Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-iptel-cpl-04.ps

IPTEL WG
Lennox/Schulzrinne
Columbia University

November 14, 2000
Expires: May, 2001

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. This document primarily addresses the usage in servers. 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 RFC 2824, "Call Processing Language Framework and Requirements" [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.

> 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 operations and decisions a telephony signalling server performs on a call set-up event. There are two types of call processing actions: *top-level actions* and *subactions.* 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. Subactions are actions which can be called from other actions. The CPL forbids subactions from being called recursively: see Section 9.

Ancillary information is information which is necessary for a server to correctly process a script, but which does not directly describe any operations or decisions. Currently, no ancillary information is defined, but the section is reserved for use by extensions.

2.2 Abstract Structure of a Call Processing Action

Abstractly, a call processing action is described by a collection of nodes, which describe operations that can be performed or decisions 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 decision 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 decision described by the action's top-level node is performed; based on the result of that node, the server follows one of the node's outputs, and the subsequent node it points to 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 does not point to another node, it indicates that the CPL server should perform a node- or protocol-specific action. Some nodes have specific default behavior associated with them; for others, the default behavior is implicit in the underlying signalling protocol, or can be configured by the administrator of the server. For further details on this, see Section 11.

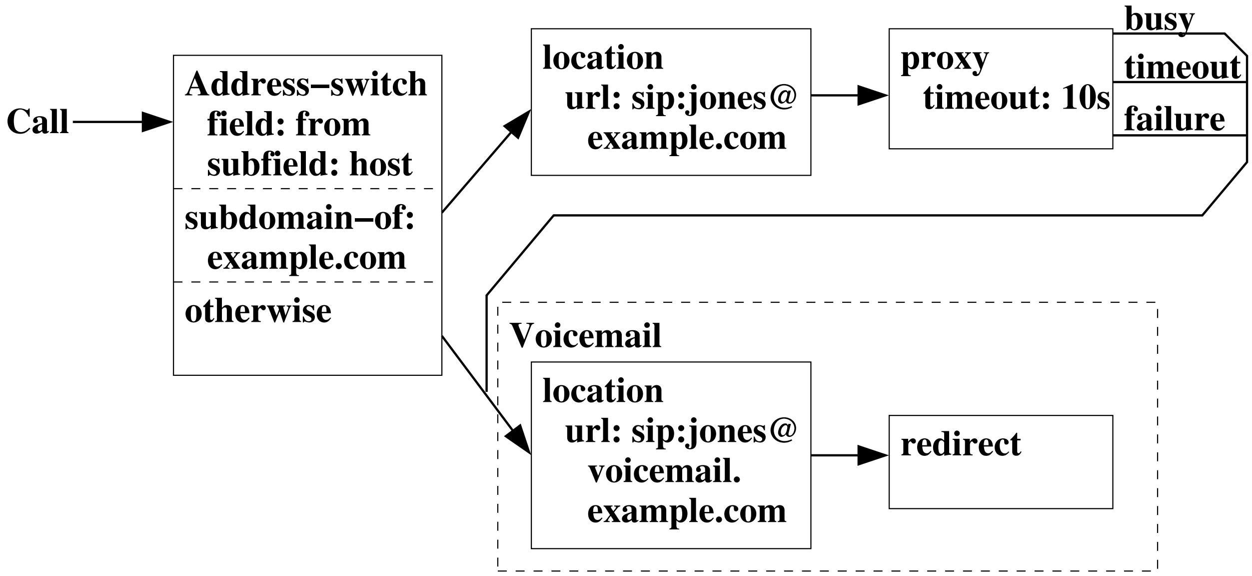

Figure 1: Sample CPL Action: Graphical Version

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 subactions). This allows locations to be retrieved from external sources, filtered, and so forth, without requiring general language support for such operations (which could harm the simplicity and tractability of understanding the language). The specific operations which add, retrieve, or filter location sets are given in Section 6.

For the incoming top-level call 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

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "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 timeout="10">
            <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

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 a few exceptions: see Sections 6.1, 6.3, 8.1, and 8.2).

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 subactions, discussed further in Section 9.

The higher-level structure of a CPL script is represented by tags corresponding to each piece of ancillary information, subactions, 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 C. 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 Document Information

This section gives information describing how CPL scripts are identified.

3.1 CPL Document Identifiers for XML

A CPL script list which appears as a top-level XML document is identified with the formal public identifier "-//IETF//DTD RFCxxxx CPL 1.0//EN".

A CPL embedded as a fragment within another XML document is identified with the XML namespace identifier "http://www.rfc-editor.org/rfc/rfcxxxx.txt".

> [Note to RFC editor: please replace "xxxx" above with the number of this RFC.]
>
> Note that the URIs specifying XML namespaces are only globally unique names; they do not have to reference any particular actual object. The URI of a canonical source of this specification meets the requirement of being globally unique, and is also useful to document the format.

3.2 MIME Registration

As an XML type, CPL's MIME registration conforms with "XML Media Types," RFC YYYY [8].

> [Note to RFC Editor: please replace "YYYY" in this section, and in bibliography entry [8], with the RFC number assigned to the Internet-Draft `draft-murata-xml-09.txt`, approved for Proposed Standard.]

MIME media type name: application

MIME subtype name: cpl+xml

Mandatory parameters: none

Optional parameters: charset
As for `application/xml` in RFC YYYY.

Encoding considerations: As for `application/xml` in RFC YYYY.

Security considerations: See Section 14, and Section 10 of RFC YYYY.

Interoperability considerations: Different CPL servers may use incompatible address types. However, all potential interoperability issues should be resolvable at the time a script is uploaded; there should be no interoperability issues which cannot be detected until runtime.

Published specification: This document.

Applications which use this media type: None publicly released at this time, as far as the authors are aware.

Additional information: **Magic number:** None

File extension: `.cpl` or `.xml`

Macintosh file type code: "TEXT"

Person and e-mail address for further information:
Jonathan Lennox <lennox@cs.columbia.edu>
Henning Schulzrinne <hgs@cs.columbia.edu>

Intended usage: COMMON

Author/Change Controller: The IETF.

4 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.

Tag:	cpl	
Parameters:	None	
Sub-tags:	ancillary	See Section 10
	subaction	See Section 9
	outgoing	Top-level actions to take on this user's outgoing calls
	incoming	Top-level actions to take on this user's incoming calls

Figure 3: Syntax of the top-level cpl tag

Call processing actions, both top-level actions and sub-actions, consist of a tree of nodes and outputs. Nodes and outputs are both described by XML tags. There are four categories of CPL nodes: *switches*, which represent choices a CPL script can make; *location modifiers*, which add or remove locations from the location set; *signalling operations*, which cause signalling events in the underlying protocol; and *non-signalling operations,* which trigger behavior which does not effect the underlying protocol.

5 Switches

Switches represent choices a 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 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, which MAY occur anywhere in the list of outputs, is true if the variable the switch was to match was not present in the original call setup request. (In this document, this is sometimes described by saying that the information is "absent".) The output otherwise, which MUST be the last output specified if it is present, matches if no other condition matched.

If no condition matches and no otherwise output was present in the script, the default script behavior is taken. See Section 11 for more information on this.

5.1 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, or display (also: password and alias-type)
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 field 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 for protocol-specific values. (The subfield password is defined for SIP in Section 5.1.1; the subfield alias-type is defined for H.323 in Appendix B.1.) If no subfield is specified, the "entire" address is matched; the precise meaning of this is defined for each underlying signalling protocol. Servers MAY define additional subfield values.

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 a URI. The types specifically discussed by this document are sip, tel, and h323. The address type is not case-sensitive. It has a value for all defined address types.

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 absent.

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 absent.

