and very verbosely) in its own syntax. Just specifying time zones as UTC offsets would be possible, but this doesn't cover daylight-savings time rules. Thus, we currently ignore the problem.

The time outputs can take the following optional arguments: 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 the ranges in all of the specified arguments.

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 links (see section 7).

3.3 Other Switches

Question: how should we switch based on media? We need a syntax for this. We could just switch on media type as a MIME type; the problem is that you may have several media types defined. Other important attributes of media include required bandwidth (numeric) and source address (IPv4 address, usually, but IPv6 in the future) — do we need switch types for these?

4 Locations

A number of CPL actions (defined in section 5) need to have locations specified. An executing CPL always has some set of locations specified; CPLs use location nodes to add or clear locations from the set.

By default, location nodes add to the current set of locations. Alternately, they can re-initialize the set, clearing it before adding additional nodes. This is specified with the argument merge, which can take two possible values, merge and clear. Its default value is merge.

4.1 Basic Location

Basic location nodes (which have the tag name location) specify a location literally, as a URL. They take a single argument, url; the desired location is given as an argument. Only one location 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.

4.2 Location Lookup

Locations can also be looked up through external means, through the use of the lookup tag. The location to look up the result can be specified either as a url, or as another source. External URLs, are specified with the attribute url, and should refer to an external source which returns the `application/url` media type. Other sources are specified with the attribute source. The only source currently defined is registration, which specifies all the locations currently registered with the server, using SIP REGISTER or H.323 RAS messages. A lookup tag MUST specify exactly one of url or source.

Lookup also has an optional attribute, timeout, which specifies the time in seconds the script is willing to wait for the lookup to be performed. 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 given.

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 Signalling Actions

Signalling action nodes cause signalling events in the underlying signalling protocol.

5.1 Proxy

Proxy causes the triggering call to be forwarded on to the currently specified set of locations. The server chooses the "best" response to the call attempt, as defined by the protocol or its configuration rules. If the call attempt was successful, CPL execution terminates; 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 as well.

Question: What other outputs are needed? Redirect? More varieties of failure?

If no locations are specified at the time the proxy command is executed, the server SHOULD attempt to proxy the call to its standard set of addresses for the user, or inform the caller that the caller is unavailable.

Proxy has one argument: timeout, which specifies the time, in seconds, to wait for the call to be completed or rejected, after which time the call attempt is terminated and the noanswer branch is taken.

Question: Do we want to be able to specify timeouts in other units, notably "number of rings"?

5.2 Redirect

Redirect causes the server to direct the calling party to attempt to place its call to the currently specified set of locations. This immediately terminates execution of the CPL script, so this node has no outputs. This node also has no arguments other than the standard Link ID target (see section 7).

5.3 Response

Response causes the server to reject the call attempt. This immediately terminates execution of the CPL script, so this node has no outputs.

This node has two arguments in addition to the standard Link ID (see section 7): 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 here 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.

The CPL does not define any way to send intermediate responses to call attempts. Servers SHOULD send them automatically, as appropriate.

Note: we need more named statuses.

Question: Success and redirection are also responses. Should this node be called "failure" or "reject" instead?

6 Other Actions

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

6.1 Notify

The Notify node causes the server to notify a user of the status of the CPL script through some non-telephony means; for instance, sending electronic mail, or delivering an instant message. It takes two arguments: a required URL indicating the means and address to contact (attribute url), and optionally a comment to be included in that message (attribute comment). 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. Servers SHOULD check the specified address at script submission time to ensure that they understand the specified URL scheme.

This node has two outputs, success and failure. The success branch is mandatory; if no failure branch is specified, the success branch is taken. The outputs SHOULD be specified in the order given.

> Question: is this too general? Notification is a very broad concept. Would simply having a "Mailto" tag be cleaner?

6.2 Log

The Log node causes the server to log information about the call to non-volatile storage. It 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. This specification does not define how users may retrieve their logs from the server.

This node has two outputs, success and failure. The success branch is mandatory; if no failure branch is specified, the success branch is taken. The outputs SHOULD be specified in the order given.

7 Links

XML syntax defines a tree. Because the general structure of the CPL is instead intended to be a directed acyclic graph, we provide the link structure to allow several node outputs to connect to a single node.

