Internet Engineering Task Force
INTERNET-DRAFT
draft-ietf-iptel-cpl-07.ps

IPTEL WG
Lennox/Wu/Schulzrinne
Columbia University
August 1, 2003
Expires: February, 2004

CPL: A Language for User Control of Internet Telephony Services

Status of this Memo

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

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

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

Copyright Notice

Copyright (c) The Internet Society (2003). All Rights Reserved.

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@ietf.org and/or the authors.

Contents

1 Introduction **3**
1.1 Conventions of This Document 4

2 Structure of CPL Scripts **4**
2.1 High-level Structure 4
2.2 Abstract Structure of a Call Processing Action 4
2.3 Location Model 5
2.4 XML Structure 5

3 Script Structure: Overview **7**

4 Switches **7**
4.1 Address Switches 7
4.1.1 Usage of address-switch with SIP 9
4.2 String Switches 10

4.2.1 Usage of string-switch with SIP . . . 11
4.3 Language Switches . . . 11
4.3.1 Usage of language-switch with SIP . . . 11
4.4 Time Switches . . . 12
4.4.1 iCalendar differences and implementation issues . . . 16
4.5 Priority Switches . . . 16
4.5.1 Usage of priority-switch with SIP . . . 17

5 Location Modifiers 17
5.1 Explicit Location . . . 17
5.1.1 Usage of location with SIP . . . 18
5.2 Location Lookup . . . 18
5.2.1 Usage of lookup with SIP . . . 19
5.3 Location Removal . . . 19
5.3.1 Usage of remove-location with SIP . . . 20

6 Signalling Operations 20
6.1 Proxy . . . 20
6.1.1 Usage of proxy with SIP . . . 21
6.2 Redirect . . . 22
6.2.1 Usage of redirect with SIP . . . 22
6.3 Reject . . . 22
6.3.1 Usage of reject with SIP . . . 23

7 Non-signalling Operations 23
7.1 Mail . . . 23
7.1.1 Suggested Content of Mailed Information . . . 24
7.2 Log . . . 24

8 Subactions 25

9 Ancillary Information 26

10 Default Behavior 26

11 CPL Extensions 27

12 Examples 27
12.1 Example: Call Redirect Unconditional . . . 27
12.2 Example: Call Forward Busy/No Answer . . . 28
12.3 Example: Call Forward: Redirect and Default . . . 29
12.4 Example: Call Screening . . . 29
12.5 Example: Priority and Language Routing . . . 29
12.6 Example: Outgoing Call Screening . . . 29
12.7 Example: Time-of-day Routing . . . 30
12.8 Example: Location Filtering . . . 31

12.9 Example: Non-signalling Operations 32
12.10 Example: Hypothetical Extensions 32
12.11 Example: A Complex Example 32

13 Security Considerations **37**

14 IANA Considerations **37**
14.1 URN Sub-Namespace Registration for urn:ietf:params:xml:ns:cpl 37
14.2 MIME Registration 38

15 Acknowledgments **38**

A An Algorithm for Resolving Time Switches **39**

B Suggested Usage of CPL with H.323 **40**
B.1 Usage of address-switch with H.323 40
B.2 Usage of string-switch with H.323 41
B.3 Usage of language-switch with H.323 41
B.4 Usage of priority-switch with H.323 41
B.5 Usage of location with H.323 41
B.6 Usage of lookup with H.323 43
B.7 Usage of remove-location with H.323 43

C The XML Schema for CPL **43**

D Changes from Earlier Versions **54**
D.1 Changes from Draft -06 54
D.2 Changes from Draft -05 55
D.3 Changes from Draft -04 55
D.4 Changes from Draft -03 56
D.5 Changes from Draft -02 56
D.6 Changes from Draft -01 57
D.7 Changes from Draft -00 58

E Authors' Addresses **59**

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 the Session Initiation Protocol (SIP) [1] and H.323 [16].

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.

CPL is also designed to be easily created and edited by graphical tools. It is based on the Extensible Markup Language (XML) [2], 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 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" [17].

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 [3] and indicate requirement levels for compliant CPL implementations.

> Some paragraphs are indented, like this; they give motivations of design choices, advice to implementors, or thoughts on future development of or extensions to CPL. They are not essential to the specification of the language, and are non-normative.

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 actions 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. CPL forbids subactions from being called recursively: see Section 8.

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 that 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; 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 10.

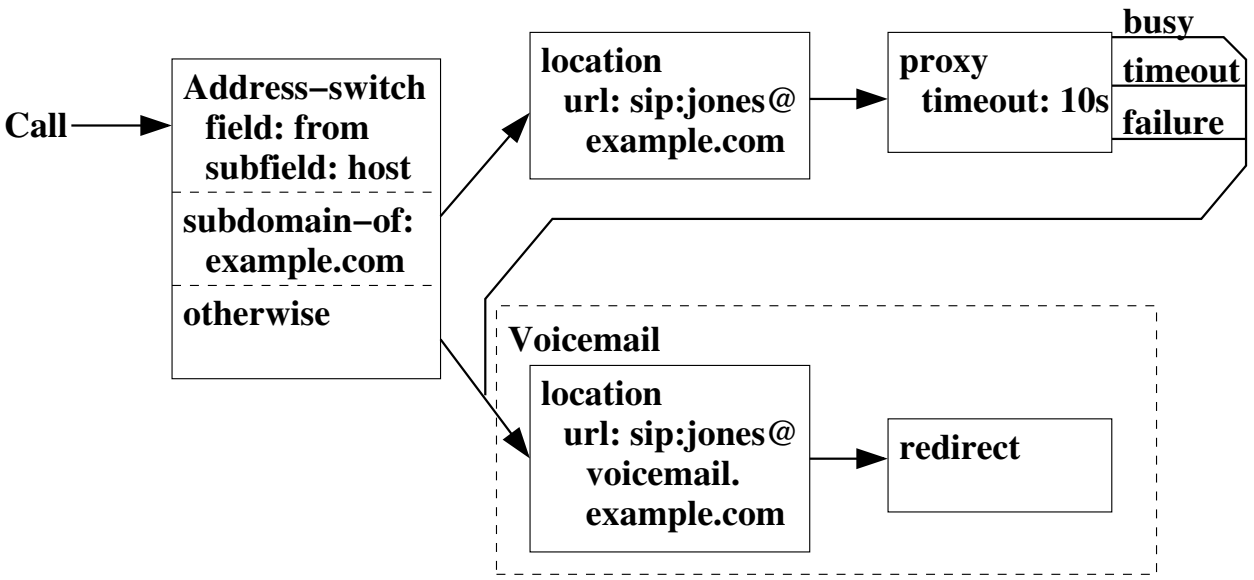

Figure 1: Sample CPL Action: Graphical Version

2.3 Location Model

For flexibility, one piece of information necessary for 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 5.