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 absent; 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 absent. 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 5.2. The contains operator may be applied to it. It may be absent.

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, `subdomain-of="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; `subdomain-of="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.

5.1.1 Usage of address-switch with 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, with visual separators stripped, 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, and is case-sensitive. 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, visual separators are stripped. The host and port subfields are both not present.

For h323 URLs, subfields MAY be set according to the scheme described in Appendix B.

For other URI schemes, only the address-type subfield is defined by this specification; servers MAY set other 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. For "sip" URLs, all parameters are stripped; for other URLs, the URL is used verbatim.

5.2 String Switches

String switches allow a CPL script to make decisions based on free-form 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, user-agent, language, or display
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.

String switches are dependent on the call signalling protocol being used.

Five fields are defined, listed below. The value of each of these fields, except as specified, is a free-form Unicode string with no other structure defined.

subject The subject of the call.

organization The organization of the originator of the call.

user-agent The name of the program or device with which the call request was made.

language The languages in which the originator of the call wishes to receive responses. This contains a list of RFC 1766 [9] language tags, separated by commas.

> Note that matching based on contains is likely to be much more useful than matching based on is, for this field.

display Free-form text associated with the call, intended to be displayed to the recipient, with no other semantics defined by the signalling protocol.

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 [10]. Then, strings are compared using locale-insensitive caseless mapping, as specified in Unicode Technical Report 21 [11].

Code to perform the first step, in Java and Perl, is available; see the links from Annex E of UTR 15 [10]. 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.

5.2.1 Usage of string-switch with SIP

For SIP, the fields subject, organization, and user-agent correspond to the SIP header fields with the same name. These are used verbatim as they appear in the message.

The field language corresponds to the SIP Accept-Language header. It is converted to a list of comma-separated languages as described above.

The field display is not used, and is never present.

5.3 Time Switches

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

Time switches are independent of the underlying signalling protocol.

Node:	time-switch	
Outputs:	time	Specific time to match
Parameters:	tzid	RFC 2445 Time Zone Identifier
	tzurl	RFC 2445 Time Zone URL
Output:	time	
Parameters:	dtstart	Start of interval (RFC 2445 DATE-TIME)
	dtend	End of interval (RFC 2445 DATE-TIME)
	duration	Length of interval (RFC 2445 DURATION)
	freq	Frequency of recurrence (one of “daily”, “weekly”, “monthly”, or “yearly”)
	interval	How often the recurrence repeats
	until	Bound of recurrence (RFC 2445 DATE-TIME)
	byday	List of days of the week
	bymonthday	List of days of the month
	byyearday	List of days of the year
	byweekno	List of weeks of the year
	bymonth	List of months of the year
	wkst	First day of workweek

Figure 6: Syntax of the time-switch node

Time switches are based on a large subset of how recurring intervals of time are specified in the Internet Calendaring and Scheduling Core Object Specification (iCal COS), RFC 2445 [12].

This allows CPLs to be generated automatically from calendar books. It also allows us to re-use the extensive existing work specifying time intervals.

> The subset was designed with the goal that a time-switch can be evaluated — an instant can be determined to fall within an interval, or not — in constant ($O(1)$) time.

An algorithm to whether an instant falls within a given recurrence is given in Appendix A.

The time-switch tag takes two optional parameters, tzid and tzurl, both of which are defined in RFC 2445 (Sections 4.8.3.1 and 4.8.3.5 respectively). The TZID is the identifying label by which a time zone definition is referenced. If it begins with a forward slash (solidus), it references a to-be-defined global time zone registry; otherwise it is locally-defined at the server. The TZURL gives a network location from which an up-to-date VTIMEZONE definition for the timezone can be retrieved.

While TZID labels that do not begin with a forward slash are locally defined, it is RECOMMENDED that servers support at least the naming scheme used by Olson Time Zone database [13]. Examples of timezone databases that use the Olson scheme are the `zoneinfo` files on most Unix-like systems, and the standard Java `TimeZone` class.

If a script is uploaded with a tzid and tzurl which the CPL server does not recognize or cannot resolve, it SHOULD diagnose and reject this at script upload time. If neither tzid nor tzurl are present, all non-UTC times within this time switch should be interpreted as being "floating" times, i.e. that they are specified in the local timezone of the CPL server.

> Because of daylight-savings-time changes over the course of a year, it is necessary to specify time switches in a given timezone. UTC offsets are not sufficient, or a time-of-day routing rule which held between 9 am and 5 pm in the eastern United States would start holding between 8 am and 4 pm at the end of October.

Authors of CPL servers should be careful to handle correctly the intervals when local time is discontinuous, at the beginning or end of daylight-savings time. Note especially that some times may occur more than once when clocks are set back. The algorithm in Appendix A is believed to handle this correctly.

Time nodes specify a list of periods during which their output should be taken. They have two required parameters: dtstart, which specifies the beginning of the first period of the list, and exactly one of dtend or duration, which specify the ending time or the duration of the period, respectively. The dtstart and dtend parameters are formatted as iCal COS DATE-TIME values, as specified in Section 4.3.5 of RFC 2445 [12]. Because time zones are specified in the top-level time-switch tag, only forms 1 or 2 (floating or UTC times) can be used. The duration parameter is given as an iCal COS DURATION parameter, as specified in section 4.3.6 of RFC 2445. Both the DATE-TIME and the DURATION syntaxes are subsets of the corresponding syntaxes from ISO 8601 [14].

For a recurring interval, the duration parameter MUST be less than twenty-four hours. For non-recurring intervals, durations of any length are permitted.

If no other parameters are specified, a time node indicates only a single period of time. More complicated sets periods intervals are constructed as recurrences. A recurrence is specified by including the freq parameter, which indicates the type of recurrence rule. No parameters other than dtstart, dtend, and duration SHOULD be specified unless freq is present.

The freq parameter takes one of the following values: daily, to specify repeating periods based on an interval of a day or more; weekly, to specify repeating periods based on an interval of a week or more; monthly, to specify repeating periods based on an interval of a month or more; and yearly, to specify repeating periods based on an interval of a year or more. These values are not case-sensitive.

> The values secondly, minutely, and hourly are present in iCal, but were removed from CPL.

The interval parameter contains a positive integer representing how often the recurrence rule repeats. The default value is "1", meaning every day for a daily rule, every week for a weekly rule, every month for a monthly rule and every year for a yearly rule.

The until parameter defines an iCal COS DATE or DATE-TIME value which bounds the recurrence rule in an inclusive manner. If the value specified by until is synchronized with the specified recurrence, this date or date-time becomes the last instance of the recurrence. If specified as a date-time value, then it MUST be specified in an UTC time format. If not present, the recurrence is considered to repeat forever.

> iCal also defines a count parameter, which allows an alternate method of specifying a bound to a recurrence. This bound has been removed from CPL. Translating from full iCal recurrences to CPL recurrences requires that the count parameter be converted to an until parameter, which can be done by enumerating the recurrence and determining its final date.

The byday parameter specifies a comma-separated list of days of the week. MO indicates Monday; TU indicates Tuesday; WE indicates Wednesday; TH indicates Thursday; FR indicates Friday; SA indicates Saturday; SU indicates Sunday. These values are not case-sensitive.

Each byday value can also be preceded by a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of the specific day within the monthly or yearly recurrence. For example, within a monthly rule, +1MO (or simply 1MO) represents the first Monday within the month, whereas -1MO represents the last Monday of the month. If an integer modifier is not present, it means all days of this type within the specified frequency. For example, within a monthly rule, MO represents all Mondays within the month.

The bymonthday parameter specifies a comma-separated list of days of the month. Valid values are 1 to 31 or -31 to -1. For example, -10 represents the tenth to the last day of the month.

