Network Working Group
Request for Comments: 4722
Category: Informational

J. Van Dyke
E. Burger, Ed.
Cantata Technology, Inc.
A. Spitzer
Pingtel Corporation

November 2006

Media Server Control Markup Language (MSCML) and Protocol

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

Media Server Control Markup Language (MSCML) is a markup language used in conjunction with SIP to provide advanced conferencing and interactive voice response (IVR) functions. MSCML presents an application-level control model, as opposed to device-level control models. One use of this protocol is for communications between a conference focus and mixer in the IETF SIP Conferencing Framework.

Table of Contents

1. Introduction

This document describes the Media Server Control Markup Language (MSCML) and its usage. It describes payloads that one can send to a media server using standard SIP INVITE and INFO methods and the capabilities these payloads implement. RFC 4240 [2] describes media server SIP URI formats.

Prior to MSCML, there was not a standard way to deliver SIP-based enhanced conferencing. Basic SIP constructs, such as those described in RFC 4240 [2], serve simple n-way conferencing well. The SIP URI provides a natural mechanism for identifying a specific SIP conference, while INVITE and BYE methods elegantly implement conference join and leave semantics. However, enhanced conferencing applications also require features such as sizing and resizing, in-conference IVR operations (e.g., recording and playing participant names to the full conference), and conference event reporting. MSCML payloads within standard SIP methods realize these features.

The structure and approach of MSCML satisfy the requirements set out in RFC 4353 [10]. In particular, MSCML serves as the interface between the conference server or focus and a centralized conference mixer. In this case, a media server has the role of the conference mixer.

There are two broad classes of MSCML functionality. The first class includes primitives for advanced conferencing, such as conference configuration, participant leg manipulation, and conference event reporting. The second class comprises primitives for interactive voice response (IVR). These include collecting DTMF digits and playing and recording multimedia content.

MSCML fills the need for IVR and conference control with requests and responses over a SIP transport. VoiceXML [11] fills the need for IVR with requests and responses over a HTTP transport. This enables developers to use whatever model fits their needs best.

In general, a media server offers services to SIP UACs, such as Application Servers, Feature Servers, and Media Gateway Controllers. See the IPCC Reference Architecture [12] for definitions of these terms. It is unlikely, but not prohibited, for end-user SIP UACs to have a direct signaling relationship with a media server. The term "client" is used in this document to refer generically to an entity that interacts with the media server using SIP and MSCML.

The media server fulfills the role of the Media Resource Function (MRF) in the IP Multimedia Subsystem (IMS) [13] as described by 3GPP. MSCML and RFC 4240 [2], upon which MSCML builds, are specifically focused on the Media resource (Mr) interface which supports interactions between application logic and the MRF.

This document describes a working framework and protocol with which there is considerable implementation experience. Application developers and service providers have created several MSCML-based services since the availability of the initial version in 2001. This experience is highly relevant to the ongoing work of the IETF, particularly the SIP [26], SIPPING [27], MMUSIC [28], and XCON [29] work groups, the IMS [30] work in 3GPP, and the CCXML work in the Voice Browser Work Group of the W3C.

1.1. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

2. MSCML Approach

It is critically important to emphasize that the goal of MSCML is to provide an application interface that follows the SIP, HTTP, and XML development paradigm to foster easier and more rapid application deployment. This goal is reflected in MSCML in two ways.

First, the programming model is that of peer to peer rather than master-slave. Importantly, this allows the media server to be used simultaneously for multiple applications rather than be tied to a single point of control. It also enables standard SIP mechanisms to be used for media server location and load balancing.

Second, MSCML defines constructs and primitives that are meaningful at the application level to ensure that programmers are not distracted by unnecessary complexity. For example, the mixing resource operates on constructs such as conferences and call participants rather than directly on individual media streams.

The MSCML paradigm is important to the developer community, in that developers and operators conceptually write applications about calls, conferences, and call legs. For the majority of developers and applications this approach significantly simplifies and speeds development.

3. Use of SIP Request Methods

As mentioned above, MSCML payloads may be carried in either SIP INVITE or INFO requests. The initial INVITE, which creates an enhanced conference, MAY include an MSCML payload. A subsequent INVITE to the same Request-URI joins a participant leg to the conference. This INVITE MAY include an MSCML payload. The initial INVITE that establishes an IVR session MUST NOT include an MSCML payload. The client sends all mid-call MSCML payloads for conferencing and IVR via SIP INFO requests.

SIP INVITE requests that contain both MSCML and Session Description Protocol (SDP) body parts are used frequently in conferencing scenarios. Therefore, the media server MUST support message bodies with the MIME type "multipart/mixed" in SIP INVITE requests.

The media server transports MSCML responses in the final response to the SIP INVITE containing the matching MSCML request or in a SIP INFO message. The only allowable final response to a SIP INFO containing a message body is a 200 OK, per RFC 2976 [3]. Therefore, if the client sends the MSCML request via SIP INFO, the media server responds with the MSCML response in a separate INFO request. In general, these responses are asynchronous in nature and require a separate transaction due to timing considerations.

There has been considerable debate on the use of the SIP INFO method for any purpose. Our experience is that MSCML would not have been possible without it. At the time the first MSCML specification was published, the first SIP Event Notification draft had just been submitted as an individual submission. At that time, there was no mechanism to link SUBSCRIBE/NOTIFY to an existing dialog. This prevented its use in MSCML, since all events occurred in an INVITE-established dialog. And while SUBSCRIBE/NOTIFY was well suited for reporting conference events, its semantics seemed inappropriate for modifying a participant leg or conference setting where the only "event" was the success or failure of the request. Lastly, since SIP INFO was an established RFC, most SIP stack implementations supported it at that time. We had few, if any, interoperability issues as a result.

More recent developments have provided additional reasons why SUBSCRIBE/NOTIFY is not appropriate for use in MSCML. Use of SUBSCRIBE presents two problems. The first is semantic. The purpose of SUBSCRIBE is to register interest in User Agent state. However, using SUBSCRIBE for MSCML results in the SUBSCRIBE modifying the User Agent state. The second reason SUBSCRIBE is not appropriate is because MSCML is inherently call based. The association of a SIP dialog with a call leg means MSCML can be incredibly straightforward.

For example, if one used SUBSCRIBE or other SIP method to send commands about some context, one must identify that context somehow. Relating commands to the SIP dialog they arrive on defines the context for free. Moreover, it is conceptually easy for the developer. Using NOTIFY to transport MSCML responses is also not appropriate, as the NOTIFY would be in response to an implicit subscription. The SIP and SIPPING lists have discussed the dangers of implicit subscription.

In order to guarantee interoperability with this specification, as well as with SIP User Agents that are unaware of MSCML, SIP UACs that wish to use MSCML services MUST specify a service indicator that supports MSCML in the initial INVITE. RFC 4240 [2] defines the service indicator "conf", which MUST be used for MSCML conferencing applications. The service indicator "ivr" MUST be used for MSCML interactive voice response applications. In this specification, only "conf" and "ivr" are described.

The media server MUST support moving the call between services through sending the media server a BYE on the existing dialog and establishing a new dialog with an INVITE to the desired service. Media servers SHOULD support moving between services without requiring modification of the previously established SDP parameters. This is achieved by sending a re-INVITE on the existing dialog in which the Request-URI is modified to specify the new service desired by the client. This eliminates the need for the client to send an INVITE to the caller or gateway to establish new SDP parameters.

The media server, as a SIP UAS, MUST respond appropriately to an INVITE that contains an MSCML body. If MSCML is not supported, the media server MUST generate a 415 final response and include a list of the supported content types in the response per RFC 3261 [4]. The media server MUST also advertise its support of MSCML in responses to OPTIONS requests, by including "application/mediaservercontrol+xml" as a supported content type in an Accept header. This alleviates the major issues with using INFO for the transport of application data; namely, the User Agent's proper interpretation of what is, by design, an opaque message request.

4. MSCML Design

4.1. Transaction Model

To avoid undue complexity, MSCML establishes two rules regarding its usage. The first is that only one MSCML body may be present in a SIP request. The second is that each MSCML body may contain only one request or response. This greatly simplifies transaction management. MSCML syntax does provide for the unique identification of multiple requests in a single body part. However, this is not supported in this specification.