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 the XML specification [2], 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

```
<?xml version="1.0" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <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

visually and structurally very similar to HTML [18], as both languages are simplifications of the earlier and larger standard SGML [19].

See Figure 2 for the XML document corresponding to the graphical representation of the CPL script in Figure 1. Both nodes and outputs in 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 5.1, 5.3, 7.1, and 7.2).

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 8.

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 XML Schema for CPL is provided in Appendix C. The remainder of the main sections of this document describe the semantics of CPL, while giving its syntax informally. For the formal syntax, please see the appendix.

3 Script Structure: Overview

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

Tag:	cpl	
Parameters:	None	
Sub-tags:	ancillary	See Section 9
	subaction	See Section 8
	outgoing	Top-level actions to take on this user's outgoing calls
	incoming	Top-level actions to take on this user's incoming calls

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.

4 Switches

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

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 10 for more information on this.

Switches MAY contain no outputs. They MAY contain only an otherwise output.

> Such switches are not particularly useful, but might be created by tools which automatically generate CPL scripts.

4.1 Address Switches

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

Node:	address-switch	
Outputs:	address	Specific addresses to match
Parameters:	field	origin, destination, or original-destination
	subfield	address-type, user, host, port, tel, 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)

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 4.1.1; 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 [4] textual representation format. Host names are compared as strings. IP addresses are compared numerically. (In particular, the presence or location of an IPv6 :: omitted-zero-bits block is not significant for matching purposes.) Host names are never equal to IP addresses — no DNS resolution is performed. IPv4 addresses are never equal to IPv6 addresses, even if the IPv6 address is a v4-in-v6 embedding. This subfield is not case sensitive, and may be absent.

For host names only, subdomain matching is supported with the subdomain-of match operator. The subdomain-of operator ignores leading dots in the hostname or match pattern, if any.

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. Leading zeros are ignored.

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 4.2. 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.

4.1.1 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 URIs, the user, host, and port subfields correspond to the "user," "host," and "port" elements of the URI syntax. (Note that, following the definitions of RFC 3261 [1], a SIP URI which does not specify a port is not the same as an explicit port 5060; the former is indicated by an absent port subfield.) The tel subfield is defined to be the "user" part of the URI, with visual separators stripped, if the "user=phone" parameter is given to the URI, or if the server is otherwise configured to recognize the user part as a telephone

number. 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 is matches, standard SIP URI matching rules are used; for contains matches, the URI is used verbatim.

4.2 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, 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.

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 [5]. Then, strings are compared using locale-insensitive caseless mapping, as specified in Unicode Technical Report 21 [6].

Code to perform the first step, in Java and Perl, is available; see the links from Annex E of UTR 15 [5]. 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 tag of string matching is named string, and has a mandatory argument, one of is or contains, indicating whole-string match or substring match, respectively.

4.2.1 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 display is not used, and is never present.

4.3 Language Switches

Language switches allow a CPL script to make decisions based on the languages in which the originator of the call wishes to communicate. They are summarized in Figure 6.

Node:	language-switch	
Outputs:	language	Specific string to match
Parameters:	None	
Output:	language	
Parameters:	matches	Match if the given language matches a language-range of the call.

Figure 6: Syntax of the language-switch node

Language switches take no parameters.

The language output takes one parameter, matches. The value of the parameter is a language-tag, as defined in RFC 3066 [7]. The caller may have specified a set of language-ranges, also as defined in RFC 3066. The CPL server checks each language-tag specified by the script against the language-ranges specified in the request.

See RFC 3066 for the details of how language-ranges match language-tags. Briefly, a language-range matches a language-tag if it exactly equals the tag, or if it exactly equals a prefix of the tag such that the first character following the prefix is "-".

If the caller specified the special language-range "*", it is ignored for the purpose of matching. Languages with a q value of 0 are also ignored.

This switch MAY be not-present.

4.3.1 Usage of language-switch with SIP

The language-ranges for the language-switch switch are obtained from the SIP Accept-Language header field. The switch is not-present if the initial SIP request did not contain this header field.

Note that because of CPL's first-match semantics in switches, q values other than 0 of the Accept-Language header fields are ignored.

4.4 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 7.

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 (“secondly”, “minutely”, “hourly”, “daily”, “weekly”, “monthly”, or “yearly”)
	interval	How often the recurrence repeats
	until	Bound of recurrence (RFC 2445 DATE-TIME)
	count	Number of occurrences of recurrence
	bysecond	List of seconds within a minute
	byminute	List of minutes within an hour
	byhour	List of hours of the day
	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 the work week
	bysetpos	List of values within set of events specified

Figure 7: Syntax of the time-switch node

Time switches are based closely on the specification of recurring intervals of time in the Internet Calendaring and Scheduling Core Object Specification (iCalendar COS), RFC 2445 [8].

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

If future standards-track documents are published that update or obsolete RFC 2445, any changes or clarifications those documents make to recurrence handling apply to CPL time-switches as well.

An algorithm to determine 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 [9]. Examples of timezone databases that use the Olson scheme are the `zoneinfo` files on most Unix-like systems, and the standard Java `TimeZone` class.

Servers SHOULD resolve tzid and tzurl references to time zone definitions at the time the script is uploaded. They MAY periodically refresh these resolutions to obtain the most up-to-date definition of a time zone. If a tzurl becomes invalid, servers SHOULD remember the most recent valid data retrieved from the URL.

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 iCalendar COS DATE-TIME values, as specified in Section 4.3.5 of RFC 2445 [8]. 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 iCalendar 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 [20].

For a recurring interval, the duration parameter MUST be small enough such that subsequent intervals do not overlap. For non-recurring intervals, durations of any positive length are permitted. Zero-length and negative-length durations are not allowed.

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. Parameters other than dtstart, dtend, and duration SHOULD NOT be specified unless freq is present, though CPL servers SHOULD accept scripts with such parameters present, and ignore the other parameters.

The freq parameter takes one of the following values: secondly, to specify repeating periods based on an interval of a second or more; minutely, to specify repeating periods based on an interval of a minute or more; hourly, to specify repeating periods based on an interval of an hour or more; 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 interval parameter contains a positive integer representing how often the recurrence rule repeats. The default value is "1", meaning every second for a secondly rule, every minute for a minutely rule, every

hour for an hourly rule, 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 iCalendar 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, and the count parameter is not also present, the recurrence is considered to repeat forever.

The count parameter defines the number of occurrences at which to range-bound the recurrence. The dtstart parameter counts as the first occurrence. The until and count parameters MUST NOT occur in the same time output.

The bysecond parameter specifies a comma-separated list of seconds within a minute. Valid values are 0 to 59. The byminute parameter specifies a comma-separated list of minutes within an hour. Valid values are 0 to 59. The byhour parameter specifies a comma-separated list of hours of the day. Valid values are 0 to 23.

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 [20]. 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 work week 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 [20].

The bysetpos parameter specifies a comma-separated list of values which corresponds to the nth occurrence within the set of events specified by the rule. Valid values are 1 to 366 or -366 to -1. It MUST only be used in conjunction with another byxxx parameter. For example “the last work day of the month” could be

represented as:

```
<time -timerange- freq="monthly" byday="MO,TU,WE,TH,FR"
      bysetpos="-1">
```

Each bysetpos value can include a positive (+n) or negative (-n) integer. If present, this indicates the nth occurrence of the specific occurrence within the set of events specified by the rule.

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, byday, byhour, byminute, bysecond and bysetpos; then count and until are evaluated.

Here is an example of evaluating multiple Byxxx parameters.