The byyearday parameter specifies a comma-separated list of days of the year. Valid values are 1 to 366 or -366 to -1. For example, -1 represents the last day of the year (December 31st) and -306 represents the 306th to the last day of the year (March 1st).

The byweekno parameter specifies a comma-separated list of ordinals specifying weeks of the year. Valid values are 1 to 53 or -53 to -1. This corresponds to weeks according to week numbering as defined in ISO 8601 [14]. A week is defined as a seven day period, starting on the day of the week defined to be the week start (see wkst). Week number one of the calendar year is the first week which contains at least four (4) days in that calendar year. This parameter is only valid for yearly rules. For example, 3 represents the third week of the year.

> Note: Assuming a Monday week start, week 53 can only occur when Thursday is January 1 or if it is a leap year and Wednesday is January 1.

The bymonth parameter specifies a comma-separated list of months of the year. Valid values are 1 to 12.

The wkst parameter specifies the day on which the workweek starts. Valid values are MO, TU, WE, TH, FR, SA and SU. This is significant when a weekly recurrence has an interval greater than 1, and a byday parameter is specified. This is also significant in a yearly recurrence when a byweekno parameter is specified. The default value is MO, following ISO 8601 [14].

> iCal also includes the Byxxx parameters bysecond, byminute, byhour, and bysetpos, which have been removed from CPL.

If byxxx parameter values are found which are beyond the available scope (ie, bymonthday="30" in February), they are simply ignored.

Byxxx parameters modify the recurrence in some manner. Byxxx rule parts for a period of time which is the same or greater than the frequency generally reduce or limit the number of occurrences of the recurrence generated. For example, freq="daily" bymonth="1" reduces the number of recurrence instances from all days (if the bymonth parameter is not present) to all days in January. Byxxx parameters for a period of time less than the frequency generally increase or expand the number of occurrences of the recurrence. For example, freq="yearly" bymonth="1,2" increases the number of days within the yearly recurrence set from 1 (if bymonth parameter is not present) to 2.

If multiple Byxxx parameters are specified, then after evaluating the specified freq and interval parameters, the Byxxx parameters are applied to the current set of evaluated occurrences in the following order: bymonth, byweekno, byyearday, bymonthday, and byday; then until is evaluated.

Here is an example of evaluating multiple Byxxx parameters.

```
<time dtstart="19970105T083000" duration="PT10M"
      freq="yearly" interval="2" bymonth="1" byday="SU">
```

First, the interval="2" would be applied to freq="YEARLY" to arrive at "every other year." Then, bymonth="1" would be applied to arrive at "every January, every other year." Then, byday="SU" would be applied to arrive at "every Sunday in January, every other year." Then the time of day is derived from dtstart to end up in "every Sunday in January from 8:30:00 AM to 8:40:00 AM, every other year." Similarly, if the byday, bymonthday or bymonth parameter were missing, the appropriate day or month would have been retrieved from the dtstart parameter.

The iCal COS RDATE, EXRULE and EXDATE recurrence rules are not specifically mapped to components of the time-switch node. Equivalent functionality to the exception rules can be attained by using the ordering of switch rules to exclude times using earlier rules; equivalent functionality to the additional-date RDATE rules can be attained by using sub nodes (see Section 9) to link multiple outputs to the same subsequent node.

The not-present output is never true for a time switch. However, it MAY be included, to allow switch processing to be more regular.

5.3.1 Motivations for the iCal Subset

(This sub-sub-section is non-normative.)

The syntax of the CPL time-switch was based on that of the iCal COS RRULE, but as mentioned above, certain features were omitted and restrictions were added. Specifically:

1. All recurrence intervals and rules describing periods less than a day were removed. These were the frequencies secondly, minutely, and hourly, and the Byxxx rules bysecond, byminute, and byhour.

2. The count and bysetpos parameters were removed.

3. Durations were constrained to less than 24 hours for recurring intervals.

These restrictions were added so that time switches could be resolved efficiently, in $O(1)$ time. This restriction means that it must be possible to resolve a time switch without having to enumerate all its recurrences from dtstart to the present interval. As far as we have been able to determine, it is not possible to test whether the count and bysetpos parameters are satisfied without performing such an enumeration.

Constant running time of time switches also requires that a candidate starting time for a recurrence can be established quickly and uniquely, to check whether it satisfies the other restrictions. This requires that a recurrence's duration not be longer than its repetition interval, so that a given instant cannot fall within several consecutive repetitions of the recurrence. We guaranteed this by eliminating durations longer than 24 hours, and repetitions shorter than that period. The one-day point seemed to be the most generally useful place to place this division, as some investigation showed that many common calendaring applications do not support durations longer than a day, none that we found supported repetitions shorter than a day. Eliminating sub-day repetitions also greatly simplifies the handling of daylight-savings transitions.

The algorithm given in Appendix A runs in constant time, and motivated the development of this iCal subset.

5.4 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. They are dependent on the underlying signalling protocol.

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 one of the following priorities: 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 call 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 specified in the call, it 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. However, it MAY be included, to allow switch processing to be more regular.

5.4.1 Usage of priority-switch with SIP

The priority of a SIP message corresponds to the Priority header in the initial INVITE message.

6 Location Modifiers

The abstract location model of the CPL is described in Section 2.3. The behavior of several of the signalling operations (defined in Section 7) is dependent on the current location set specified. Location nodes add 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.

6.1 Explicit Location

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

Explicit location nodes are dependent on the underlying signalling protocol.

Node:	location	
Outputs:	None (next node follows directly)	
Next node:	Any node	
Parameters:	url	URL of address to add to location set
	priority	Priority of this location ($0.0 - 1.0$)
	clear	Whether to clear the location set before adding the new value

Figure 8: Syntax of the location node

Explicit location nodes have three node parameters. The mandatory url parameter's 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.

The optional priority parameter specifies a priority for the location. Its value is a floating-point number between 0.0 and 1.0. If it is not specified, the server SHOULD assume a default priority of 1.0. The optional clear parameter specifies whether the location set should be cleared before adding the new location to it. Its value can be "yes" or "no", with "no" as the default.

Basic location nodes have only one possible result, 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, their XML representations do not have explicit output tags; the `<location>` tag directly contains another node.

6.1.1 Usage of location with SIP

All SIP locations are represented as URLs, so the locations specified in location tags are interpreted directly.

6.2 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 is dependent on the underlying signalling protocol.

Location lookup nodes have one mandatory parameter, and four optional parameters. The mandatory parameter is source, the source of the lookup. This can either be a URI, or a non-URI value. If the value of source is a URI, it indicates a location which the CPL server can query to obtain an object with the `text/uri-list` media type (see the IANA registration of this type, which also appears in RFC 2483 [15]). The query is performed verbatim, with no additional information (such as URI parameters) added. The server adds the locations contained in this object to the location set.

CPL servers MAY refuse to allow URI-based sources for location queries for some or all URI schemes. In this case, they SHOULD reject the script at script upload time.

> There has been discussion of having CPL servers add URI parameters to the location request, so that (for instance) CGI scripts could be used to resolve them. However, the consensus was that this should be a CPL extension, not a part of the base specification.

Node:	lookup	
Outputs:	success	Next node if lookup was successful
	notfound	Next node if lookup found no addresses
	failure	Next node 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
	clear	Whether to clear the location set before adding the new values
Output:	success	
Parameters:	none	
Output:	notfound	
Parameters:	none	
Output:	failure	
Parameters:	none	

Figure 9: Syntax of the lookup node

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.

The lookup node also has four optional parameters. The timeout parameter 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 clear parameter specifies whether the location set should be cleared before the new locations are added.