Every XML tag which represents a node has an optional argument id, which can be any XML id. (The id attribute is a standard XML attribute, defined in section 3.3.1 of the XML specification.) Any output which normally contains a node can, instead, contain a link tag with a ref attribute specifying the ID of some other node.

Every link ref MUST refer to a link ID specified in 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 [9].

When the CPL server initially processes the script, it MUST verify that no link refers to a node that is its parent in the tree; i.e., it MUST verify that the directed graph created by the tree and the links is acyclic. If it is not, the server SHOULD treat this error in the same manner as any other syntax error in a script. (This verification is algorithmically simply a matter of verifying that a depth-first search of the directed graph contains no back edges; see, for instance, [10], Lemma 23.10. It can typically be done simultaneously with the resolution of links.)

> If cycles were allowed in the graph, it would introduce the possibility of non-terminating CPL scripts, a possibility our requirements specifically excluded.

CPL servers MAY use link IDs to identify nodes for other purposes, for instance to report errors or to provide real-time debugging or flow information. Thus, scripts SHOULD provide IDs for every node for which they are interested in such information, even if no link connects to that node.

8 Examples

8.1 Example: Call Redirect Unconditional

The script in figure 3 is a simple script which redirects all calls to a single fixed location.

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

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

Figure 3: Example Script: Call Redirect Unconditional

8.2 Example: Call Forward Busy/No Answer

The script in figure 4 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 can point to the same node, through the use of the link tag.

8.3 Example: Call Screening

The script in figure 5 illustrates string switches and call rejection, in the form of a call screening script. Note also that because the string-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.

8.4 Example: Time-of-day Routing

Figure 6 illustrates time-based conditions.

8.5 Example: Non-call Actions

Figure 7 illustrates non-call actions; in particular, alerting a user by electronic mail if the lookup server failed. The primary reason for the Notify node is to allow this sort of out-of-band notification of error conditions, as the user might otherwise be unaware of any problem.

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

<call>
  <location url="sip:jones@jonespc.example.com">
     <proxy timeout="8s">
       <busy>
         <location url="sip:jones@voicemail.example.com" merge="clear"
                   id="voicemail" >
            <proxy />
         </location>
       </busy>
       <noanswer>
         <link ref="voicemail" />
       </noanswer>
     </proxy>
  </location>
</call>
```

Figure 4: Example Script: Call Forward Busy/No Answer

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

<call>
  <string-switch field="from">
    <string matches="anonymous@*">
       <response status="reject"
         reason="I don't accept anonymous calls" />
    </string>
  </string-switch>
</call>
```

Figure 5: Example Script: Call Screening

8.6 Example: A Complex Example

Finally, figure 8 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 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 call SYSTEM "cpl.dtd">

<call>
  <time-switch>
    <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>
</call>
```

Figure 6: Example Script: Time-of-day Routing

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

<call>
  <lookup url="http://www.example.com/cgi-bin/locate.cgi?user=jones"
          timeout="8s">
    <success>
      <proxy />
    </success>
    <failure>
      <notify url="mailto:jones@example.com"
              comment="The lookup server failed">
        <success>
          <response status="error" />
        </success>
      </notify>
    </failure>
  </lookup>
</call>
```