Per the guidelines of RFC 3470 [14], MSCML bodies MUST be well formed and valid.

MSCML is a direct request-response protocol. There are no provisional responses, only final responses. A request may, however, result in multiple notifications. For example, a request for active talker reports will result in a notification for each speaker set. This maps to the three major element trees for MSCML: <request>, <response>, and <notification>.

Figure 1 shows a request body. Depending on the command, one can send the request in an INVITE or an INFO. Figure 2 shows a response body. The SIP INFO method transports response bodies. Figure 3 shows a notification body. The SIP INFO method transports notifications.

```
<?xml version="1.0" encoding="utf-8"?>
<MediaServerControl version="1.0">
  <request>
    ... request body ...
  </request>
</MediaServerControl>
```

Figure 1: MSCML Request Format

```
<?xml version="1.0" encoding="utf-8"?>
<MediaServerControl version="1.0">
  <response>
    ... response body ...
  </response>
</MediaServerControl>
```

Figure 2: MSCML Response Format

```
<?xml version="1.0" encoding="utf-8"?>
<MediaServerControl version="1.0">
  <notification>
    ... notification body ...
  </notification>
</MediaServerControl>
```

Figure 3: MSCML Notification Format

MSCML requests MAY include a client-defined ID attribute for the purposes of matching requests and responses. The values used for these IDs need only be unique within the scope of the dialog in which the requests are issued.

4.2. XML Usage

In the philosophy of XML as a text-based description language, and not as a programming language, MSCML makes the choice of many attribute values for readability by a human. Thus, many attributes that would often be "boolean" instead take "yes" or "no" values. For example, what does 'report="false"' or 'report="1"' mean? However, 'report="yes"' is clearer: I want a report. Some programmers prefer the precision of a boolean. To satisfy both styles, MSCML defines an XML type, "yesnoType", that takes on the values "yes" and "no" as well as "true", "false", "1", and "0".

Many attributes in the MSCML schema have default values. In order to limit demands on the XML parser, MSCML applies these values at the protocol, not XML, level. The MSCML schema documents these defaults as XML annotations to the appropriate attribute.

4.2.1. MSCML Time Values

For clarity, time values in MSCML are based on the time designations described in the Cascading Style Sheets level 2 (CSS2) Specification [15]. Their format consists of a number immediately followed by an optional time unit identifier of the following form:

- ms: milliseconds (default)
- s: seconds

If no time unit identifier is present, the value MUST be interpreted as being in milliseconds. As extensions to [15] MSCML allows the string values "immediate" and "infinite", which have special meaning for certain timers.

5. Advanced Conferencing

5.1. Conference Model

The advanced conferencing model is a star controller model, with both signaling and media directed to a central location. Figure 4 depicts a typical signaling relationship between end users' UACs, a conference application server, and a media server.

RFC 4353 [10] describes this model. The application server is an instantiation of the conference focus. The media server is an instantiation of the media mixer. Note that user-level constructs, such as event notifications, are in the purview of the application server. This is why, for example, the media server sends active talker reports using MSCML notifications, while the application server would instead use the conference package [16] for individual notifications to SIP user agents. Note that we do not recommend the use of the conference package for media server to application server notifications because none of the filtering and membership information is available at the media server.

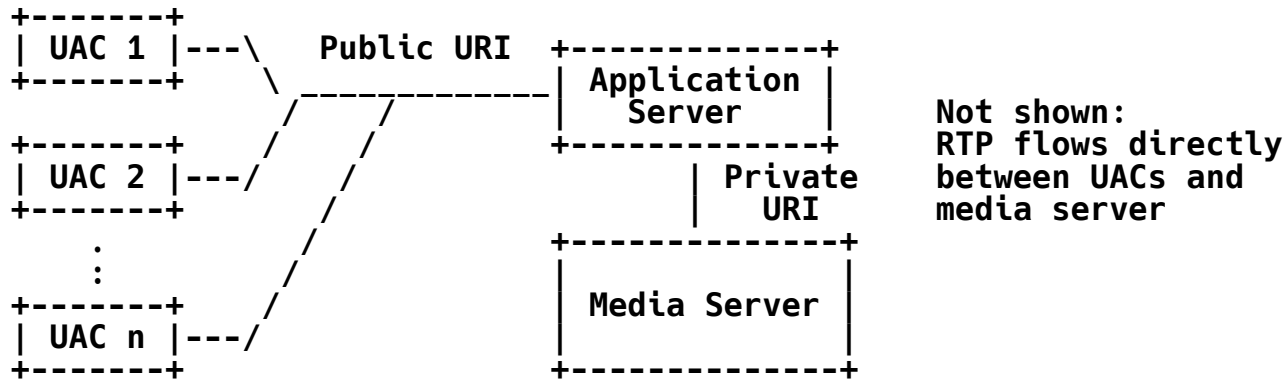

Figure 4: Conference Model

Each UAC sends an INVITE to a Public Conference URI. Presumably, the client publishes this URI, or it is an ad hoc URI. In any event, the client generates a Private URI, following the rules specified by RFC 4240 [2]. That is, the URI is of the following form:

sip:conf=UniqueID@ms.example.net

where UniqueID is a unique conference identifier and ms.example.net is the host name or IP address of the media server. There is nothing to prevent the UACs from contacting the media

server directly. However, one would expect the owner of the media server to restrict who can use its resources.

As for basic conferencing, described by RFC 4240 [2], the first INVITE to the media server with a UniqueID creates a conference. However, in advanced conferencing, the first INVITE MAY include a MSCML <configure_conference> payload rather than the SDP of a conference participant. The <configure_conference> payload conveys extended session parameters (e.g., number of participants) that SDP does not readily express, but the media server must know to allocate the appropriate resources.

When the conference is created by sending an INVITE containing a MSCML <configure_conference> payload, the resulting SIP dialog is termed the "Conference Control Leg." This leg has several useful properties. The lifetime of the conference is the same as that of its control leg. This ensures that the conference remains in existence even if all participant legs leave or have not yet arrived. In addition, when the client terminates the Conference Control Leg, the media server automatically terminates all participant legs. The Conference Control Leg is also used for play or record operations to/from the entire conference and for active talker notifications. Full conference media operations and active talker report subscriptions MUST be executed on the Conference Control Leg.

Creation of a Conference Control Leg is RECOMMENDED because full advanced conferencing capabilities are not available without it. Clients MUST establish the Conference Control Leg in the initial INVITE that creates the conference; it cannot be created later.

Once the client has created the conference with or without the Conference Control Leg, participants can be joined to the conference. This is achieved by the client's directing an INVITE to the Private Conference URI for each participant. Using the example conference URI given above, this would be sip:conf=UniqueID@ms.example.net.

5.2. Configure Conference Request <configure_conference>

The <configure_conference> request has two attributes that control the resources the media server sets aside for the conference. These are described in the list below.

Attributes of <configure_conference>:

- reservedtalkers - optional (see note), no default value: The maximum number of talker legs allocated for the conference. Note:

required when establishing the Conference Control Leg but optional in subsequent <configure_conference> requests.

- reserveconfmedia - optional, default value "yes": Controls allocation of resources to enable playing or recording to or from the entire conference

When the reservedtalkers+1st INVITE arrives at the media server, the media server SHOULD generate a 486 Busy Here response. Failure to send a 486 response to this condition can cause the media server to oversubscribe its resources.

NOTE: It would be symmetric to have a reservedlisteners parameter. However, the practical limitation on the media server is the number of talkers for a mixer to monitor. In either case, the client regulates who gets into the conference by either proxying the INVITEs from the user agent clients or metering to whom it gives the conference URI.

For example, to create a conference with up to 120 active talkers and the ability to play audio into the conference or record portions or all of the conference full mix, the client specifies both attributes, as shown in Figure 6.

```
<?xml version="1.0" encoding="utf-8"?>
<MediaServerControl version="1.0">
  <request>
    <configure_conference reservedtalkers="120"
      reserveconfmedia="yes"/>
  </request>
</MediaServerControl>
```

Figure 6: 120 Speaker MSCML Example

In addition to these attributes, a <configure_conference> request MAY contain a child <subscribe> element. The <subscribe> element is used to request notifications for conference-wide active talker events. Detailed information regarding active talker events is contained in Section 5.7.