The other two optional parameters affect the interworking of the CPL script with caller preferences and caller capabilities. By default, a CPL server SHOULD invoke the appropriate caller preferences filtering of the underlying signalling protocol, if the corresponding information is available. The two parameters use and ignore allow the script to modify how the script applies caller preferences filtering. The specific meaning of the values of these parameters is signalling-protocol dependent; see Section 6.2.1 for SIP and Appendix B.5 for H.323.

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 a given output is not present, script execution terminates and the default behavior is performed.

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

6.2.1 Usage of lookup with SIP

Caller preferences for SIP are defined in "SIP Caller Preferences and Callee Capabilities" [16]. By default, a CPL server SHOULD honor any Accept-Contact and Reject-Contact headers of the original call request, as specified in that document. The two parameters use and ignore allow the script to modify the data input to the caller preferences algorithm. These 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.

If a SIP server does not support caller preferences and callee capabilities, if the call request does not contain any preferences, or if the callee's registrations do not contain any capabilities, the use and ignore parameters are ignored.

6.3 Location Removal

A CPL script can also remove locations from the location set, through the use of the remove-location node. The syntax of this node is defined in Figure 10.

The meaning of this node is dependent on the underlying signalling protocol.

Node:	remove-location	
Outputs:	None (next node follows directly)	
Next node:	Any node	
Parameters:	location	Location to remove
	param	Caller preference parameters to apply
	value	Value of caller preference parameters

Figure 10: Syntax of the remove-location node

A remove-location node removes locations from the location set. It is primarily useful following a lookup node. An example of this is given in Section 13.8.

The remove-location node has three optional parameters. The parameter location gives the URL (or a signalling-protocol-dependent URL pattern) of location or locations to be removed from the set. If this parameter is not given, all locations, subject to the constraints of the other parameters, are removed from the set.

If param and value are present, their values are comma-separated lists of caller preferences parameters and corresponding values, respectively. The nth entry in the param list matches the nth entry in the value list. There MUST be the same number of parameters as values specified. The meaning of these parameters is signalling-protocol dependent.

The remove-location node has no explicit output tags. In the XML syntax, the XML remove-location tag directly encloses the next node's tag.

6.3.1 Usage of remove-location with SIP

For SIP-based CPL servers, the remove-location node has the same effect on the location set as a Reject-Contact header in caller preferences [16]. The value of the location parameter is treated as though it were the addr-spec field of a Reject-Contact header; thus, an absent header is equivalent to an addr-spec of "*" in that specification. The param and value parameters are treated as though they appeared in the params field of a Reject-Location header, as "; param=value" for each one.

If the CPL server does not support caller preferences and callee capabilities, or if the callee did not supply any preferences, the param and value parameters are ignored.

7 Signalling Operations

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

7.1 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.

The specific signalling events invoked by the proxy node are signalling-protocol-dependent, though the general concept should apply to any signalling protocol.

After a proxy operation 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, CPL execution terminates and the server proceeds to its default behavior (normally, to allow the call to be set up). Otherwise, the next node corresponding to one of the proxy node's outputs is taken. The busy output is followed if the call was busy; noanswer is followed if the call was not answered before the timeout parameter expired; redirection is followed if the call was redirected; and failure is followed if the call setup failed for any other reason.

If one of the conditions above is true, but the corresponding output was not specified, the default output of the proxy node is followed instead. If there is also no default node specified, CPL execution terminates and the server returns to its default behavior (normally, to forward the best response upstream to the originator).

> Note: CPL extensions to allow in-call or end-of-call operations will require an additional output, such as success, to be added.

If no locations were present in the set, or if the only locations in the set were locations to which the server cannot proxy a call (for example, "http" URLs), the failure output is taken.

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 noanswer or default output specified; otherwise the server SHOULD allow the call to ring for a reasonably long period of time (to the maximum extent that server policy allows).

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. Note that if the value of recurse is yes, the redirection

Node:	proxy	
Outputs:	busy	Next node if call attempt returned “busy”
	noanswer	Next node if call attempt was not answered before timeout
	redirection	Next node if call attempt was redirected
	failure	Next node if call attempt failed
	default	Default next node for unspecified outputs
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:	redirection
Parameters:	none

Output:	failure
Parameters:	none

Output:	default
Parameters:	none

Figure 11: Syntax of the proxy node

output to the script is never taken. In this case this output SHOULD NOT be present. The default value of this parameter is yes.

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 one with the highest priority be tried first, the one with the next-highest priority second, and so forth, until one succeeds or the set is exhausted. First-only instructs the server to try only the highest-priority address in the set, and then follow one of the outputs. The priority of locations in a set is determined by server policy, though CPL servers SHOULD honor the priority parameter of the location tag. The default value of this parameter is parallel.

Once a proxy operation completes, if control is passed on to other nodes, all locations which have been used are cleared from the location set. That is, the location set is emptied of proxyable locations if the ordering was parallel or sequential; the highest-priority item in the set is removed from the set if ordering was first-only. (In all cases, non-proxyable locations such as “http” URIs remain.) In the case of a redirection output, the new addresses to which the call was redirected are then added to the location set.

7.1.1 Usage of proxy with SIP

For SIP, the best response to a proxy node is determined by the algorithm of the SIP specification. The node's outputs correspond to the following events:

busy A 486 or 600 response was the best response received to the call request.

redirection A 3xx response was the best response received to the call request.

failure Any other 4xx, 5xx, or 6xx response was the best response received to the call request.

no-answer No final response was received to the call request before the timeout expired.

SIP servers SHOULD honor the q parameter of SIP registrations and the output of the caller preferences lookup algorithm when determining location priority.

7.2 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.

The specific behavior the redirect node invokes is dependent on the underlying signalling protocol involved, though its semantics are generally applicable.

Node:	redirect	
Outputs:	None (no node may follow)	
Next node:	None	
Parameters:	permanent	Whether the redirection should be considered permanent

Figure 12: Syntax of the redirect node

Redirect immediately terminates execution of the CPL script, so this node has no outputs and no next node. It has one parameter, permanent, which specifies whether the result returned should indicate that this is a permanent redirection. The value of this parameter is either "yes" or "no" and its default value is "no."

7.2.1 Usage of redirect with SIP

The SIP server SHOULD send a 3xx class response to a call request upon executing a redirect tag. If permanent was yes, the server SHOULD send the response "301 Moved permanently"; otherwise it SHOULD send "302 Moved temporarily".

7.3 Reject

Reject nodes cause the server to reject the call attempt. Their syntax is given in Figure 13. The specific behavior they invoke is dependent on the underlying signalling protocol involved, though their semantics are generally applicable.

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

Node:	reject	
Outputs:	None (no node may follow)	
Next node:	None	
Parameters:	status	Status code to return
	reason	Reason phrase to return

Figure 13: Syntax of the reject node

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, or a signalling-protocol-defined status.

The reason argument optionally allows the script to specify a reason for the rejection.

7.3.1 Usage of reject with SIP

Servers which implement SIP SHOULD also allow the status field to be a numeric argument corresponding to a SIP status in the 4xx, 5xx, or 6xx range.

They SHOULD send the "reason" parameter in the SIP reason phrase.

A suggested mapping of the named statuses is as follows. Servers MAY use a different mapping, though similar semantics SHOULD be preserved.

busy: 486 Busy Here

notfound: 404 Not Found

reject: 603 Decline

error: 500 Internal Server Error

8 Non-signalling Operations

In addition to the signalling operations, the CPL defines several operations which do not affect and are not dependent on the telephony signalling protocol.

8.1 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:	None (next node follows directly)	
Next node:	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 original call request and the CPL script at the time of the notification.

> 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" />`.