```
<time dtstart="19970105T083000" duration="10M"
      freq="yearly" interval="2" bymonth="1" byday="SU"
      byhour="8,9" byminute="30">
```

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, `byhour="8,9"` would be applied to arrive at "every Sunday in January at 8 AM and 9 AM, every other year." Then, `byminute="30"` would be applied to arrive at "every Sunday in January at 8:30 AM and 9:30 AM, every other year." Then the second is derived from dtstart to end up in "every Sunday in January from 8:30:00 AM to 8:40:00 AM, and from and 9:30:00 AM to 9:40:00 AM, every other year." Similarly, if the byminute, byhour, byday, bymonthday or bymonth parameter were missing, the appropriate minute, hour, day or month would have been retrieved from the dtstart parameter.

The iCalendar 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 8) 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.

4.4.1 iCalendar differences and implementation issues

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

The specification of recurring events in this section is identical (except for syntax and formatting issues) to that of RFC 2445 [8], with only one additional restriction. That one restriction is that consecutive instances of recurrence intervals may not overlap.

It was a matter of some debate, during the design of CPL, whether the entire iCalendar COS recurrence specification should be included in CPL, or whether only a subset should be included. It was eventually decided that compatibility between the two protocols was of primary importance. This imposes some additional implementation issues on implementors of CPL servers.

It does not appear to be possible to determine, in constant time, whether a given instant of time falls within one of the intervals defined by a full iCalendar COS recurrence. The primary concerns are as follows:

- The count parameter cannot be checked in constant running time, since it requires that the server enumerate all recurrences from dtstart to the present time, in order to determine whether the current recurrence satisfies the parameter. However, a server can expand a count parameter once, off-line, to determine the date of the last recurrence. This date can then be treated as a virtual until parameter for the server's internal processing.

- Similarly, the bysetpos parameter requires that the server enumerate all instances of the occurrence from the start of the current recurrence set until the present time. This requires somewhat more complex pre-processing, but generally, a single recurrence with a bysetpos parameter can be split up into several recurrences without them.

- Finally, 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 potential repetitions of the recurrence. The restriction that consecutive intervals not overlap partially satisfies this condition, but does not fully ensure it. Again, to some extent pre-processing can help resolve this.

The algorithm given in Appendix A runs in constant time after these pre-processing steps.

Servers ought to check that recurrence rules do not create any absurd run-time or memory requirements, and reject those that do, just as they ought to check that CPL scripts in general are not absurdly large.

4.5 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 8. They are dependent on the underlying signalling protocol.

Priority switches take no parameters.

The priority tag takes one of the three parameters greater, less, and equal. The values of these parameters 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 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.

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

Figure 8: Syntax of the priority-switch node

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.

4.5.1 Usage of priority-switch with SIP

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

5 Location Modifiers

The abstract location model of CPL is described in Section 2.3. The behavior of several of the signalling operations (defined in Section 6) 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.

5.1 Explicit Location

Explicit location nodes specify a location literally. Their syntax is described in Figure 9.

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 9: 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.

5.1.1 Usage of location with SIP

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

5.2 Location Lookup

Locations can also be specified up through external means, through the use of location lookups. The syntax of these tags is given in Figure 10.

Location lookup is dependent on the underlying signalling protocol.

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
	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 10: Syntax of the lookup node

Location lookup nodes have one mandatory parameter and two 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 [10]). 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.

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 two optional parameters. The timeout parameter specifies the time, as a positive integer number of 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.

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.

5.2.1 Usage of lookup with SIP

For SIP, the registration lookup source corresponds to the locations registered with the server using REGISTER messages.

5.3 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 11.

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

Figure 11: 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 12.8.

The remove-location node has one optional parameter. 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 are removed from the set.

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.

5.3.1 Usage of remove-location with SIP

The location specified in the location parameter of the remove-location node is matched against the location set using the standard rules for SIP URI matching (as are used, e.g., to match Contact addresses when refreshing registrations).

6 Signalling Operations

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

6.1 Proxy

Proxy causes the triggering call to be forwarded on to the currently specified set of locations. The syntax of the proxy node is given in Figure 12.

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, as a positive integer number of 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 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

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 12: Syntax of the proxy node

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.

6.1.1 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 when determining location priority.

6.2 Redirect

Redirect causes the server to direct the calling party to attempt to place its call to the currently specified set of locations. The syntax of this node is specified in Figure 13.

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 13: 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."

6.2.1 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).

6.3 Reject

Reject nodes cause the server to reject the call attempt. Their syntax is given in Figure 14. The specific behavior they invoke is dependent on the underlying signalling protocol involved, though their semantics are generally applicable.

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

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

Figure 14: Syntax of the reject node

6.3.1 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

7 Non-signalling Operations

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

7.1 Mail

The mail node causes the server to notify a user of the status of the CPL script through electronic mail. Its syntax is given in Figure 15.

Node:	mail	
Outputs:	None	(Next node follows directly)
Next node:	Any node	
Parameters:	url	Mailto url to which the mail should be sent

Figure 15: 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" />`.

A mail node has 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 the XML specification [2]).

7.1.1 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.

7.2 Log

The Log node causes the server to log information about the call to non-volatile storage. Its syntax is specified in Figure 16.

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 16: Syntax of the log node

Log takes two arguments, both optional: name, which specifies the name of the log, and comment, which gives a comment about the information being logged. Servers SHOULD also include other information

in the log, such as the time of the logged event, information that triggered the call to be logged, and so forth. Logs are specific to the owner of the script which 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 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.

8 Subactions

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

Two tags are defined for subactions: subaction definitions and subaction references. Their syntax is given in Figure 17.

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 17: Syntax of subactions and sub pseudo-nodes

Subactions are defined through subaction tags. These tags are placed in CPL after any ancillary information (see Section 9) but before any top-level tags. They take one argument: id, a token indicating a script-chosen name for the subaction. The id value for every subaction tag in a script MUST be unique within that script.

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 [21]. Ensuring termination in the presence of external links is a difficult problem.

9 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 18.

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 18: Syntax of the ancillary tag

10 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.

11 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; and mid-call or end-of-call controls.

CPL extensions are indicated by XML namespaces [11]. 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 "urn:ietf:params:xml:ns:cpl".

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

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 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 [22], was that this added excessive complexity to languages. The extension-switch tag could, of course, itself be defined in a CPL extension.

In the XML schema of CPL, we introduce three abstract elements, namely 'toplevelaction', 'switch', and 'action', which accordingly have the abstract type 'TopLevelActionType', 'SwitchType', and 'ActionType'. Any top-level action in a CPL extension MUST be defined as the substitutionGroup of the abstract 'toplevelaction' element, and has the type extended from the 'TopLevelActionType'. Any switch in a CPL extension MUST be defined as the substitutionGroup of the abstract 'switch' element, and has the type extended from the 'SwitchType'. Any action in a CPL extension MUST be defined as the substitutionGroup of the abstract 'action' element, and has the type extended from the 'ActionType'.

12 Examples

12.1 Example: Call Redirect Unconditional

The script in Figure 19 is a simple script which redirects all calls to a single fixed location.