The client MUST include a <configure_conference> request in the initial INVITE which establishes the conference when creating the Conference Control Leg. The client server MUST issue asynchronous commands, such as <play>, separately (i.e., in INFO messages) to avoid ambiguous responses.

Media operations on the Conference Control leg are performed internally, no external RTP streams are involved. Accordingly, the

media server does not expect RTP on the Conference Control Leg. Therefore, the client MUST send either no SDP or hold SDP in the INVITE request containing a <configure_conference> payload. The media server MUST treat SDP with all media lines set to "inactive" or with connection addresses set to 0.0.0.0 (for backwards compatibility) as hold SDP.

The media server sends a response when it has finished processing the <configure_conference> request. The format of the <configure_conference> response is detailed in Section 10.2.

5.3. Configure Leg Request <configure_leg>

Conference legs have a number of properties the client can modify. These are set using the <configure_leg> request. This request has the attributes described in the list below.

Attributes of <configure_leg>:

- type - optional, default value "talker": Consider this leg's audio for inclusion in the output mix. Alternative is "listener".
- dtmfclamp - optional, default value "yes": Remove detected DTMF digits from the input audio.
- toneclamp - optional, default value "yes": Remove tones from the input audio. Tones include call progress tones and the like.
- mixmode - optional, default value "full": Be a candidate for the full mix. Alternatives are "mute", to disallow media in the mix, "parked", to disconnect the leg's media streams from the conference for IVR operations, "preferred", to give this stream preferential selection in the mix (i.e., even if not loudest talker, include media, if present, from this leg in the mix), and "private", which enables personalized mixes.

In addition to these attributes, there are four child elements defined for <configure_leg>. These are <inputgain>, <outputgain>, <configure_team>, and <subscribe>.

The first two, <inputgain> and <outputgain>, modify the gain applied to the input and output audio streams, respectively. These may contain <auto>, to use automatic gain control (AGC) or <fixed>. The <auto> element has the attributes "startlevel", "targetlevel", and "silencethreshold". All the parameters are in dB. The <fixed> element has the attribute "level", which is in dB. The default for both <inputgain> and <outputgain> is <fixed>. The media server MAY

silently cap <inputgain> or <outputgain> requests that exceed the gain limits imposed by the platform.

Clients most commonly manipulate only the input gain for a conference leg and rely on the mixer to set an optimum output gain based on the inputs currently in the mix. However, as described above, MSCML does allow for manipulation of the output gain as well. Some of the IVR commands, such as <play>, enable control of the output gain for content playback operations. The interaction of conference output gain and IVR playback gain controls is described in Section 6.1.1. Note that <inputgain> and <outputgain> settings apply only to conference legs and do not apply to IVR sessions.

The <configure_team> element is used to create and manipulate groups for personalized mixes. Details of personalized mixes are discussed in Section 5.8.

The <subscribe> element is used to request notifications for call leg related events, such as asynchronous DTMF digit reports. Detailed information regarding call leg events is discussed in Section 7.

If the default parameters are acceptable for the leg the client wishes to enter into the conference, then a normal SIP INVITE, with no MSCML body, is sufficient. However, if the client wishes to modify one or more of the parameters, the client can include a MSCML body in addition to the SDP body.

The client can modify the conference leg parameters during the conference by issuing a SIP INFO on the dialog representing the conference leg. Of course, the client cannot modify SDP in an INFO message.

The media server sends a response when it has finished processing the <configure_leg> request. The format of the <configure_leg> response is detailed in Section 10.3.

5.4. Terminating a Conference

To remove a leg from the conference, the client issues a SIP BYE request on the selected dialog representing the conference leg.

The client can terminate all legs in a conference by issuing a SIP BYE request on the Conference Control Leg. If one or more participants are still in the conference when the media server receives a SIP BYE request on the Conference Control Leg, the media server issues SIP BYE requests on all remaining conference legs to ensure cleanup of the legs.

The media server returns a 200 OK to the SIP BYE request as it sends BYE requests to the other legs. This is because we cannot issue a provisional response to a non-INVITE request, yet the teardown of the other legs may exceed the retransmission timer limits of the original request. While the conference is being cleaned up, the media server MUST reject any new INVITEs to the terminated conference with a 486 Busy Here response. This response indicates that the specified conference cannot accept any new members, pending deletion.

5.5. Conference Manipulation

Once the conference has begun, the client can manipulate the conference as a whole or a particular participant leg by issuing commands on the associated SIP dialog. For example, by sending MSCML requests on the Conference Control Leg the client can request that the media server record the conference, play a prompt to the conference, or request reports on active talker events. Similarly, the client may mute a participant leg, configure a personalized mix or request reports for call leg events, such as DTMF keypresses.

Figure 7 shows an example of an MSCML command that plays a prompt to all conference participants.

```
<?xml version="1.0" encoding="utf-8"?>
<MediaServerControl version="1.0">
  <request>
    <play>
      <prompt>
        <audio url="http://prompts.example.net/en_US/welcome.au"/>
      </prompt>
    </play>
  </request>
</MediaServerControl>
```

Figure 7: Full Conference Audio Command - Play

A client can modify a leg by issuing an INFO on the dialog associated with the participant leg. For example, Figure 8 mutes a conference leg.

```
<?xml version="1.0" encoding="utf-8"?>
<MediaServerControl version="1.0">
  <request>
    <configure_leg mixmode="mute"/>
  </request>
</MediaServerControl>
```

Figure 8: Sample Change Leg Command

In Figure 7, we saw a request to play a prompt to the entire conference. The client can also request to play a prompt to an individual call leg. In that case, the MSCML request is issued within the SIP dialog of the desired conference participant.

Section 6 describes the interactive voice response (IVR) services offered by MSCML. If an IVR command arrives on the control channel, it takes effect on the whole conference. This is a mechanism for playing prompts to the entire conference (e.g., announcing new participants). If an IVR command arrives on an individual leg, it only affects that leg. This is a mechanism for interacting with users, such as the creation of "waiting rooms", allowing a user to mute themselves using key presses, allowing a moderator to out-dial, etc.

A participant leg MUST be configured with mixmode="parked" prior to the issuance of any IVR commands with prompt content ('prompturl' attribute or <prompt> element). Parking the leg isolates the participant's input and output media from the conference and allows use of those streams for playing and recording purposes. However, the mixmode has no effect if just digit collection or recording is desired. <playcollect> and <playrecord> requests without prompt content MAY be sent on participant legs without setting mixmode="parked".

5.6. Video Conferencing

MSCML-controlled advanced conferences, as well as RFC 4240 [2] controlled basic conferences, implicitly support video conferencing in the form of video switching. In video switching, the video stream of the loudest talker (with some hysteresis) is sent to all participants other than that talker. The loudest talker receives the video stream from the immediately prior loudest talker.

Media servers MUST ensure that participants receive video media compatible with their session. For example, a participant who has established an H.263 video stream will not receive video from another participant employing H.264 media. Media servers SHOULD implement video transcoding to minimize media incompatibilities between participants.

The media server MUST switch video streams only when it receives a refresh video frame. A refresh frame contains all the video information required to decode that frame (i.e., there is no dependency on data from previous video frames).

Refresh frames are large and generally sent infrequently to conserve network bandwidth. The media server MUST implement standard mechanisms to request that the new loudest talker's video encoder transmits a refresh frame to ensure that video can be switched quickly.

5.7. Conference Events

A client can subscribe for periodic active talker event reports that indicate which participants are included in the conference mix. As these are conference-level events, the subscription and notifications are sent on the Conference Control Leg.

Media servers MAY impose limits on the minimum interval for active talker reports for performance reasons. If the client request is below the imposed minimum, the media server SHOULD set the interval to the minimum value supported. To limit unnecessary notification traffic, the media server SHOULD NOT send a report if the active talker information for the conference has not changed during the reporting interval.