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

Note that the syntax of XML requires that ampersand characters, "&", which are used as parameter separators in mailto URLs, be quoted as "`&`" inside parameter values (see Section C.12 of [3]).

8.1.1 Suggested Content of Mailed Information

This section presents suggested guidelines for the mail sent as a result of the mail node, for requests triggered by SIP. The message mailed (triggered by any protocol) SHOULD contain all this information, but servers MAY elect to use a different format.

1. If the mailto URI did not specify a subject header, the subject of the e-mail is "[CPL]" followed by the subject header of the SIP request. If the URI specified a subject header, it is used instead.
2. The From field of the e-mail is set to a CPL server configured address, overriding any From field in the mailto URI.
3. Any Reply-To header in the URI is honored. If none is given, then an e-mail-ized version of the origin field of the request is used, if possible (e.g., a SIP From header with a sip: URI would be converted to an e-mail address by stripping the URI scheme).
4. If the mailto URI specifies a body, it is used. If none was specified, the body SHOULD contain at least the identity of the caller (both the caller's display name and address), the date and time of day, the call subject, and if available, the call priority.

The server SHOULD honor the user's requested languages, and send the mail notification using an appropriate language and character set.

8.2 Log

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

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 logged the event. If the name parameter is not given, the event is logged to a standard, server-defined log file for the script owner. This specification does not define how users may retrieve their logs from the server.

The name of a log is a logical name only, and does not necessarily correspond to any physical file on the server. The interpretation of the log file name is server defined, as is a mechanism to access these logs. The

Node:	log	
Outputs:	None (next node follows directly)	
Next node:	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

CPL server SHOULD NOT directly map log names uninterpreted onto local file names, for security reasons, lest a security-critical file be overwritten.

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

9 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.

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

Subactions are defined through subaction tags. These tags are placed in the CPL after any ancillary information (see Section 10) 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.

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 subaction 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.

Subaction IDs are case sensitive.

If any subsequent version or extension defines external linkages, it should probably use a different tag, perhaps XLink [17]. Ensuring termination in the presence of external links is a difficult problem.

10 Ancillary Information

No ancillary information is defined in the base CPL specification. If ancillary information, not part of any operation, is found to be necessary for a CPL extension, it SHOULD be placed within this tag.

The (trivial) definition of the ancillary information tag is given in Figure 17.

It may be useful to include timezone definitions inside CPL scripts directly, rather than referencing them externally with tzid and tzurl parameters. If it is, an extension could be defined to include them here.

Tag:	ancillary
Parameters:	None
Subtags:	None

Figure 17: Syntax of the ancillary tag

11 Default Behavior

When a CPL node reaches an unspecified output, either because the output tag is not present, or because the tag is present but does not contain a node, the CPL server's behavior is dependent on the current state of script execution. This section gives the operations that should be taken in each case.

no location modifications or signalling operations 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 rejection message, using whatever policy the server would use in the absence of a CPL script.

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

location modifications performed, no signalling operations: 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 notfound 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 operation 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.

12 CPL Extensions

Servers MAY support additional CPL features beyond those listed in this document. Some of the extensions which have been suggested are a means of querying how a call has been authenticated; richer control over H.323 addressing; end-system or administrator-specific features; regular-expression matching for strings and addresses; mid-call or end-of-call controls; and the parts of iCal COS recurrence rules omitted from time switches.

CPL extensions are indicated by XML namespaces [18]. Every extension MUST have an appropriate XML namespace assigned to it. All XML tags and attributes that are part of the extension MUST be appropriately qualified so as to place them within that namespace.

Tags or attributes in a CPL script which are in the global namespace (i.e., not associated with any namespace) are equivalent to tags and attributes in the CPL namespace “http://www.rfc-editor.org/rfc/rfcxxxx.txt”.

A CPL server MUST reject any script which contains a reference to a namespace which it does not understand. It MUST reject any script which contains an extension tag or attribute which is not qualified to be in an appropriate namespace.

A CPL script SHOULD NOT specify any namespaces it does not use. For compatibility with non-namespace-aware parsers, a CPL script SHOULD NOT specify the base CPL namespace for a script which does not use any extensions.

A syntax such as

```
<extension-switch>
  <extension has="http://www.example.com/foo">
     [extended things]
  </extension>
  <otherwise>
     [non-extended things]
  </otherwise>
</extension-switch>
```

was suggested as an alternate way of handling extensions. This would allow scripts to be uploaded to a server without requiring a script author to somehow determine which extensions a server supports. However, experience developing other languages, notably Sieve [19], was that this added excessive complexity to languages. The extension-switch tag could, of course, itself be defined in a CPL extension.

It is unfortunately true that XML DTDs, such as the CPL DTD given in Appendix C, are not powerful enough to encompass namespaces, since the base XML specification (which defines DTDs) predates the XML namespace specification. XML schemas [20] are a work in progress to define a namespace-aware method for validating XML documents, as well as improving upon DTDs’ expressive power in many other ways.

13 Examples

13.1 Example: Call Redirect Unconditional

The script in Figure 18 is a simple script which redirects all calls to a single fixed location.

13.2 Example: Call Forward Busy/No Answer

The script in Figure 19 illustrates some more complex behavior. We see an initial proxy attempt to one address, with further operations if that fails. We also see how several outputs take the same action subtree, through the use of subactions.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

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

Figure 18: Example Script: Call Redirect Unconditional

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "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="8">
         <busy>
           <sub ref="voicemail" />
         </busy>
         <noanswer>
           <sub ref="voicemail" />
         </noanswer>
       </proxy>
    </location>
  </incoming>
</cpl>
```

Figure 19: Example Script: Call Forward Busy/No Answer

13.3 Example: Call Forward: Redirect and Default

The script in Figure 20 illustrates further proxy behavior. The server initially tries to proxy to a single address. If this attempt is redirected, a new redirection is generated using the locations returned. In all other failure cases for the proxy node, a default operation — forwarding to voicemail — is performed.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <subaction id="voicemail">
  </subaction>

  <incoming>
    <location url="sip:jones@jonespc.example.com">
       <proxy>
         <redirection>
           <redirect />
         </redirection>
         <default>
           <location url="sip:jones@voicemail.example.com" >
              <proxy />
           </location>
         </default>
       </proxy>
    </location>
  </incoming>
</cpl>
```

Figure 20: Example Script: Call Forward: Redirect and Default

13.4 Example: Call Screening

The script in Figure 21 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 operations. The server therefore proceeds with its default behavior, which would presumably be to contact the user.

13.5 Example: Priority and Language Routing

The script in Figure 22 illustrates service selection based on a call's priority value and language settings. If the call request had a priority of "urgent" or higher, the default script behavior is performed. Otherwise, the language string field is checked for the string "es" (Spanish). If it is present, the call is proxied to a Spanish-speaking operator; other calls are proxied to an English-speaking operator.