```
<?xml version="1.0" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <incoming>
    <location url="sip:smith@phone.example.com">
      <redirect/>
    </location>
  </incoming>
</cpl>
```

Figure 19: Example Script: Call Redirect Unconditional

12.2 Example: Call Forward Busy/No Answer

The script in Figure 20 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" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <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 20: Example Script: Call Forward Busy/No Answer

12.3 Example: Call Forward: Redirect and Default

The script in Figure 21 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" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <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 21: Example Script: Call Forward: Redirect and Default

12.4 Example: Call Screening

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

12.5 Example: Priority and Language Routing

The script in Figure 23 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 field is checked for the language "es" (Spanish). If it is present, the call is proxied to a Spanish-speaking operator; other calls are proxied to an English-speaking operator.

12.6 Example: Outgoing Call Screening

The script in Figure 24 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" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <incoming>
    <address-switch field="origin" subfield="user">
      <address is="anonymous">
        <reject status="reject" reason="I reject anonymous calls"/>
      </address>
    </address-switch>
  </incoming>
</cpl>
```

Figure 22: Example Script: Call Screening

```
<?xml version="1.0" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <incoming>
    <priority-switch>
      <priority greater="urgent"/>
      <otherwise>
        <language-switch>
          <language matches="es">
            <location url="sip:spanish@operator.example.com">
              <proxy/>
            </location>
          </language>
          <otherwise>
            <location url="sip:english@operator.example.com">
              <proxy/>
            </location>
          </otherwise>
        </language-switch>
      </otherwise>
    </priority-switch>
  </incoming>
</cpl>
```

Figure 23: Example Script: Priority and Language Routing

12.7 Example: Time-of-day Routing

Figure 25 illustrates time-based conditions and timezones.

```
<?xml version="1.0" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <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 24: Example Script: Outgoing Call Screening

```
<?xml version="1.0" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <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 25: Example Script: Time-of-day Routing

12.8 Example: Location Filtering

Figure 26 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 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 this operation has been completed, call setup is allowed to proceed normally.

```
<?xml version="1.0" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <incoming>
    <string-switch field="user-agent">
      <string is="Inadequate Software SIP User Agent/0.9beta2">
        <lookup source="registration">
          <success>
            <remove-location location="sip:me@mobile.provider.net">
              <proxy/>
            </remove-location>
          </success>
        </lookup>
      </string>
    </string-switch>
  </incoming>
</cpl>
```

Figure 26: Example Script: Location Filtering

12.9 Example: Non-signalling Operations

Figure 27 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.

12.10 Example: Hypothetical Extensions

The example in Figure 12.10 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 29 implements a hypothetical new attribute for address switches, to allow regular-expression matches. It defines a new attribute regex for the standard address node.

12.11 Example: A Complex Example

Finally, Figure 30 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 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" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <incoming>
    <lookup
        source="http://www.example.com/cgi-bin/locate.cgi?user=mary"
        timeout="8">
      <success>
        <proxy/>
      </success>
      <failure>
        <mail url="mailto:mary@example.com?subject=Lookup%20failed"/>
      </failure>
    </lookup>
  </incoming>
</cpl>
```

Figure 27: Example Script: Non-signalling Operations

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.example.com/distinctive-ring"
  xmlns="http://www.example.com/distinctive-ring"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:CPL="urn:ietf:params:xml:ns:cpl"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:import namespace="urn:ietf:params:xml:ns:cpl"
      schemaLocation="cpl.xsd"/>
  <xs:complexType name="DRingAction">
    <xs:complexContent>
      <xs:extension base="CPL:ActionType">
        <xs:attribute name="ringstyle" type="xs:string"
            use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="ring" type="DRingAction"
      substitutionGroup="CPL:action"/>
</xs:schema>

<?xml version="1.0" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:dr="http://www.example.com/distinctive-ring"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd
      http://www.example.com/distinctive-ring distinctive-ring.xsd">
  <incoming>
    <address-switch field="origin">
      <address is="sip:boss@example.com">
        <dr:ring ringstyle="warble"/>
      </address>
    </address-switch>
  </incoming>
</cpl>
```

Figure 28: Example Schema and Script: Hypothetical Distinctive-Ringing Extension

```
<?xml version="1.0" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <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 29: Example Script: Hypothetical Regular-Expression Extension

```
<?xml version="1.0" encoding="UTF-8"?>
<cpl xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:ietf:params:xml:ns:cpl cpl.xsd ">
  <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 30: Example Script: A Complex Example

13 Security Considerations

CPL is designed to allow services to be specified in a manner which prevents potentially hostile or misconfigured 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.

14 IANA Considerations

This document registers a new MIME type, `application/cpl+xml`, and a new URN per RFC 2141 [12], RFC 2648 [13], and RFC YYYY [14].

[Note to RFC Editor: please replace "YYYY" above with the RFC number of draft-mealling-iana-xmlns-registry, which is currently in the RFC Editor's queue, when it is published as an RFC.]

14.1 URN Sub-Namespace Registration for urn:ietf:params:xml:ns:cpl

URI: urn:ietf:params:xml:ns:cpl

Registrant Contact: Jonathan Lennox <lennox@cs.columbia.edu>
Xiaotao Wu <xiaotaow@cs.columbia.edu>
Henning Schulzrinne <hgs@cs.columbia.edu>