A request for an active talker report is in Figure 9. The active talker report enumerates the current call legs in the mix.

```
<?xml version="1.0" encoding="utf-8"?>
<MediaServerControl version="1.0">
  <request>
    <configure_conference>
      <subscribe>
        <events>
          <activetalkers report="yes" interval="60s"/>
        </events>
      </subscribe>
    </configure_conference>
  </request>
</MediaServerControl>
```

Figure 9: Active Talker Request

Event notifications are sent in SIP INFO messages. Figure 10 shows an example of a report.

```
<?xml version="1.0" encoding="utf-8"?>
<MediaServerControl version="1.0">
  <notification>
    <conference uniqueid="ab34h76z" numtalkers="47">
      <activetalkers>
        <talker callid="myhost4sn123"/>
        <talker callid="myhost2sn456"/>
        <talker callid="myhost12sn78"/>
      </activetalkers>
    </conference>
  </notification>
</MediaServerControl>
```

Figure 10: Active Talker Event Example

The value of the "callid" attribute in the <talker> element corresponds to the value of the SIP Call-ID header of the associated dialog. This enables the client to associate the active talker with a specific participant leg.

5.8. Conferencing with Personalized Mixes

MSCML enables clients to create personalized mixes through the <configure_team> element for scenarios where the standard mixmode settings do not provide sufficient control. The <configure_team> element is a child of <configure_leg>.

To create personalized mixes, the client has to identify the relationships among the participants. This is accomplished by manipulating two MSCML objects. These objects are:

1. The list of team members (<teammate> elements), set using <configure_team>

2. The mixmode attribute set through <configure_leg>

The media server uses the values of these objects to determine which audio inputs to combine for output to the participant. In a normal conference, each participant hears the conference mix minus their own input if they are part of the mixed output. The team list enables the client to specify other participants that the leg can hear in addition to the normal mixed output. Note that personalized mix settings apply only to audio media and do not affect video switching.

Team relationships are implicitly symmetric. If the client sets participant A as a team member of participant B, then the media server automatically sets participant B as a team member for A.

The id attribute set through <configure_leg> is used to identify the various participants. A unique ID MUST be assigned to each participant included in a personalized mix. The IDs used MUST be unique within the scope of the conference in which they appear.

By itself, the team list only defines those participants that the leg can hear. The mixmode attribute of each team member determines whether to include their audio input in the personalized mix. If the client sets the teammate's mixmode to private, then it is part of the mix. If the mixmode is set to any other value, it is not.

5.8.1. MSCML Elements and Attributes for Personalized Mixes

Control of personalized mixes rely on two major MSCML elements:

1. <configure_leg>, using the mixmode attribute setting mixmode="private"

2. <configure_team>

The <configure_team> element allows the user to make the participants members of a team within a specific conference. It is a child of the <configure_leg> parent element.

The client sends the <configure_team> element in a <configure_leg> request in either a SIP INVITE or SIP INFO.

- In an INVITE, to join a participant whose properties differ from the properties established for the conference as a whole.

- In an INFO, to change the properties for an existing leg.

The two attributes of the configure_team element are "id" and "action". The id attribute MUST contain the unique ID of the leg being modified, as set in the original <configure_leg> request. The action attribute can take on the values "add", "delete", "query", and "set". The default value is "query". This attribute allows the user to modify the team list. Table 1 describes the actions that can be performed on the team list.

Action	Description
add	Adds a teammate to the mix.
delete	Deletes a teammate from the mix.
query	Returns the teammate list to the requestor. This is the default value.
set	Creates a team list when followed by <teammate id="n"> and also removes all the teammates from the team list for example, when the creator (originator) of the team list on that specific conference leg wants to remove all of the teammates from the team. If the set operation removes all teammates from a participant, that participant hears the full conference mix.

Table 1: Configure Team Actions

5.8.2. Example Usage of Personalized Mixes

A common use of personalized mixing is to support coaching of one participant by another. The coaching scenario includes three participants:

1. The Supervisor, who coaches the agent.
2. The Agent, who interacts with the customer.
3. The Customer, who interacts with the agent.

Table 2 illustrates the details of the coached conference topology.

Participant	ID	Team Members	Mixmode	Hears
Supervisor	supervisor	Agent	Private	customer + agent
Agent	agent	Supervisor	Full	customer + supervisor
Customer	customer	none	Full	agent

Table 2: Coached Conference Example

To create this topology, the client performs the following actions:

1. The client joins each leg to the conference, being certain to include a unique ID in the <configure_leg> request. The leg ID needs to be unique only within the scope of the conference to which it belongs.

2. The client configures the teammate list and mixmode of each participant, as required.

Both actions (steps 1 and 2) may be combined in a single MSCML request. The following sections detail these actions and their corresponding MSCML payloads.

5.8.2.1. Create the Conference

Before joining any participants, the client must create the conference by sending a SIP INVITE that contains an MSCML <configure_conference> request with a unique conference identifier.

5.8.2.2. Joining and Configuring the Coach

Join the coach leg to the conference and configure its desired properties by sending a SIP INVITE containing a <configure_leg> request. The <configure_leg> element sets the leg's unique ID to supervisor and its mixmode to private.

The corresponding MSCML request is as follows.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <configure_leg id="supervisor" mixmode="private"/>
  </request>
</MediaServerControl>
```

Figure 11: Join Coach Request

Note that the client cannot configure the teammate list for the coach yet, as there are no other participants in the conference. One must join a participant to the conference before one can add it as a teammate for another leg.

5.8.2.3. Joining and Configuring the Agent

Join the agent leg to the conference and configure its desired properties by sending a SIP INVITE containing a <configure_leg> request. The <configure_leg> element sets the leg's unique ID to "agent" and sets the supervisor as a team member of the agent. Because team member relationships are symmetric, this action also adds the agent as a team member for the coach.

The corresponding MSCML request is as follows.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
 <request>
    <configure_leg id="agent">
      <configure_team action="set">
        <teammate id="supervisor"/>
      </configure_team>
    </configure_leg>
  </request>
</MediaServerControl>
```

Figure 12: Join Agent Request

Because the desired mixmode for this leg is full, which is the default value, there is no need to set it explicitly.

5.8.2.4. Joining and Configuring the Client

Join the client leg to the conference and configure its desired properties by sending a SIP INVITE containing a <configure_leg> request. The <configure_leg> element simply sets the leg's unique ID to "customer". The media server does not need further configuration because the desired mixmode, full, is the default and the customer has no team members.