13.6 Example: Outgoing Call Screening

The script in Figure 23 illustrates a script filtering outgoing calls, in the form of a script which prevent 1-900 (premium) calls from being placed. This script also illustrates subdomain matching.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "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 21: Example Script: Call Screening

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <priority-switch>
      <priority greater="urgent" />
      <otherwise>
        <string-switch field="language">
          <string contains="es">
            <location url="sip:spanish@operator.example.com">
              <proxy />
            </location>
          </string>
          <otherwise>
            <location url="sip:english@operator.example.com">
              <proxy />
            </location>
          </otherwise>
        </string-switch>
      </otherwise>
    </priority-switch>
  </incoming>
</cpl>
```

Figure 22: Example Script: Priority and Language Routing

13.7 Example: Time-of-day Routing

Figure 24 illustrates time-based conditions and timezones.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <outgoing>
    <address-switch field="original-destination" subfield="tel">
      <address subdomain-of="1900">
        <reject status="reject"
                reason="Not allowed to make 1-900 calls." />
      </address>
    </address-switch>
  </outgoing>
</cpl>
```

Figure 23: Example Script: Outgoing Call Screening

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <time-switch tzid="America/New_York"
        tzurl="http://zones.example.com/tz/America/New_York">
      <time dtstart="20000703T090000" duration="PT8H"
            freq="weekly" byday="MO,TU,WE,TH,FR">
        <lookup source="registration">
          <success>
            <proxy />
          </success>
        </lookup>
      </time>
      <otherwise>
        <location url="sip:jones@voicemail.example.com">
          <proxy />
        </location>
      </otherwise>
    </time-switch>
  </incoming>
</cpl>
```

Figure 24: Example Script: Time-of-day Routing

13.8 Example: Location Filtering

Figure 25 illustrates filtering operations on the location set. In this example, we assume that version 0.9beta2 of the "Inadequate Software SIP User Agent" mis-implements some features, and so we must work around its problems. We assume, first, that the value of its “feature” parameter in caller preferences is known to be unreliable, so we ignore it; we also know that it cannot talk successfully to one particular mobile device we may have registered, so we remove that location from the location set. Once these two operations have been completed, call setup is allowed to proceed normally.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <string-switch field="user-agent">
      <string is="Inadequate Software SIP User Agent/0.9beta2">
        <lookup source="registration" ignore="feature">
           <success>
             <remove-location location="sip:me@mobile.provider.net">
               <proxy />
             </remove-location>
           </success>
        </lookup>
      </string>
    </string-switch>
  </incoming>
</cpl>
```

Figure 25: Example Script: Location Filtering

13.9 Example: Non-signalling Operations

Figure 26 illustrates non-signalling operations; in particular, alerting a user by electronic mail if the lookup server failed. The primary motivation for having 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.

13.10 Example: Hypothetical Extensions

The example in Figure 27 shows a hypothetical extension which implements distinctive ringing. The XML namespace “http://www.example.com/distinctive-ring” specifies a new node named ring.

The example in Figure 28 implements a hypothetical new attribute for address switches, to allow regular-expression matches. It defines a new attribute regex for the standard address node. In this example, the global namespace is not specified.

13.11 Example: A Complex Example

Finally, Figure 29 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 mobile phone, and all

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "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 26: Example Script: Non-signalling Operations

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl xmlns="http://www.ietf.org/internet-drafts/draft-ietf-iptel-cpl-
03.txt"
     xmlns:dr="http://www.example.com/distinctive-ring">
  <incoming>
    <address-switch field="origin">
      <address is="sip:boss@example.com">
         <dr:ring ringstyle="warble" />
      </address>
    </address-switch>
  </incoming>
</cpl>
```

Figure 27: Example Script: Hypothetical Distinctive-Ringing Extension

other calls are directed to voicemail. If the call setup failed, no operation is specified, so the server's default behavior is performed.

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "cpl.dtd">

<cpl>
  <incoming>
    <address-switch field="origin" subfield="user"
        xmlns:re="http://www.example.com/regex">
      <address re:regex="(.*\.smith|.*\.jones)">
         <reject status="reject"
            reason="I don't want to talk to Smiths or Joneses" />
      </address>
    </address-switch>
  </incoming>
</cpl>
```

Figure 28: Example Script: Hypothetical Regular-Expression Extension

```
<?xml version="1.0" ?>
<!DOCTYPE cpl PUBLIC "-//IETF//DTD RFCxxxx CPL 1.0//EN" "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="8">
        <busy>
          <sub ref="voicemail" />
        </busy>
        <noanswer>
          <address-switch field="origin">
            <address is="sip:boss@example.com">
              <location url="tel:+19175551212">
                <proxy />
              </location>
            </address>
            <otherwise>
              <sub ref="voicemail" />
            </otherwise>
          </address-switch>
        </noanswer>
      </proxy>
    </location>
  </incoming>
</cpl>
```

Figure 29: Example Script: A Complex Example

14 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 operations 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 operations are permitted.

15 IANA Considerations

This document registers the MIME type `application/cpl+xml`. See Section 3.2.

16 Acknowledgments

This document was reviewed and commented upon by IETF IP Telephony Working Group. We specifically acknowledge the following people for their help:

The outgoing call screening script was written by Kenny Hom.

Paul E. Jones contributed greatly to the mappings of H.323 addresses.

The text of the time-switch section was taken (lightly modified) from RFC 2445 [12], by Frank Dawson and Derik Stenerson.

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 [19], a language for user filtering of electronic mail messages.

Thomas F. La Porta and Jonathan Rosenberg had many useful discussions, contributions, and suggestions.

A An Algorithm for Resolving Time Switches

The following algorithm resolves, in constant time, whether a given instant falls within a repetition of a time-switch recurrence. Open-source Java code implementing this algorithm is available on the world wide web at `<http://www.cs.columbia.edu/~lennox/Cal-Code/>`.

1. Compute the time of the call, in the timezone of the time switch. (No step after this needs to consider time zones — all calculations are done using continuously-running standard Gregorian time.)

2. If the call time is earlier than dtstart, fail **NOMATCH.**

3. If the call time is less than duration after dtstart, succeed **MATCH.**

4. Determine the smallest unit specified in a byxxx rule or by the freq. Call this the *Minimum Unit.* Determine the previous instant (before the call time) when all the time units smaller than the minimum unit are the same as those of dtstart. (For all minimum units, the time-of-day must be the same as

dtstart. If the minimum unit is a week, the day-of-the-week must be the same as dtstart. If the minimum unit is a month, the day-of-the-month must be the same as dtstart. If the minimum unit is a year, the month and day-of-month must both be the same as dtstart. (Note that this means it may be necessary to roll back more than one minimum unit — if the minimum unit is a month, then some months do not have a 31st (or 30th or 29th) day; if the minimum unit is a year, then some years do not have a February 29th. In the Gregorian calendar, it is never necessary to roll back more than two months, or eight years (four years between 1904 and 2096).)

Call this instant the *Candidate Start Time.*

5. If the time between the candidate start time and the call time is more than the duration, fail **NOMATCH.**
6. If the candidate start time is later than the until parameter of the recurrence, fail **NOMATCH.**
7. Call the unit of the freq parameter of the recurrence the *Frequency Unit.* Determine the frequency unit enclosing the Candidate Start Time, and that enclosing dtstart. Calculate the number of frequency units that have passed between these two times. If this is not a multiple of the interval parameter, fail **NOMATCH.**
8. For every byxxx rule, confirm that the candidate start time matches one of the options specified by that byxxx rule. If not, fail **NOMATCH.**
9. Succeed **MATCH.**

B Suggested Usage of CPL with H.323

This appendix gives a suggested usage of CPL with H.323 [2]. Study Group 16 of the ITU, which developed H.323, is proposing to work on official CPL mappings for that protocol. This section is therefore not normative.

B.1 Usage of address-switch with H.323

Address switches are specified in Section 5.1. This section specifies the mapping between H.323 messages and the fields and subfields of address-switches

For H.323, the origin address corresponds to the alias addresses in the sourceAddress field of the Setup-UUIE user-user information element, and to the Q.931 [21] information element "Calling party number." If both fields are present, or if multiple aliases addresses for sourceAddress are present, which one has priority is a matter of local server policy; the server SHOULD use the same resolution as it would use for routing decisions in this case. Similarly, the destination address corresponds to the alias addresses of the destinationAddress field, and to the Q.931 information element "Called party number."

The original-destination address corresponds to the "Redirecting number" Q.931 information element, if it is present; otherwise it is the same as the destination address.