XML: BEGIN

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
    "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
     content="text/html;charset=iso-8859-1"/>
  <title>Call Processing Language Namespace</title>
</head>
<body>
  <h1>Namespace for Call Processing Language</h1>
  <h2>application/cpl+xml</h2>
  <p><a href="[[[URL of published RFC]]]">RFCXXXX</a>.</p>
</body>
</html>
```

END

[Note to RFC Editor: please replace "[[[URL of published RFC]]]" above with the official URL of this RFC at rfc-editor.org, and "XXXX" above with the number of this RFC.]

14.2 MIME Registration

As an XML type, CPL's MIME registration conforms with "XML Media Types," RFC 3023 [15].

MIME media type name: application

MIME subtype name: cpl+xml

Mandatory parameters: none

Optional parameters: charset
As for `application/xml` in RFC 3023.

Encoding considerations: As for `application/xml` in RFC 3023.

Security considerations: See Section 13, and Section 10 of RFC 3023.

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: A number of Internet telephony servers, mostly using SIP. Specifically, the authors of this specification have implemented it in their SIP proxy server (part of the CINEMA project).

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>
Xiaotao Wu <xiaotaow@cs.columbia.edu>
Henning Schulzrinne <hgs@cs.columbia.edu>

Intended usage: COMMON

Author/Change Controller: The IETF.

15 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 [8], 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 [22], a language for user filtering of electronic mail messages.

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

Richard Gumpertz performed a very useful last-minute technical and editorial review of the specification.

A An Algorithm for Resolving Time Switches

The following algorithm determines whether a given instant falls within a repetition of a time-switch recurrence. If the pre-processing described in Section 4.4.1 has been done, it operates in constant time. Open-source Java code implementing this algorithm is available on the world wide web at `<http://www.cs.columbia.edu/~lennox/Cal-Code/>`.

This algorithm is believed to be correct, but this section is non-normative. Section 4.4, and RFC 2445 [8], are the definitive definitions of recurrences.

1. Compute the time of the call, in the timezone of the time switch.
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 or equal to the call time) when all the time units smaller than the minimum unit are the same as those of dtstart. If the minimum unit is a second, this time is the same as the instant. If the minimum unit is a minute or an hour, the minutes or the minutes and hours, respectively, must be the same as dtstart. For all other 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 if the minimum unit is a month, or eight years if the minimum unit is a year. Between 1904 and 2096, it is never necessary to roll back more than four years — the eight-year rollback can only occur when the Gregorian calendar "skips" a leap year.

   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 (or the virtual until computed off-line from count), 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 so, succeed **MATCH.**

9. Calculate a previous candidate start time. Repeat until the difference between the candidate start time and the call time is more than the duration. If no candidate start time has been validated, fail **NOMATCH.**

B Suggested Usage of CPL with H.323

This appendix gives a suggested usage of CPL with H.323 [16]. 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 4.1. 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 [23] 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 4.1.1 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 4.1.1.

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 [16] defines an "h323" URI scheme. This appendix defines a mapping for these URIs onto the CPL address-switch subfields, as given in Section 4.1. This definition is also available as RFC 3508 [24], which is an excerpt from the H.323 specification.

For h323 URIs, 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 4.2) is used as follows. 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 [23] 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 language-switch with H.323

The language-ranges for the language-switch switch are obtained from the H.323 UUIE language. The switch is not-present if the initial message did not contain this UUIE.

B.4 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 4.5).

B.5 Usage of location with H.323

Locations in explicit location nodes (Section 5.1) 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 11).

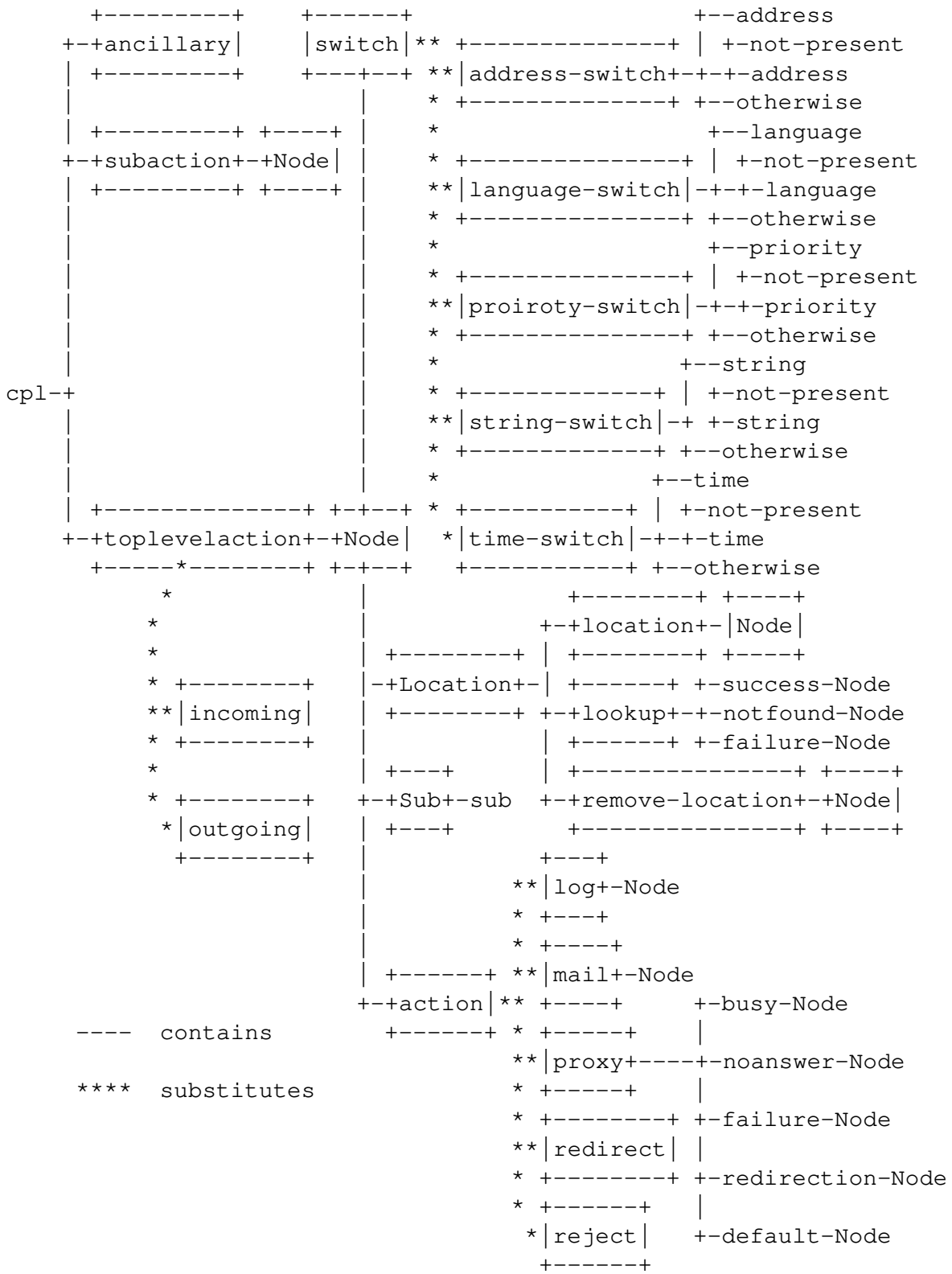

Figure 31: The structure of the XML schema for CPL

B.6 Usage of lookup with H.323

For location lookup nodes (Section 5.2), the registration lookup source corresponds to the locations registered with the server using RAS messages.

B.7 Usage of remove-location with H.323

Location removal nodes (Section 5.3) remove addresses with alias type url-ID using verbatim string matching on the URLs. If a tel URL is specified as the location, matching addresses (ignoring visual separators) with alias types dialedDigits (e164), partyNumber, mobileUIM, or Q.931IE are also removed. No mechanism is provided to remove other alias types.

C The XML Schema for CPL

This section includes a full XML Schema describing the XML syntax of CPL. Every script submitted to a CPL server SHOULD comply with this XML Schema. When parsing scripts comply with the CPL DTD in earlier drafts, the DOCTYPE lines in the scripts should be ignored. Note that compliance with this schema is not a sufficient condition for correctness of a CPL script, as many of the conditions described in this specification are not expressible in schema syntax. Figure B.4 shows the structure of the schema. 'incoming' and 'outgoing' are defined as the substitutionGroup of the 'toplevelaction'. All the switches are defined as the substitutionGroup of the 'switch' element. All the actions are defined as the substitutionGroup of the 'action' element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:cpl"
  xmlns="urn:ietf:params:xml:ns:cpl"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:complexType name="TopLevelActionType" abstract="true">
    <xs:group ref="Node"/>
  </xs:complexType>
  <xs:element name="toplevelaction" type="TopLevelActionType"/>
  <xs:complexType name="ActionType" abstract="true"/>
  <xs:element name="action" type="ActionType"/>
  <xs:complexType name="SwitchType" abstract="true"/>
  <xs:element name="switch" type="SwitchType"/>
  <xs:group name="Location">
    <xs:choice>
      <xs:element name="location" type="LocationType"/>
      <xs:element name="lookup" type="LookupType"/>
      <xs:element name="remove-location" type="RemoveLocationType"/>
    </xs:choice>
  </xs:group>
  <xs:group name="Sub">
    <xs:all>
```

```
      <xs:element name="sub" type="SubAction"/>
    </xs:all>
  </xs:group>
  <xs:group name="Node">
    <xs:choice>
      <xs:element ref="switch" minOccurs="0" maxOccurs="1"/>
      <xs:group ref="Location" minOccurs="0" maxOccurs="1"/>
      <xs:group ref="Sub" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="action" minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
  </xs:group>
  <xs:complexType name="OtherwiseAction">
    <xs:group ref="Node"/>
  </xs:complexType>
  <xs:complexType name="NotPresentAction">
    <xs:group ref="Node"/>
  </xs:complexType>
  <xs:simpleType name="YesNoType">
    <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="yes"/>