The corresponding MSCML request is as follows.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <configure_leg id="customer"/>
  </request>
</MediaServerControl>
```

Figure 13: Join Client Request

Strictly speaking, it is not a requirement that the client give the customer leg a unique ID because it will not be a team member. However, when using coached conferencing, we RECOMMEND that one assign a unique ID to each leg in the initial INVITE request. Assigning a unique ID eliminates the need to set it later by sending a SIP INFO if one later desires personalized mixing for the customer leg.

The conference is now in the desired configuration, shown previously in Table 2.

6. Interactive Voice Response (IVR)

In the IVR model, the media server acts as a media-processing proxy for the UAC. This is particularly useful when the UAC is a media gateway or other device with limited media processing capability.

The typical use case for MSCML is when there is an application server that is the MSCML client. The client can use the SIP Service URI concept (RFC 3087) to initiate a service. The client then uses RFC 4240 [2] to initiate a MSCML session on a media server. These relationships are shown in Figure 14.

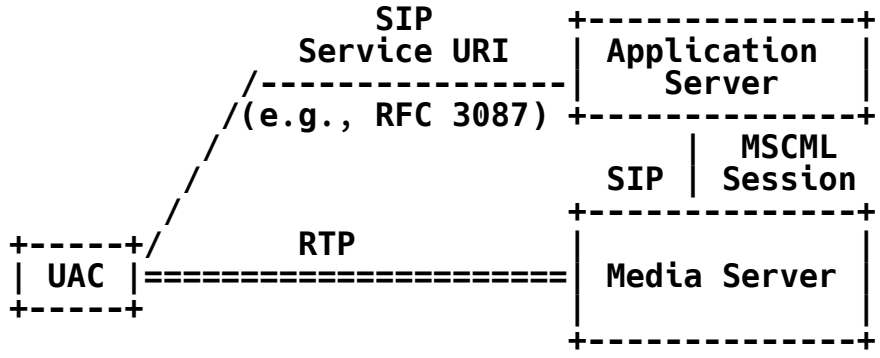

Figure 14: IVR Model

The IVR service supports basic Interactive Voice Response functions, playing announcements, collecting DTMF digits, and recording, based on Media Server Control Markup Language (MSCML) directives added to the message body of a SIP request. The major MSCML IVR requests are <play>, <playcollect>, and <playrecord>.

Multifunction media servers MUST use the URI conventions described in RFC 4240 [2]. The service indicator for MSCML IVR MUST be set to "ivr", as shown in the following example:

sip:ivr@ms.example.net

The VoiceXML IVR service indicator is "dialog". This service indicator MUST NOT be used for any other interactive voice response control mechanism.

The media server MUST accept MSCML IVR payloads in INFO requests and MUST NOT accept MSCML IVR payloads in the initial or subsequent INVITEs. The INFO method reduces certain timing issues that occur with INVITEs and requires less processing on both the client and media server.

The media server notifies the client that the command has completed through a <response> message containing final status information and associated data such as collected DTMF digits.

The media server does not queue IVR requests. If the media server receives a new IVR request while another is in progress, the media server stops the first operation and it carries out the new request. The media server generates a <response> message for the first request and returns any data collected up to that point. If a client wishes to stop a request in progress but does not wish to initiate another operation, it issues a <stop> request. This also causes the media server to generate a <response> message.

The media server treats a SIP re-INVITE that modifies the established SDP parameters as an implicit <stop> request. Examples of such SDP modifications include receiving hold SDP or removing an audio or video stream. When this occurs, the media server immediately terminates the running <play>, <playcollect>, or <playrecord> request and sends a <response> indicating "reason=stopped".

6.1. Specifying Prompt Content

The MSCML IVR requests support two methods of specifying content to be delivered to the user. These are the <prompt> element and the promptur attribute. Clients MUST NOT utilize both methods in a single IVR request. Clients SHOULD use the more flexible <prompt> mechanism. Use of the promptur attribute is deprecated and may not be supported in future MSCML versions.

6.1.1. Use of the Prompt Element

The <prompt> element MAY be included in the body of a <play>, <playcollect>, or <playrecord> request to specify a prompt sequence to be delivered to the caller. The prompt sequence consists of one or more references to physical content files, spoken variables, or dynamic URLs that return a sub-sequence of files or variables. In addition, the <prompt> element has several attributes that control playback of the included content. These are described in the list below.

Attributes of <prompt>:

- baseurl - optional, no default value: For notational convenience, as well as reducing the MSCML payload size, the "baseurl" attribute is used to specify a base URL that is prepended to any other URLs in the sequence that are not fully qualified.

- delay - optional, default value "0": The "delay" attribute to the prompt element specifies the time to pause between repetitions of the <prompt> sequence. It has no effect on the first iteration of the sequence. Expressed as a time value (Section 4.2.1) from 0 onwards.

- duration - optional, default value "infinite": The "duration" attribute to the prompt element controls the maximum amount of time that may elapse while the media server repeats the sequence. This allows the client to set an upper bound on the length of play. Expressed as a time value (Section 4.2.1) from 1ms onwards or the strings "immediate" and "infinite". "Immediate" directs the media server to end play immediately, whereas "infinite" indicates that the media server imposes no limit.

- gain - optional, default value "0": Sets the absolute gain to be applied to the content contained in <prompt>. The value of this attribute is specified in units of dB. The media server MAY silently cap values that exceed the gain limits imposed by the platform. The level reverts back to its original value when playback of the content contained in <prompt> has been completed.

- gaindelta - optional, default value "0": Sets the relative gain to be applied to the content contained in <prompt>. The value of this attribute is specified in units of dB. The media server MAY silently cap values which exceed the gain limits imposed by the platform. The level reverts back to its original value when playback of the content contained in <prompt> has been completed.

- rate - optional, default value "0": Specifies the absolute playback rate of the content relative to normal as either a positive percentage (faster) or a negative percentage (slower). Any value that attempts to set the rate above the maximum allowed or below the minimum allowed silently sets the rate to the maximum or minimum. The rate reverts back to its original value when playback of the content contained in <prompt> has been completed.

- ratedelta - optional, default value "0": Specifies the playback rate of the content relative to it's current rate as either a positive percentage (faster) or negative percentage (slower). Any value that attempts to set the rate above the maximum allowed or below the minimum allowed silently sets the rate to the maximum or minimum. The rate reverts back to its original value when playback of the content contained in <prompt> has completed.

- locale - optional, no default value: Specifies the language and country variant used for resolving spoken variables. The language is defined as a two-letter code per ISO 639. The country variant is also defined as a two-letter code per ISO 3166. These codes are concatenated with a single underscore (%x5F) character.

- offset - optional, default value "0": A time value (Section 4.2.1) which specifies the time from the beginning of the sequence at which play is to begin. Offset only applies to the first repetition; subsequent repetitions begin play at offset 0. Allowable values are positive time values from 0 onwards. When the sequence consists of multiple content files, the offset may select any point in the sequence. If the offset value is greater than the total time of the sequence, it will "wrap" to the beginning and continue from there until the media server reaches the specified offset.

- repeat - optional, default value "1": The "repeat" attribute to the prompt element controls the number of times the media server plays the sequence in the <prompt> element. Allowable values are integers from 0 on and the string "infinite", which indicates that repetition should occur indefinitely. For example, "repeat=2" means that the sequence will be played twice, and "repeat=0", which is allowed, means that the sequence is not played.

- stoponerror - optional, default value "no": Controls media server handling and reporting of errors encountered when retrieving remote content. If set to "yes", content play will end if a fetch error occurs, and the response will contain details regarding the failure. If set to "no", the media server will silently move on to the next URL in the sequence if a fetch failure occurs.

Clients MUST NOT include both 'gain' and 'gaindelta' attributes within a single <prompt> element.

When the client explicitly controls the output gain on a conference leg, as described in Section 5.3, the 'gain' and 'gaindelta' attributes SHOULD interact with the conference leg output gain settings in the following manner.

- Conference leg output gain set to <fixed>: The operation of the 'gain' and 'gaindelta' attributes are unchanged. However, the baseline gain value before any playback changes are applied is the value specified for the conference leg.

- Conference leg output gain set to <auto>: When playback gain controls are used, the automatic gain control settings for the leg are suspended for the duration of the playback operation. The

operation of the 'gain' and 'gaindelta' attributes are unchanged. The automatic gain control settings are reinstated when playback has finished.

Media servers SHOULD support rate controls for content. However, media servers MAY silently ignore rate change requests if content limitations do not allow the request to be honored. Clients MUST NOT include both 'rate' and 'ratedelta' attributes within a single <prompt> element.

Figure 16 shows a sample prompt block.