The mapping of H.323 addresses into subfields depends on the type of the alias address. An additional subfield type, alias-type, is defined for H.323 servers, corresponding to the type of the address. Possible values are dialedDigits, h323-ID, url-ID, transportID, email-ID, partyNumber, mobileUIM, and Q.931IE. If future versions of the H.323 specification define additional types of alias addresses, those names MAY also be used.

In versions of H.323 prior to version 4, dialedDigits was known as e164. The two names SHOULD be treated as synonyms.

The value of the address-type subfield for H.323 messages is “h323” unless the alias type is url-ID and the URL scheme is something other than h323; in this case the address-type is the URL scheme, as specified in Section 5.1.1 for SIP.

An H.323-aware CPL server SHOULD map the address subfields from the primary alias used for routing. It MAY also map subfields from other aliases, if subfields in the primary address are not present.

The following mappings are used for H.323 alias types:

dialedDigits, partyNumber, mobileUIM, and Q.931IE: the tel and user subfields are the string of digits, as is the “entire-address” form. The host and port subfields are not present.

url-ID: the same mappings are used as for SIP, in Section 5.1.1.

h323-ID: the user field is the string of characters, as is the “entire-address” form. All other subfields are not present.

email-ID: the user and host subfields are set to the corresponding parts of the e-mail address. The port and tel subfields are not present. The “entire-address” form corresponds to the entire e-mail address.

transportID: if the TransportAddress is of type “ipAddress,” “ipSourceRoute,” or “ip6Address,” the host subfield is set to the “ip” element of the sequence, translated into the standard IPv4 or IPv6 textual representation, and the port subfield is set to the “port” element of the sequence represented in decimal. The tel and user fields are not present. The “entire-address” form is not defined. The representation and mapping of transport addresses is not defined for non-IP addresses.

H.323 version 4 [22] and the Internet-Draft `draft-levin-iptel-h323-url-scheme-00` [23] define a “h323” URI scheme. This appendix defines a mapping for these URIs onto the CPL address-switch subfields, as given in Section 5.1. Neither of these documents has yet been formally published in a final form, so this appendix is non-normative.

For h323 URIs, the the user, host, and port subfields are set to the corresponding parts of the H.323 URL. The tel subfield is not present. The “entire-address” form corresponds to the entire URI.

This mapping MAY be used both for h323 URIs in an h323 url-ID address alias, and for h323 URIs in SIP messages.

B.2 Usage of string-switch with H.323

For H.323, the string-switch node (see Section 5.2) is used as follows. The field language corresponds to the H.323 UUIE language, translated to the format specified for that field. The field display corresponds to the Q.931 information element of the same name, copied verbatim. The fields subject, organization, and user-agent are not used and are never present.

> The display IE is conventionally used for Caller-ID purposes, so arguably it should be mapped to the display subfield of an address-match with the field originator. However, since a) it is a message-level information element, not an address-level one, and b) the Q.931 specification [21] says only that “[t]he purpose of the Display information element is to supply display information that may be displayed by the user,” it seems to be more appropriate to allow it to be matched in a string-switch instead.

B.3 Usage of priority-switch with H.323

All H.323 messages are considered to have priority normal for the purpose of a priority switch (see Section 5.4).

B.4 Usage of location with H.323

Locations in explicit location nodes (Section 6.1) are specified as URLs. Therefore, all locations added in this manner are interpreted as being of alias type url-ID in H.323.

Specifications of other H.323 address alias types will require a CPL extension (see Section 12).

B.5 Usage of lookup with H.323

For location lookup nodes (Section 6.2), the registration lookup source corresponds to the locations registered with the server using RAS messages.

As H.323 currently has no counterpart of SIP caller preferences and callee capabilities, the use and ignore parameters of the lookup node are ignored.

B.6 Usage of remove-location with H.323

For location removal nodes (Section 6.3), only literal URLs can be removed. No URL patterns are defined.

As H.323 currently has no counterpart of SIP caller preferences and callee capabilities, the param and value parameters of the remove-location node are ignored.

C 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 MAY allow minor variations from it, particularly in the ordering of the outputs 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.
-->

<!-- 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? ) >
<!-- <not-present> must appear at most once -->
<!ATTLIST address-switch
   field         CDATA    #REQUIRED
   subfield      CDATA    #IMPLIED
>

<!ELEMENT address ( %Node; ) >

<!ATTLIST address
   is            CDATA    #IMPLIED
   contains      CDATA    #IMPLIED
   subdomain-of  CDATA    #IMPLIED
> <!-- Exactly one of these three attributes must appear -->


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

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

```
<!-- <not-present> must appear at most once -->
<!ATTLIST string-switch
   field          CDATA    #REQUIRED
>

<!ELEMENT string ( %Node; ) >
<!ATTLIST string
   is             CDATA    #IMPLIED
   contains       CDATA    #IMPLIED
>  <!-- Exactly one of these two attributes must appear -->

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

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

<!ELEMENT time ( %Node; ) >

<!-- Exactly one of the two attributes "dtend" and "duration"
     must occur. -->
<!-- The value of "freq" is (daily|weekly|monthly|yearly).  It is
        case-insensitive, so it is not given as a DTD switch. -->
<!-- None of the attributes following freq are meaningful unless freq
         appears. -->
<!-- The value of "wkst" is (MO|TU|WE|TH|FR|SA|SU).  It is
        case-insensitive, so it is not given as a DTD switch. -->
<!ATTLIST time
   dtstart        CDATA  #REQUIRED
   dtend          CDATA  #IMPLIED
   duration       CDATA  #IMPLIED
   freq           CDATA  #IMPLIED
   until          CDATA  #IMPLIED
   interval       CDATA  "1"
   byday          CDATA  #IMPLIED
   bymonthday     CDATA  #IMPLIED
   byyearday      CDATA  #IMPLIED
   byweekno       CDATA  #IMPLIED
   bymonth        CDATA  #IMPLIED
   wkst           CDATA  "MO"
>
```

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

<!ELEMENT priority-switch ( (priority|not-present)+, otherwise? ) >
<!-- <not-present> must appear at most once -->

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

<!ELEMENT priority ( %Node; ) >

<!-- Exactly one of these three attributes must appear -->
<!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
   priority      CDATA    #IMPLIED
   %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?,redirection?,failure?,default? ) >

<!-- The default value of timeout is "20" if the <noanswer> output
     exists. -->
<!ATTLIST proxy
   timeout        CDATA    #IMPLIED
   recurse        (yes|no) "yes"
   ordering       CDATA    "parallel"
>

<!ELEMENT busy ( %Node; ) >
<!ELEMENT noanswer ( %Node; ) >
<!ELEMENT redirection ( %Node; ) >
<!-- "failure" repeats from lookup, above. -->
<!ELEMENT default ( %Node; ) >

<!ELEMENT redirect EMPTY >
<!ATTLIST redirect
   permanent      (yes|no) "no"
>


<!-- Statuses we can return -->

<!ELEMENT reject EMPTY >
<!-- The value of "status" is (busy|notfound|reject|error), or a SIP
     4xx-6xx status. -->
<!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 ( %Node; ) >
<!ATTLIST log
   name          CDATA    #IMPLIED
   comment       CDATA    #IMPLIED
>


<!-- Calls to subactions. -->

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


<!-- Ancillary data -->

<!ENTITY % Ancillary 'ancillary?' >

<!ELEMENT ancillary EMPTY >


<!-- Subactions -->

<!ENTITY % Subactions 'subaction*' >

<!ELEMENT subaction ( %Node; )>
<!ATTLIST subaction
   id            ID       #REQUIRED
>