Figure 7: Example Script: Non-call Actions

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

<call>
  <location url="sip:jones@phone.example.com">
    <proxy timeout="8s">
      <busy>
<location url="sip:jones@voicemail.example.com" merge="clear">
          <redirect />
        </location>
      </busy>
      <noanswer>
        <string-switch field="from">
          <string matches="boss@*example.com">
            <location url="phone:+19175551212" merge="clear">
              <proxy />
            </location>
          </string>
          <otherwise>
    <location url="sip:jones@voicemail.example.com" merge="clear">
              <redirect />
       </location>
          </otherwise>
        </string-switch>
      </noanswer>
    </proxy>
  </location>
</call>
```

Figure 8: Example Script: A Complex Example

9 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.

10 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 [11], 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" ?>

<!--
    Initial draft DTD for CPL, corresponding to
    draft-ietf-iptel-cpl-00.
-->

<!-- Define types of nodes -->
<!-- Switch nodes -->
<!ENTITY % Switch 'string-switch|time-switch' >

<!-- Location nodes -->
<!ENTITY % Location 'location|lookup' >

<!-- Signalling action nodes -->
<!ENTITY % SignallingAction 'proxy|redirect|response' >

<!-- Other actions -->
<!ENTITY % OtherAction 'notify|log' >
```

```
<!-- Nodes are one of the above four categories, or a link.
     This entity (macro) describes the contents of an output. -->
<!ENTITY % Node     '%Location;|%Switch;|%SignallingAction;|
                     %OtherAction;|link' >

<!-- Nodes can have link IDs.  Since this is an attribute of every
     node, we need to define it early. -->
<!ENTITY % Link-ID 'id  ID    #IMPLIED'>


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

<!-- All switches contain an 'otherwise' node. -->

<!ELEMENT otherwise ( %Node; ) >

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

<!ELEMENT string-switch ( string+, otherwise? ) >
<!ATTLIST string-switch
   field  CDATA    #REQUIRED
   %Link-ID;
>

<!ELEMENT string ( %Node; ) >
<!ATTLIST string
   is         CDATA    #IMPLIED
   contains   CDATA    #IMPLIED
   matches    CDATA    #IMPLIED
   comparator CDATA    "i;ascii-casemap"
>

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

<!ELEMENT time-switch ( time+, otherwise? ) >
<!ATTLIST time-switch
   %Link-ID;
>

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

```
   day         CDATA  #IMPLIED
   timeofday   CDATA  #IMPLIED
>

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

<!ENTITY % Merge  'merge (merge|clear) "merge"' >

<!ELEMENT location ( %Node; ) >
<!ATTLIST location
   url CDATA    #REQUIRED
   %Merge;
   %Link-ID;
>

<!-- Sources of location lookups that aren't URIs. -->
<!ENTITY % Sources '(registration)' >

<!ELEMENT lookup ( success,notfound?,failure? ) >
<!ATTLIST lookup
  url     CDATA     #IMPLIED
  source  %Sources; #IMPLIED
  timeout CDATA     #IMPLIED
  %Merge;
  %Link-ID;
>

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

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

<!ELEMENT proxy ( busy?,noanswer?,failure? ) >
<!ATTLIST proxy
   timeout CDATA   #IMPLIED
   %Link-ID;
>

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

```
<!ELEMENT redirect EMPTY >
<!ATTLIST redirect
   %Link-ID;
>

<!-- Statuses we can return -->

<!ELEMENT response EMPTY >
<!ATTLIST response
   status CDATA    #REQUIRED
   reason CDATA    #IMPLIED
   %Link-ID;
>

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

<!ELEMENT notify ( success,failure? ) >
<!ATTLIST notify
   url     CDATA    #REQUIRED
   comment CDATA    #IMPLIED
   %Link-ID;
>

<!ELEMENT log ( success,failure? ) >
<!ATTLIST log
   name    CDATA    #IMPLIED
   comment CDATA    #IMPLIED
   %Link-ID;
>


<!-- Links to other nodes. -->

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


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

<!ELEMENT call  ( %Node; ) >
```

B 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," Internet Draft, Internet Engineering Task Force, Jan. 1999. Work in progress.

[2] International Telecommunication Union, "Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service," Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, May 1996.

[3] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, "Extensible markup language (XML) 1.0," W3C Recommendation 10-February-1998, World Wide Web Consortium (W3C), Feb. 1998. http://www.w3.org/TR/REC-xml.

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

[5] D. Raggett, A. L. Hors, and I. Jacobs, "HTML 4.0 specification," W3C Recommendation revised on 24-Apr-1998, World Wide Web Consortium (W3C), Apr. 1998. http://www.w3.org/TR/REC-html40/.

[6] 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, Oct. 1986.

[7] J. Myers and C. Newman, "ACAP – application configuration access protocol," Request for Comments (Proposed Standard) 2244, Internet Engineering Task Force, Dec. 1997.

[8] F. Dawson and D. Stenerson, "Internet calendaring and scheduling core object specification (icalendar)," Request for Comments (Proposed Standard) 2445, Internet Engineering Task Force, Nov. 1998.

[9] E. Maler and S. DeRose, "XML linking language (XLink)," Working Draft 3-March-1998, World Wide Web Consortium (W3C), Mar. 1998. http://www.w3.org/TR/WD-xlink.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. New York: McGraw-Hill, 1990.

[11] T. Showalter, "Sieve: A mail filtering language," Internet Draft, Internet Engineering Task Force, Jan. 1999. Work in progress.

Full Copyright Statement