```
<prompt stoponerror="yes"
  baseurl="file:////var/mediaserver/prompts/"
  locale="en_US" offset="0" gain="0" rate="0"
  delay="0" duration="infinite" repeat="1">
  <audio url="num_dialed.raw" encoding="ulaw"/>
  <variable type="dig" subtype="ndn" value="3014170700"/>
  <audio url="num_invalid.wav"/>
  <audio url="please_check.wav"/>
</prompt>
```

Figure 16: Prompt Block Example

6.1.1.1. <audio> and <variable> Elements

Clients compose prompt sequences using the <audio> and <variable> elements. An <audio> element MAY refer to content that contains audio, video, or both; the generic name is preserved for backwards compatibility. The <audio> element has the attributes described in the list below.

Attributes of <audio>:

- url - required, no default value: The URL of the content to be retrieved and played. The target may be a local or remote (NFS) "file://" scheme URL or an "http://" or "https://" scheme URL. If the URL is not fully qualified and a "baseurl" attribute was set, the value of the "baseurl" attribute will be prepended to this value to generate the target URL.

- encoding - optional, default value "ulaw": Specifies the content encoding for file formats that are not self-describing (e.g., .WAV). Allowable values are "ulaw", "alaw", and "msgsm". This attribute only affects "file://" scheme URLs.

- gain - optional, default value "0": Sets the absolute gain to be applied to the content URL. The value of this attribute is

specified in units of dB. The media server MAY silently cap values that exceed the gain limits imposed by the platform. The level reverts back to its original value when playback of the content URL has been completed.

- gaindelta - optional, default value "0": Sets the relative gain to be applied to the content URL. The value of this attribute is specified in units of dB. The media server MAY silently cap values that exceed the gain limits imposed by the platform. The level reverts back to its original value when playback of the content URL has been completed.

- rate - optional, default value "0": Specifies the absolute playback rate of the content relative to normal as either a positive percentage (faster) or a negative percentage (slower). Any value that attempts to set the rate above the maximum allowed or below the minimum allowed silently sets the rate to the maximum or minimum. The rate reverts back to its original value when playback of the content URL has been completed.

- ratedelta - optional, default value "0": Specifies the playback rate of the content relative to it's current rate as either a positive percentage (faster) or a negative percentage (slower). Any value that attempts to set the rate above the maximum allowed or below the minimum allowed silently sets the rate to the maximum or minimum. The rate reverts back to its original value when playback of the content URL has been completed.

Clients MUST NOT include both 'gain' and 'gaindelta' attributes within a single <audio> element.

When the client explicitly controls the output gain on a conference leg, as described in Section 5.3, the 'gain' and 'gaindelta' attributes SHOULD interact with the conference leg output gain settings in the following manner.

- Conference leg output gain set to <fixed>: The operation of the 'gain' and 'gaindelta' attributes are unchanged. However, the baseline gain value before any playback changes are applied is the value specified for the conference leg.

- Conference leg output gain set to <auto>: When playback gain controls are used, the automatic gain control settings for the leg are suspended for the duration of the playback operation. The operation of the 'gain' and 'gaindelta' attributes are unchanged. The automatic gain control settings are reinstated when playback has finished.

Media servers SHOULD support rate controls for content. However, media servers MAY silently ignore rate change requests if content limitations do not allow the request to be honored. Clients MUST NOT include both 'rate' and 'ratedelta' attributes within a single <audio> element.

Media servers MUST support local and remote (NFS) "file://" scheme URLs and "http://" and "https://" scheme URLs for content retrieval.

NOTE: The provisioning of NFS mount points and their mapping to the "file://" schema is purely a local matter at the media server.

MSCML also supports "http://" and "https://" scheme URLS that return a list of physical URLs using the "text/uri-list" MIME type. This facility provides flexibility for applications to dynamically generate prompt sequences at execution time and enables separation of this function from the client and media server.

Spoken variables are specified using the <variable> element. This element has the attributes described in the list below. MSCML's spoken variables are based on those described in Audio Server Protocol [17].

Attributes of <variable>:

- type - required, no default value: Specifies the major type format of the spoken variable to be played. Allowable values are "dat" (date), "dig" (digit), "dur" (duration), "mth" (month), "mny" (money), "num" (number), "sil" (silence), "str" (string), "tme" (time), and "wkd" (weekday).

- subtype - optional, no default value: Specifies the minor type format of the spoken variable to be played. Allowable values depend on the value of the corresponding "type" attribute. Possible values are "mdy", "ymd", and "dmy" for dates, "t12" and "t24" for times, "gen", "ndn", "crd", and "ord" for digits, and "USD" for money.

- value - required, no default value: A string that will be interpreted based on the formatting information specified in the "type" and "subtype" attributes and the "locale" attribute of the parent <prompt> element to render the spoken variable.

If the "locale" attribute was not specified in <prompt>, the media server SHOULD make a selection based on platform configuration. If the precise "locale" requested cannot be honored, the media server SHOULD select the closest match based on the available content.

IVR applications normally require specialized prompt content that is authored by the application provider. To deliver a quality user interaction, the specialized prompts and spoken variables must be generated by the same speaker. Since the media server inherently supports multiple simultaneous applications, it is extremely difficult to provision all the necessary application prompts and matching spoken variable content locally on the media server. Therefore, we STRONGLY RECOMMEND that clients employ the dynamic URL mechanism described earlier to generate spoken variables using an external web server that returns "text/uri-list" content.

6.2. Multimedia Processing for IVR

MSCML IVR requests implicitly support multimedia content. Multimedia capabilities are controlled by the audio and video media negotiated for the dialog and the content specified by the client for play and record operations. If the content specified for delivery contains both audio and video tracks and the dialog has audio and video streams, both tracks are streamed to the caller. Likewise, if the dialog has both audio and video streams and the content format specified supports both (e.g., .3gp files) the media server records both streams to the file.

If there is a mismatch between the real time media and specified content, the media server MUST play or record the appropriate content tracks rather than failing the request. For example, if the client has requested playback of content with audio and video tracks but only audio media has been established for the dialog, the media server should play the audio track. If the dialog has both audio and video media but the content is audio-only, the media server MAY stream a pre-provisioned video track to the caller. Media servers SHOULD implement video transcoding functions to minimize incompatibilities between real time media and content.

The media server MUST begin recording video media only when it receives a refresh video frame. A refresh frame contains all the video information required to decode that frame (i.e., there is no dependency on data from previous video frames). Refresh frames are large and generally sent infrequently to conserve network bandwidth. The media server MUST implement standard mechanisms to request that the caller (video encoder) transmit a refresh frame to ensure video recording begins quickly. The media server MUST begin recording the audio track immediately while waiting to receive the video refresh frame.

6.3. Playing Announcements <play>

The client issues a <play> request to play an announcement without interruption and with no digit collection. One use, for example, is to announce the name of a new participant to the entire conference. The <play> request has the attributes described in the list below.

Attributes of <play>:

- id - optional, no default value: Specifies a client-defined ID for purposes of matching requests and responses.

- offset - optional, default value "0": Specifies the time from the beginning of the URL specified in the 'prompturl' attribute at which play will begin. Expressed as a time value (Section 4.2.1) from 0 onwards. If the offset value is greater than the total time of the content, it will "wrap" to the beginning and continue from there until the media server reaches the specified offset. NOTE: Use of this attribute is deprecated.

- promptencoding - optional, no default value: Specifies the content encoding for file formats that are not self-describing (e.g., .WAV). Allowable values are "ulaw", "alaw", and "msgsm". This attribute only affects "file://" scheme URLs. NOTE: Use of this attribute is deprecated.

- prompturl - optional, no default value: The URL of the content to be retrieved and played. The target may be a local or remote (NFS) "file://" scheme URL or an "http://" or "https://" scheme URL. NOTE: Use of this attribute is deprecated.

The <play> request has one child element defined, <prompt>. Use of <prompt> is described in Section 6.1.1.

The client MUST NOT use both the <prompt> element and "prompturl" attribute in a single request. As previously discussed, the "prompturl" attribute is supported for backwards compatibility with older MSCML applications, but its use is deprecated. The more flexible <prompt> element SHOULD be used instead.

The following play request (Figure 17) example shows the delivery of a complex prompt sequence consisting of content accessed via NFS and spoken variables.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <play id="332985001">
      <prompt stoponerror="yes"
        baseurl="file:////var/mediaserver/prompts/"
        locale="en_US" offset="0" gain="0" rate="0"
        delay="0" duration="infinite" repeat="1">
        <audio url="num_dialed.raw" encoding="ulaw"/>
        <variable type="dig" subtype="ndn" value="3014170700"/>
        <audio url="num_invalid.wav"/>
        <audio url="please_check.wav"/>
      </prompt>
    </play>
  </request>
</MediaServerControl>
```

Figure 17: <Play> Request Example

When the announcement has finished playing, the media server sends a <response> payload to the client in a SIP INFO message. Details regarding the format of <play> responses are provided in Section 10.4.

6.4. Prompt and Collect <playcollect>

The client issues a <playcollect> request to play an announcement (optional) and collect digits. The <playcollect> request is executed in two phases, prompt and collect. If the client specifies prompt content to be played, using the <prompt> element or prompturl attribute, the media server plays the content before starting the collection phase. If no prompt content is specified, the collect phase begins immediately.

The basic attributes of <playcollect> are the same as those of <play>, which were described in Section 6.3. In addition to these basic attributes, <playcollect> defines others which control digit buffering and barge-in behavior, collection timers, special purpose DTMF key functions, and logging of user DTMF input. Each functional category and its attributes are described below.