<!-- Top-level actions -->

<!ENTITY % TopLevelActions 'outgoing?,incoming?' >

<!ELEMENT outgoing ( %Node; )>

<!ELEMENT incoming ( %Node; )>

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

<!ELEMENT cpl  ( %Ancillary;,%Subactions;,%TopLevelActions; ) >
```

D Changes from Earlier Versions

[Note to RFC Editor: please remove this appendix before publication as an RFC.]

D.1 Changes from Draft -03

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

- Removed an obsolete reference to a usage in examples which wasn't actually used anywhere.
- Added forward references to remove-location, mail and log, as well as location, in the XML syntax as examples of nodes that don't have explicit output tags.
- Made the usage of some terminology more consistent: "output" vs. "next node"; "action" vs. "operation" vs. "behavior"; "sub-actions" and "subactions"; "other operations" and "non-call operations" and "non-signalling operations"; "meta-information" and "ancillary information."
- The tel subfield of addresses which come from sip URIs should have its visual separators stripped.
- The default value of the priority value of the location node is 1.0.
- Corrected the media type of a set of URIs to `text/uri-list`, and added a reference to it.
- Added some wording clarifying how URI-based lookup queries work.
- Corrected the syntax of duration parameter in the examples.
- Performed some pre-RFC textual cleanups (e.g. removing the reference to the Internet-Draft URL from the XML namespace identifier).
- Re-worded text in the description of the Ancillary tag which implied that information could be placed in that node in the base CPL specification. Clarified that the tag is for use by extensions only.
- Expunged some references to sub-daily recurrences which had accidentally been left in the text.
- Updated bibliography to refer to the latest versions of the cited documents.
- Fixed a number of typographical errors.

D.2 Changes from Draft -02

- Reduced time-switches from the full iCal recurrence to an iCal subset. Added an appendix giving an algorithm to resolve time-switches.
- Added the extension mechanism.
- Made explicit how each node is dependent on protocol handling. Separated out protocol-specific information — for SIP in subsections of the main text, for H.323 in a non-normative appendix.

- Clarified some address mapping rules for H.323.
- Corrected the name of the "Redirecting number" in Q.931.
- Clarified that address matching on the password subfield is case-sensitive.
- Added a recommendation that TZID labels follow the usage of the Olson database.
- Added the priority parameter to location nodes.
- Added the default output to the proxy node.
- Made the meaning of the proxy node's outputs explicit.
- Added suggested content for the e-mail generated by mail nodes.
- Pointed out that "&" must be escaped in XML (this is relevant for mailto URIs).
- Pointed out that log names are logical names, and should not be interpreted as verbatim filenames.
- Added some examples.
- Clarified some wording.
- Fixed some typographical errors.

D.3 Changes from Draft -01

- Completely re-wrote changes to time switches: they are now based on iCal rather than on crontab.
- Timezone references are now defined within time switches rather than in the ancillary section. The ancillary section is now empty, but still defined for future use. To facilitate this, an explicit ancillary tag was added.
- Added XML document type identifiers (the public identifier and the namespace), and MIME registration information.
- Clarified that the not-present output can appear anywhere in a switch.
- Re-wrote H.323 address mappings. Added the alias-type subfield for H.323 addresses.
- Added the language and display string switch fields.
- Clarified why useless not-present outputs can appear in time and priority switches.
- Added the clear parameter to location and lookup nodes. (It had been in the DTD previously, but not in the text.)
- Weakened support for non-validating scripts from SHOULD to MAY, to allow the use of validating XML parsers.
- Added redirection output of proxy nodes.

- Clarified some aspects of how proxy nodes handle the location set.
- Added permanent parameter of redirect nodes.
- Add example script for outgoing call screening (from Kenny Hom)
- Updated example scripts to use the public identifier.
- Add omitted tag to example script for call forward busy/no answer
- Clarified in introduction that this document mainly deals with servers.
- Updated reference to RFC 2824 now that it has been published.
- Added explanatory text to the introduction to types of nodes.
- Numerous minor clarifications and wording changes.
- Fixed copy-and-paste errors, typos.

D.4 Changes from Draft -00

- 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.

E 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 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 (second edition),” W3C Recommendation REC-xml-20001006, World Wide Web Consortium (W3C), Oct. 2000. Available at http://www.w3.org/XML/.

[4] J. Lennox and H. Schulzrinne, “Call processing language framework and requirements,” Request for Comments 2824, Internet Engineering Task Force, May 2000.

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

[6] D. Raggett, A. Le Hors, and I. Jacobs, "HTML 4.01 specification," W3C Recommendation REC-html401-19991224, World Wide Web Consortium (W3C), Dec. 1999. Available at http://www.w3.org/TR/html4/.

[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. Murata, S. S. Laurent, and D. Kohn, "XML media types," Request for Comments YYYY, Internet Engineering Task Force, Sept. 2000. [Draft draft-murata-xml-09.txt, approved for Proposed Standard. RFC Editor: please fill in appropriate bibliographic information.].

[9] H. Alvestrand, "Tags for the identification of languages," Request for Comments 1766, Internet Engineering Task Force, Mar. 1995.

[10] M. Davis and M. Duerst, "Unicode normalization forms," Unicode Technical Report 15, Unicode Consortium, Aug. 2000. Revision 19; part of Unicode 3.0.1. Available at http://www.unicode.org/unicode/reports/tr15/.

[11] M. Davis, "Case mappings," Unicode Technical Report 21, Unicode Consortium, Oct. 2000. Revision 4.3. Available at http://www.unicode.org/unicode/reports/tr21/.

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

[13] P. Eggert, "Sources for time zone and daylight saving time data." Available at http://www.twinsun.com/tz/tz-link.htm.

[14] ISO (International Organization for Standardization), "Data elements and interchange formats — information interchange — representation of dates and times," ISO Standard ISO 8601:1988(E), International Organization for Standardization, Geneva, Switzerland, June 1986.

[15] M. Mealling and R. Daniel, "URI resolution services necessary for URN resolution," Request for Comments 2483, Internet Engineering Task Force, Jan. 1999.

[16] H. Schulzrinne and J. Rosenberg, "SIP caller preferences and callee capabilities," Internet Draft, Internet Engineering Task Force, July 2000. Work in progress.

[17] S. DeRose, E. Maler, D. Orchard, and B. Trafford, "XML linking language (XLink) version 1.0," W3C Candidate Recommendation CR-xlink-20000703, World Wide Web Consortium (W3C), July 2000. Available at http://www.w3.org/TR/xlink/.

[18] T. Bray, D. Hollander, and A. Layman, "Namespaces in XML," W3C Recommendation REC-xml-names-19900114, World Wide Web Consortium (W3C), Jan. 1999. Available at http://www.w3.org/TR/REC-xml-names/.

[19] T. Showalter, “Sieve: A mail filtering language,” Request for Comments YYYY, Internet Engineering Task Force, Aug. 2000. [Draft draft-showalter-sieve-12.txt, approved for Proposed Standard. RFC Editor: please fill in appropriate bibliographic information.].

[20] D. C. Fallside, “XML schema part 0: Primer,” W3C Candidate Recommendation CR-xmlschema-0-20001024, World Wide Web Consortium (W3C), Oct. 2000. Available at http://www.w3.org/TR/xmlschema-0/.

[21] International Telecommunication Union, “Digital subscriber signalling system no. 1 (dss 1) - isdn user-network interface layer 3 specification for basic call control,” Recommendation Q.931, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Mar. 1993.

[22] International Telecommunication Union, “Packet based multimedia communication systems,” Recommendation H.323 Draft v4, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, July 2000. To be published November 2000.

[23] O. Levin, “H.323 URL scheme definition,” Internet Draft, Internet Engineering Task Force, Aug. 2000. Work in progress.

Full Copyright Statement