      <xs:enumeration value="no"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="StatusType">
    <xs:restriction base="xs:NMTOKEN">
      <xs:pattern value="busy"/>
      <xs:pattern value="notfound"/>
      <xs:pattern value="reject"/>
      <xs:pattern value="error"/>
      <xs:pattern value="[4-6][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="OrderingType">
    <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="parallel"/>
      <xs:enumeration value="sequential"/>
      <xs:enumeration value="first-only"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="DirectionType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="incoming"/>
      <xs:enumeration value="outgoing"/>
      <xs:enumeration value="both"/>
    </xs:restriction>
```

```
</xs:simpleType>
<xs:simpleType name="AddressFieldType">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="origin"/>
    <xs:enumeration value="destination"/>
    <xs:enumeration value="original-destination"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="AddressSubfieldType">
  <xs:restriction base="xs:NMTOKEN">
    <xs:enumeration value="address-type"/>
    <xs:enumeration value="user"/>
    <xs:enumeration value="host"/>
    <xs:enumeration value="port"/>
    <xs:enumeration value="tel"/>
    <xs:enumeration value="display"/>
    <xs:enumeration value="password"/>
    <xs:enumeration value="alias-type"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="AddressType">
  <xs:annotation>Exactly one of the three attributes must
      appear</xs:annotation>
  <xs:group ref="Node"/>
  <xs:attribute name="is" type="xs:string" use="optional"/>
  <xs:attribute name="contains" type="xs:string" use="optional">
    <xs:annotation>for "display" only</xs:annotation>
  </xs:attribute>
  <xs:attribute name="subdomain-of" type="xs:string"
      use="optional">
    <xs:annotation>for "host", "tel" only</xs:annotation>
  </xs:attribute>
  <xs:anyAttribute namespace="##any" processContents="lax"/>
</xs:complexType>
<xs:complexType name="AddressSwitchType">
  <xs:sequence>
    <xs:element name="address" type="AddressType" minOccurs="0"
        maxOccurs="unbounded"/>
    <xs:sequence minOccurs="0">
      <xs:element name="not-present" type="NotPresentAction"/>
      <xs:element name="address" type="AddressType" minOccurs="0"
          maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:element name="otherwise" type="OtherwiseAction"
        minOccurs="0"/>
```

```
    </xs:sequence>
    <xs:attribute name="field" type="AddressFieldType"
        use="required"/>
    <xs:attribute name="subfield" type="AddressSubfieldType"
        use="optional"/>
  </xs:complexType>
  <xs:element name="address-switch" type="AddressSwitchType"
      substitutionGroup="switch"/>
  <xs:simpleType name="StringFieldType">
    <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="subject"/>
      <xs:enumeration value="organization"/>
      <xs:enumeration value="user-agent"/>
      <xs:enumeration value="display"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="StringType">
    <xs:group ref="Node"/>
    <xs:attribute name="is" type="xs:string" use="optional"/>
    <xs:attribute name="contains" type="xs:string" use="optional"/>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>
  <xs:complexType name="StringSwitchType">
    <xs:sequence>
      <xs:element name="string" type="StringType" minOccurs="0"
          maxOccurs="unbounded"/>
      <xs:sequence minOccurs="0">
        <xs:element name="not-present" type="NotPresentAction"/>
        <xs:element name="string" type="StringType" minOccurs="0"
            maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:element name="otherwise" type="OtherwiseAction"
          minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="field" type="StringFieldType"
        use="required">
      <xs:annotation>Strings are matched as case-insensitive Unicode
          strings.</xs:annotation>
    </xs:attribute>
  </xs:complexType>
  <xs:element name="string-switch" type="StringSwitchType"
      substitutionGroup="switch"/>
  <xs:complexType name="LanguageType">
    <xs:group ref="Node"/>
    <xs:attribute name="matches" type="xs:string" use="required">
```

```
      <xs:annotation>The value of one of these parameters is a
          language-tag, as defined in RFC 3066.</xs:annotation>
    </xs:attribute>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>
  <xs:complexType name="LanguageSwitchType">
    <xs:sequence>
      <xs:element name="language" type="LanguageType" minOccurs="0"
          maxOccurs="unbounded"/>
      <xs:sequence minOccurs="0">
        <xs:element name="not-present" type="NotPresentAction"/>
        <xs:element name="language" type="LanguageType" minOccurs="0"
            maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:element name="otherwise" type="OtherwiseAction"
          minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="language-switch" type="LanguageSwitchType"
      substitutionGroup="switch"/>
  <xs:simpleType name="FreqType">
    <xs:restriction base="xs:NMTOKEN">
      <xs:pattern value="[s|S][e|E][c|C][o|O][n|N][d|D][l|L][y|Y]"/>
      <xs:pattern value="[m|M][i|I][n|N][u|U][t|T][e|E][l|L][y|Y]"/>
      <xs:pattern value="[h|H][o|O][u|U][r|R][l|L][y|Y]"/>
      <xs:pattern value="[d|D][a|A][i|I][l|L][y|Y]"/>
      <xs:pattern value="[w|W][e|E][e|E][k|K][l|L][y|Y]"/>
      <xs:pattern value="[m|M][o|N][n|N][t|T][h|H][l|L][y|Y]"/>
      <xs:pattern value="[y|Y][e|E][a|A][r|R][l|L][y|Y]"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="YearDayType">
    <xs:union>
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="-366"/>
          <xs:maxInclusive value="-1"/>
        </xs:restriction>
      </xs:simpleType>
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="1"/>
          <xs:maxExclusive value="366"/>
        </xs:restriction>
      </xs:simpleType>
```

```
    </xs:union>
  </xs:simpleType>
  <xs:simpleType name="DayType">
    <xs:restriction base="xs:NMTOKEN">
      <xs:pattern value="[m|M][o|O]"/>
      <xs:pattern value="[t|T][u|U]"/>
      <xs:pattern value="[w|W][e|E]"/>
      <xs:pattern value="[t|T][h|H]"/>
      <xs:pattern value="[f|F][r|R]"/>
      <xs:pattern value="[s|S][a|A]"/>
      <xs:pattern value="[s|S][u|U]"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="TimeType">
    <xs:annotation>Exactly one of the two attributes "dtend" and
        "duration" must occur. None of the attributes following freq
        are meaningful unless freq appears. </xs:annotation>
    <xs:group ref="Node"/>
    <xs:attribute name="dtstart" type="xs:string" use="required">
      <xs:annotation>RFC 2445 DATE-TIME</xs:annotation>
    </xs:attribute>
    <xs:attribute name="dtend" type="xs:string" use="optional">
      <xs:annotation>RFC 2445 DATE-TIME</xs:annotation>
    </xs:attribute>
    <xs:attribute name="duration" type="xs:string" use="optional">
      <xs:annotation>RFC 2445 DURATION</xs:annotation>
    </xs:attribute>
    <xs:attribute name="freq" type="FreqType" use="optional"/>
    <xs:attribute name="interval" type="xs:positiveInteger"
        default="1"/>
    <xs:attribute name="until" type="xs:string" use="optional">
      <xs:annotation>RFC 2445 DATE-TIME</xs:annotation>
    </xs:attribute>
    <xs:attribute name="count" type="xs:positiveInteger"
        use="optional"/>
    <xs:attribute name="bysecond" type="xs:string" use="optional">
      <xs:annotation>Comma-separated list of seconds within a minute.
          Valid values are 0 to 59.</xs:annotation>
    </xs:attribute>
    <xs:attribute name="byminute" type="xs:string" use="optional">
      <xs:annotation>Comma-separated list of minutes within an hour.