6.4.1. Control of Digit Buffering and Barge-In

Whenever the media server is processing a call that specifies an MSCML service (i.e., "conf" and "ivr"), the media server continuously looks for DTMF digits and places them in a quarantine buffer. The quarantine buffer is examined when a <playcollect> request is received. The media server compares any previously buffered digits for barge-in, and to look for matches with DTMF grammars or special purpose keys. This provides the type-ahead behavior for menu traversal and other types of IVR interactions.

Attributes for Control of Digit Buffering and Barge-In:

- cleardigits - optional, default value "no": Specifies whether previous user input should be considered or ignored for barge-in purposes and DTMF matching. When it is set to "yes", any previously buffered digits are removed, so prior user input is ignored. If it is set to "no", previously buffered digits will be considered. If "cleardigits" is set to "no" and barge-in is enabled, previously buffered digits will immediately terminate the prompt phase. In this case, the prompt is not played, and digit collection begins immediately.

- barge - optional, default value "yes": Specifies whether user input will barge the prompt and force transition to the collect phase. When it is set to "yes", a DTMF input will barge the prompt. When it is set to "no", the prompt phase cannot be barged, and any user input during the prompt is placed in the quarantine buffer for inspection during the collect phase. Note that if the "barge" attribute is set to "no", the "cleardigits" attribute implicitly has a value of "yes". This ensures that the media server does not leave DTMF input that occurred prior to the current collection in the quarantine buffer after the request is completed.

6.4.2. Mapping DTMF Keys to Special Functions

The client can define mappings between DTMF digits and special functions. The media server invokes the special function if the associated DTMF digit is detected. MSCML has two attributes that define mappings that affect termination of the collect phase. These attributes are described in the list below.

DTMF Key Mappings for <playcollect>:

- escapekey - optional, default value "*": Specifies a DTMF key that indicates that the user wishes to terminate the current operation without saving any input collected to that point. Detection of the mapped DTMF key terminates the request immediately and generates a response.

- returnkey - optional, default value "#": Specifies a DTMF key that indicates that the user has completed input and wants to return all collected digits to the client. When the media server detects the returnkey, it immediately terminates collection and returns the collected digits to the client in the <response> message.

MSCML defines three additional mappings to enable video cassette recorder (VCR) type controls while playing a prompt sequence. Media servers SHOULD support VCR controls. However, if the media server does not support VCR controls, it MUST silently ignore DTMF inputs mapped to VCR functions and complete the <playcollect> request. The VCR control attributes are described in the list below.

Attributes for VCR Controls:

- skipinterval - optional, default value "6s": The "skipinterval" attribute indicates how far the media server should skip backwards or forwards when the rewind key (rwkey) or fast forward key (ffkey) is pressed, specified as a time value (Section 4.2.1).

- ffkey - optional, no default value: The "ffkey" attribute maps a DTMF key to a fast forward operation equal to the value of the "skipinterval" attribute.

- rwkey - optional, no default value: The "rwkey" attribute maps a DTMF key to a rewind action equal to the value of the "skipinterval" attribute.

Clients MUST NOT map the same DTMF digit to both the "rwkey" and "ffkey" attributes in a single <playcollect> request.

VCR control operations are bounded by the beginning and end of the prompt sequence. A rewind action that moves the offset before the beginning of the sequence results in playback starting at the beginning of the sequence (i.e., offset=0). A fast forward action that moves the offset past the end of the sequence results in the media server's treating the sequence as complete.

6.4.3. Collection Timers

MSCML defines several timer attributes that control how long the media server waits for digits in the input sequence. All timer settings are time values (Section 4.2.1). The list below describes these attributes and their use.

Collection Timer Attributes:

- firstdigittimer - optional, default value "5000ms": Specifies how long the media server waits for the initial DTMF input before terminating the collection. Expressed as a time value (Section 4.2.1) from 1ms onwards or the strings "immediate" and "infinite." The value "immediate" indicates that the timer should fire immediately whereas "infinite" indicates that the timer will never fire.

- interdigittimer - optional, default value "2000ms": Specifies how long the media server waits between DTMF inputs. Expressed as a time value (Section 4.2.1) from 1ms onwards or the strings "immediate" and "infinite." The value "immediate" indicates that the timer should fire immediately, whereas "infinite" indicates that the timer will never fire.

- extradigittimer - optional, default value "1000ms": Specifies how long the media server waits for additional user input after the specified number of digits has been collected. Expressed as a time value (Section 4.2.1) from 1ms onwards or the strings "immediate" and "infinite." The value "immediate" indicates that the timer should fire immediately, whereas "infinite" indicates that the timer will never fire.

- interdigitcriticaltimer - optional, defaults to the value of the interdigittimer attribute: Specifies how long the media server waits after a grammar has been matched for a subsequent digit that may cause a longer match. Expressed as a time value (Section 4.2.1) from 1ms onwards or the strings "immediate" and "infinite." The value "immediate" results in "shortest match first" behavior, whereas "infinite" means to wait indefinitely for additional input. If not explicitly specified otherwise, this attribute is set to the value of the 'interdigittimer' attribute.

The extradigittimer setting enables the "returnkey" input to be associated with the current collection. For example, if maxdigits is set to 3 and returnkey is set to #, the user may enter either "x#", "xx#", or "xxx#", where x represents a DTMF digit.

If the media server detects the "returnkey" pattern during the "extradigit" interval, the media server returns the collected digits to the client and removes the "returnkey" from the digit buffer.

If this were not the case, the example would return "xxx" to the client and leave the terminating "#" in the digit buffer. At the next <playcollect> request, the media server would process the '#'. This might result in the termination of the following prompt, which is clearly not what the user intended.

The extradigittimer has no effect unless returnkey has been set.

6.4.4. Logging Caller DTMF Input

Standard SIP mechanisms, such as those discussed in Security Considerations (Section 14), protect MSCML protocol exchanges and the information they contain. These protections do not apply to data captured in media server log files. In general, media server logging is platform specific and therefore is not covered by this specification. However, one aspect of logging, the capture of sensitive information (such as personal identification numbers or credit card numbers), is relevant. The media server has no means to determine whether the DTMF input it receives may be sensitive, as that is in the purview of the client. Recognizing this, MSCML includes a per- request mechanism to suppress logging of captured DTMF to be enabled by clients as needed.

The "maskdigits" attribute controls whether detected DTMF digits appear in the log output. Clients use this attribute when the media server collects sensitive information that should not be accessible through the log files.

Maskdigits Attribute:

- maskdigits - optional, default value "no": Controls whether user DTMF inputs are captured in media server log files. The possible values for this attribute are "yes" and "no".

6.4.5. Specifying DTMF Grammars

MSCML supports four methods for specifying DTMF grammars: the "maxdigits" attribute, which provides a simple mechanism for collecting any number of digits up to the maximum, regular expressions, MGCP [5] digit maps, and H.248.1 [6] digit maps. A media server MUST support the maxdigits and regular expression methods for specifying DTMF grammars and SHOULD support MGCP and H.248.1 methods. A client MUST NOT mix DTMF grammar types in a single <playcollect> request.

Following is a description of the "maxdigits" attribute.

Maxdigits Attribute:

- maxdigits - optional, no default value: Specifies the maximum number of DTMF digits to be collected.

The <pattern> element specifies a digit pattern or patterns for the media server to look for. This element may contain three different child elements that specify the type of DTMF grammar used in the expression. The <pattern> element has no attributes.

<regex> Use regular expressions to define DTMF patterns to match. The complete regular expression syntax used in MSCML is described in Appendix A.

<mgcpdigitmap> Use digit maps as specified in MGCP [5].

<megacodigitmap> Use digit maps as specified in H.248.1 [6].

At least one <regex> element MUST be present in <pattern> when regex grammars are used. Multiple <regex> elements MAY be present. When <mgcpdigitmap> or <megacodigitmap> grammars are used, <pattern> MUST contain only one grammar element.

The DTMF grammar elements <regex>, <mgcpdigitmap>, and <megacodigitmap> have the attributes described in the list below.

Attributes of DTMF Grammar Elements:

- value - required, no default value: Specifies a string representing a DTMF grammar matching the parent element type (e.g., regex). Regex values represent a single DTMF grammar. MGCP and MEGACO digit maps allow multiple grammars to be described in a single string.