          Valid values are 0 to 59.</xs:annotation>
    </xs:attribute>
    <xs:attribute name="byhour" type="xs:string" use="optional">
      <xs:annotation>Comma-separated list of hours of the day. Valid
```

```
        values are 0 to 23.</xs:annotation>
    </xs:attribute>
    <xs:attribute name="byday" type="xs:string" use="optional">
      <xs:annotation>Comma-separated list of days of the week. Valid
        values are "MO", "TU", "WE", "TH", "FR", "SA" and "SU".
        These values are not case-sensitive. Each can be preceded
        by a positive (+n) or negative (-n)
        integer.</xs:annotation>
    </xs:attribute>
    <xs:attribute name="bymonthday" type="xs:string" use="optional">
      <xs:annotation>Comma-separated list of days of the month. Valid
        values are 1 to 31 or -31 to -1.</xs:annotation>
    </xs:attribute>
    <xs:attribute name="byyearday" type="xs:string" use="optional">
      <xs:annotation>Comma-separated list of days of the year. Valid
        values are 1 to 366 or -366 to -1.</xs:annotation>
    </xs:attribute>
    <xs:attribute name="byweekno" type="xs:string" use="optional">
      <xs:annotation>Comma-separated list of ordinals specifying
        weeks of the year. Valid values are 1 to 53 or -53 to
        -1.</xs:annotation>
    </xs:attribute>
    <xs:attribute name="bymonth" type="xs:string" use="optional">
      <xs:annotation>Comma-separated list of months of the year.
        Valid values are 1 to 12.</xs:annotation>
    </xs:attribute>
    <xs:attribute name="wkst" type="DayType" default="MO"/>
    <xs:attribute name="bysetpos" type="YearDayType"/>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>
  <xs:simpleType name="TZIDType">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="TZURLType">
    <xs:restriction base="xs:anyURI"/>
  </xs:simpleType>
  <xs:complexType name="TimeSwitchType">
    <xs:sequence>
      <xs:element name="time" type="TimeType" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:sequence minOccurs="0">
        <xs:element name="not-present" type="NotPresentAction"/>
        <xs:element name="time" type="TimeType" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
```

```
      <xs:element name="otherwise" type="OtherwiseAction"
          minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="tzid" type="TZIDType"/>
    <xs:attribute name="tzurl" type="TZURLType"/>
  </xs:complexType>
  <xs:element name="time-switch" type="TimeSwitchType"
      substitutionGroup="switch"/>
  <xs:simpleType name="PriorityValues">
    <xs:restriction base="xs:NMTOKEN">
      <xs:pattern
          value="[e|E][m|M][e|E][r|R][g|G][e|E][n|N][c|C][y|Y]"/>
      <xs:pattern value="[u|U][r|R][g|G][e|E][n|N][t|T]"/>
      <xs:pattern value="[n|N][o|O][r|R][m|M][a|A][l|L]"/>
      <xs:pattern
          value="[n|N][o|O][n|N]-[u|U][r|R][g|G][e|E][n|N][t|T]"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="PriorityType">
    <xs:annotation>Exactly one of the three attributes must appear
        </xs:annotation>
    <xs:group ref="Node"/>
    <xs:attribute name="less" type="PriorityValues"/>
    <xs:attribute name="greater" type="PriorityValues"/>
    <xs:attribute name="equal" type="xs:string">
      <xs:annotation>case-insensitive</xs:annotation>
    </xs:attribute>
    <xs:anyAttribute namespace="##any" processContents="lax"/>
  </xs:complexType>
  <xs:complexType name="PrioritySwitchType">
    <xs:sequence>
      <xs:element name="priority" type="PriorityType" minOccurs="0"
          maxOccurs="unbounded"/>
      <xs:sequence minOccurs="0">
        <xs:element name="not-present" type="NotPresentAction"/>
        <xs:element name="priority" type="PriorityType" minOccurs="0"
            maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:element name="otherwise" type="OtherwiseAction"
          minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="priority-switch" type="PrioritySwitchType"
      substitutionGroup="switch"/>
  <xs:simpleType name="LocationPriorityType">
```

```
    <xs:restriction base="xs:float">
      <xs:minInclusive value="0.0"/>
      <xs:maxInclusive value="1.0"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="LocationType">
    <xs:group ref="Node"/>
    <xs:attribute name="url" type="xs:anyURI" use="required"/>
    <xs:attribute name="priority" type="LocationPriorityType"
        use="optional" default="1.0"/>
    <xs:attribute name="clear" type="YesNoType" default="no"/>
  </xs:complexType>
  <xs:complexType name="LookupType">
    <xs:all>
      <xs:element name="success" minOccurs="0">
        <xs:complexType>
          <xs:group ref="Node"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="notfound" minOccurs="0">
        <xs:complexType>
          <xs:group ref="Node"/>
        </xs:complexType>
      </xs:element>
      <xs:element name="failure" minOccurs="0">
        <xs:complexType>
          <xs:group ref="Node"/>
        </xs:complexType>
      </xs:element>
    </xs:all>
    <xs:attribute name="source" type="xs:string" use="required"/>
    <xs:attribute name="timeout" type="xs:positiveInteger"
        default="30"/>
    <xs:attribute name="clear" type="YesNoType" default="no"/>
  </xs:complexType>
  <xs:complexType name="RemoveLocationType">
    <xs:group ref="Node"/>
    <xs:attribute name="location" type="xs:string" use="optional"/>
  </xs:complexType>
  <xs:complexType name="LogAction">
    <xs:complexContent>
      <xs:extension base="ActionType">
        <xs:group ref="Node"/>
        <xs:attribute name="name" type="xs:string" use="optional"/>
        <xs:attribute name="comment" type="xs:string"
```

```
            use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="log" type="LogAction"
      substitutionGroup="action"/>
  <xs:complexType name="IncomingType">
    <xs:complexContent>
      <xs:extension base="TopLevelActionType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="incoming" type="IncomingType"
      substitutionGroup="toplevelaction"/>
  <xs:complexType name="OutgoingType">
    <xs:complexContent>
      <xs:extension base="TopLevelActionType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="outgoing" type="OutgoingType"
      substitutionGroup="toplevelaction"/>
  <xs:complexType name="ProxyAction">
    <xs:complexContent>
      <xs:extension base="ActionType">
        <xs:all>
          <xs:element name="busy" minOccurs="0">
            <xs:complexType>
              <xs:group ref="Node"/>
            </xs:complexType>
          </xs:element>
          <xs:element name="noanswer" minOccurs="0">
            <xs:complexType>
              <xs:group ref="Node"/>
            </xs:complexType>
          </xs:element>
          <xs:element name="failure" minOccurs="0">
            <xs:complexType>
              <xs:group ref="Node"/>
            </xs:complexType>
          </xs:element>
          <xs:element name="redirection" minOccurs="0">
            <xs:complexType>
              <xs:group ref="Node"/>
            </xs:complexType>
          </xs:element>
          <xs:element name="default" minOccurs="0">
```

```
            <xs:complexType>
              <xs:group ref="Node"/>
            </xs:complexType>
          </xs:element>
        </xs:all>
        <xs:attribute name="timeout" type="xs:positiveInteger"
            use="optional" default="20"/>
        <xs:attribute name="recursive" type="YesNoType"
            use="optional" default="yes"/>
        <xs:attribute name="ordering" type="OrderingType"
            use="optional" default="parallel"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="proxy" type="ProxyAction"
      substitutionGroup="action"/>
  <xs:complexType name="RedirectAction">
    <xs:complexContent>
      <xs:extension base="ActionType">
        <xs:attribute name="permanent" type="YesNoType"
            default="no"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="redirect" type="RedirectAction"
      substitutionGroup="action"/>
  <xs:complexType name="RejectAction">
    <xs:complexContent>
      <xs:extension base="ActionType">
        <xs:attribute name="status" type="StatusType"
            use="required"/>
        <xs:attribute name="reason" type="xs:string"
            use="optional"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:element name="reject" type="RejectAction"
      substitutionGroup="action"/>
  <xs:complexType name="MailAction">
    <xs:complexContent>
      <xs:extension base="ActionType">
        <xs:group ref="Node"/>
        <xs:attribute name="url" type="xs:anyURI" use="required"/>
      </xs:extension>
    </xs:complexContent>
```

```
  </xs:complexType>
  <xs:element name="mail" type="MailAction"
      substitutionGroup="action"/>
  <xs:complexType name="SubAction">
    <xs:attribute name="ref" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:complexType name="AncillaryType"/>
  <xs:complexType name="SubactionType">
    <xs:group ref="Node"/>
    <xs:attribute name="id" use="required"/>
  </xs:complexType>
  <xs:element name="cpl">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ancillary" type="AncillaryType"
            minOccurs="0"/>
        <xs:element name="subaction" type="SubactionType"
            minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="toplevelaction" minOccurs="0"
            maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

D Changes from Earlier Versions

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

D.1 Changes from Draft -06

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

- Added Xiaotao Wu as a co-author.
- Converted CPL DTD to CPL XML Schema.
- Dropped all features dependent on caller preferences and callee capabilities.
- Added an XML namespace URN urn:ietf:params:xml:ns:cpl and registration information for it.
- Separated normative and informative references.
- Updated some references; most notably, updated SIP reference to RFC 3261. Updated text to reflect changes in these references.
- Allowed servers more flexibility about recognizing SIP addresses as telephone numbers.

- Restored some text, in the definition of interval, accidentally omitted when sub-day recurrences were re-added in draft -05.
- Clarified the usages of lookup and remove-location with SIP, and remove-location with H.323.
- Updated address of the IPTel working group's mailing list.
- Improved wording, cleaned up formatting, and corrected typos.

D.2 Changes from Draft -05

- Clarified that switch nodes are allowed to be degenerate — they can have no outputs, and they can have only an otherwise output.
- Clarified the (non-) usage of the special language-range "*".
- Clarified that the Candidate Start Time can be equal to the call time.
- Modified the DTD to require that the not-present output appear only once.
- Added DTD entries for the time-switch attributes re-added in draft -05.
- Updated the reference to ISO 8601 to cite 8601:2000.
- Updated all H.323 references to cite H.323v4.
- Corrected some spelling errors.

D.3 Changes from Draft -04

- Broke out language switches into their own switch node.
- Restored the full iCalendar COS recurrence specification. Added text describing the consequences of this for implementors, and expanded somewhat on the recurrence algorithm.
- Clarified when time zones are resolved.
- Spelled out "iCalendar" rather than abbreviating it "iCal."
- Clarified some points about host and port matching.
- Whole-address matching in SIP uses the standard SIP URL-match rules.
- Specified that proxy and lookup timeouts are positive integer number of seconds.
- Specified that subaction id parameters must be unique.
- Corrected example scripts' namespace and DTD references indicating older drafts of this document.
- Deleted an unused subaction from the "Call Forward: Redirect and Default" example script.
- Made empty switches legal in the DTD.

- Made the legal values for the proxy ordering parameter explicit in the DTD.
- Made the success output of lookup optional in the DTD. It can trigger a default action, just like anything else.
- Clarified that the time-switch resolution algorithm is non-normative.
- Updated references to previously-unpublished RFCs, now published.
- Thanked Richard Gumpertz.

D.4 Changes from Draft -03

- 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.5 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.6 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.7 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

Xiaotao Wu
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue, MC 0401
New York, NY 10027
USA
electronic mail: xiaotaow@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

Normative References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: session initiation protocol," RFC 3261, Internet Engineering Task Force, June 2002.

[2] 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/.

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

[4] R. Hinden and S. E. Deering, "IP version 6 addressing architecture," RFC 2373, Internet Engineering Task Force, July 1998.

[5] M. F. 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/.

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

[7] H. Alvestrand, "Tags for the identification of languages," RFC 3066, Internet Engineering Task Force, Jan. 2001.

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

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

[10] M. Mealling and R. W. Daniel, "URI resolution services necessary for URN resolution," RFC 2483, Internet Engineering Task Force, Jan. 1999.

[11] 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/.

[12] R. Moats, "URN syntax," RFC 2141, Internet Engineering Task Force, May 1997.

[13] R. Moats, "A URN namespace for IETF documents," RFC 2648, Internet Engineering Task Force, Aug. 1999.

[14] M. Mealling, "The IETF XML registry," internet draft, Internet Engineering Task Force, June 2003. Work in progress.

[15] M. Murata, S. S. Laurent, and D. Kohn, "XML media types," RFC 3023, Internet Engineering Task Force, Jan. 2001.

Informative References

[16] International Telecommunication Union, "Packet based multimedia communication systems," Recommendation H.323, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Nov. 2000.

[17] J. Lennox and H. Schulzrinne, "Call processing language framework and requirements," RFC 2824, Internet Engineering Task Force, May 2000.

[18] 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/.

[19] 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.

[20] ISO (International Organization for Standardization), "Data elements and interchange formats — information interchange — representation of dates and times," ISO Standard ISO 8601:2000(E), International Organization for Standardization, Geneva, Switzerland, Dec. 2000.

[21] 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/.

[22] T. Showalter, "Sieve: A mail filtering language," RFC 3028, Internet Engineering Task Force, Jan. 2001.

[23] 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, International Telecommunication Union, Geneva, Switzerland, Mar. 1993.

[24] O. Levin, "H.323 uniform resource locator (URL) scheme registration," RFC 3508, Internet Engineering Task Force, Apr. 2003.

Full Copyright Statement

Copyright (c) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an ”AS IS” basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.