- name - optional, no default value: Associates a client defined name for the grammar that is sent back in the <playcollect> response. This attribute is most useful with regex type grammars as each grammar element can have a unique name.

6.4.6. Playcollect Response

When the <playcollect> has finished, the media server sends a <response> payload to the client in a SIP INFO message.

Details of the <playcollect> response are described in Section 10.5.

6.4.7. Playcollect Example

The following <playcollect> request (Figure 18) example depicts use of the "maxdigits" attribute to control digit collection.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <playcollect id="332986004" maxdigits="6" firstdigittimer="10000"
      interdigittimer="5000" extradigittimer="1000"
      interdigitcriticaltimer="1000" returnkey="#" escapekey="*"
      cleardigits="no" barge="yes" maskdigits="no">
      <prompt baseurl="http://www.example.com/prompts/">
        <audio url="generic/en_US/enter_pin.wav"/>
      </prompt>
    </playcollect>
  </request>
</MediaServerControl>
```

Figure 18: <Playcollect> Request Example Using the Maxdigits Attribute

6.5. Prompt and Record <playrecord>

The <playrecord> request directs the media server to convert and possibly to transcode the RTP payloads it receives and store them to the specified URL using the requested content codec(s) and file format. This request proceeds in two phases; prompt and record.

The <playrecord> request shares the basic attributes of <play> and <playcollect> as described in Section 6.3. MSCML also defines other attributes that control the behavior of the prompt and recording phases. These phases and the attributes that control them are described in the text and tables below.

6.5.1. Prompt Phase

The presence or absence of a "prompturl" attribute or child <prompt> element controls whether a prompt is played before recording begins. As previously noted, use of the "prompturl" attribute is deprecated, and clients SHOULD use <prompt> instead.

When the client requests that the media server prompt the caller before recording audio, <playrecord> has two stages. The first is equivalent to a <playcollect> operation. The client may set the prompt phase to be interruptible by DTMF input (barge) and may specify an escape key that will terminate the <playrecord> request before the recording phase begins.

The list below describes the attributes of <playrecord> that specify the behavior of the prompt phase of the request.

Playrecord Attributes for the Prompt Phase:

- barge - optional, default value "yes": Specifies whether user input will barge the prompt and force transition to the record phase. When it is set to "yes", a DTMF input will barge the prompt. When it is set to "no", the prompt phase cannot be barged, and any user input during the prompt is placed in the quarantine buffer for inspection during the collect phase. Note that if the "barge" attribute is set to "no", the "cleardigits" attribute implicitly has a value of "yes". This ensures that the media server does not leave DTMF input that occurred prior to the current collection in the quarantine buffer after the request completes.

- cleardigits - optional, default value "no": Specifies whether previous user input should be considered or ignored for barge-in purposes. When it is set to "yes", any previously buffered digits are removed, so prior user input is ignored. If it is set to "no", previously buffered digits will be considered. If "cleardigits" is set to "no" and barge-in is enabled, previously buffered digits will terminate the prompt phase immediately. In this case, the prompt is not played, and recording begins immediately.

- escapekey - optional, default value "*": Specifies a DTMF key that indicates the user wishes to terminate the current operation without saving any input recorded to that point. Detection of the mapped DTMF key terminates the request immediately and generates a response.

Detection of the escape key generates a response message, and the operation returns immediately. If the user presses any other keys and if the prompt is interruptible (barge="yes"), then the play stops immediately, and the recording phase begins.

6.5.2. Record Phase

If the request proceeds to the recording phase, the media server discards any digits from the collect phase from the quarantine buffer to eliminate unintended termination of the recording. The following attributes control recording behavior.

Playrecord Attributes for the Record Phase:

- recurl - required, no default value: Specifies the target URL for the recorded content.

- recencoding - optional, default value "ulaw": Specifies the encoding of the recorded content if it cannot be inferred from the recurl. Possible values are "ulaw", "alaw", and "msgsm."

- mode - optional, default value "overwrite": Specifies whether the recording should overwrite or be appended to the target URL. Allowable values are "overwrite" and "append."

- duration - optional, default value "infinite": Specifies the maximum allowable duration for the recording. Expressed as a time value (Section 4.2.1) from 1 onwards or the strings "immediate" and "infinite." The value "immediate" indicates that recording will end immediately, whereas "infinite" indicates recording should continue indefinitely. If the maximum duration is reached, the <playrecord> request will terminate and generate a response.

- beep - optional, default value "yes": Specifies whether a beep should be played to the caller immediately prior to the start of the recording phase. Allowable values are "yes" and "no."

- initsilence - optional, default value "3000ms": Specifies how long to wait for initial speech input before terminating (canceling) the recording. Expressed as a time value (Section 4.2.1) from 1ms onwards or the strings "immediate" and "infinite." The value "immediate" indicates that the timer should fire immediately, whereas "infinite" directs the media server to wait indefinitely.

- endsilence - optional, default value "4000ms": Specifies how long the media server waits after speech has ended to stop the recording. Expressed as a time value (Section 4.2.1) from 1ms onwards or the strings "immediate" and "infinite." When set to "infinite", the recording will continue indefinitely after speech has ended and will only terminate due to a DTMF keypress or because the input has reached the maximum desired duration.

- recstopmask - optional, default value "0123456789ABCD#*": Specifies a list of individual DTMF characters that, if detected, will cause the recording to be terminated. To ensure that the input of a specific key does not cause the recording to stop, remove the DTMF key from the list.

Media servers MUST support local and remote (NFS) "file://" scheme URLs in the "recurl" attribute. MSCML supports "http://" and "https://" scheme URLs indirectly through the <managecontent> (Section 8) request.

The media server buffers and returns any digits collected in the prompt phase, with the exception of those contained in the "recstopmask" attribute, in the response.

The media server compares digits detected during the recording phase to the digits specified in the "recstopmask" to determine whether they indicate a recording termination request.

The media server ignores digits not present in the recstopmask and passes them into the recording. If DTMF input terminates the recording, the media server returns the collected digit to the client in the <response>.

Once recording has begun, the media server writes the received media to the specified recurl URL no matter what DTMF events the media server detects. It is the responsibility of the client to examine the DTMF input returned in the <response> message to determine whether the audio file should be saved or deleted and, potentially, re-recorded.

If the endsilence timer expires, the media server trims the end of the recorded audio by an amount equal to the value of the "endsilence" attribute.

When the recording is finished, the media server generates a <response> message and sends it to the client in a SIP INFO message. Details of the <playrecord> response are described in Section 10.6.

6.5.3. Playrecord Example

The recording example (Figure 19) plays a prompt and records it to the destination specified in the "recurl" attribute encoded as MS-GSM in wave format.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <playrecord id="5556123"
      recurl="file:////nfs.example.com/rec/name.wav"
      recencoding="msgsm"
      initsilence="5000" endsilence="3000" duration="30000"
      barge="yes"
      beep="yes"
      mode="overwrite"
      cleardigits="no"
      escapekey="*"
      recstopmask="0123456789#*">
      <prompt>
        <audio url="http://www.example.com/prompts/recordname.wav"/>
      </prompt>
    </playrecord>
  </request>
</MediaServerControl>
```

Figure 19: Recording Example

6.6. Stop Request <stop>

The client issues a <stop> request when the objective is to stop a request in progress and not to initiate another operation. This request generates a <response> message from the media server.

The only attribute is id, which is optional.

The client-defined request id correlates the asynchronous response with its original request and echoes back to the client in the media server's response.

The following MSCML payload (Figure 20) depicts an example <stop> request.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <stop id="4578903"/>
  </request>
</MediaServerControl>
```

Figure 20: Stop Example

The format of a response to a <stop> request is detailed in Section 10.2.

As discussed previously, the media server treats a SIP re-INVITE that modifies the established SDP parameters as an implicit <stop> request. Examples of such SDP modifications include receiving hold SDP or removing an audio or video stream. When this occurs, the media server immediately terminates the running <play>, <playcollect>, or <playrecord> request and sends a <response> indicating "reason=stopped".

7. Call Leg Events

MSCML defines event notifications that are scoped to a specific SIP dialog or call leg. These events allow a client to be notified of individual, asynchronous DTMF keypresses, as well as of various call progress signals. The subscription, event detection, and notifications for call leg events occur in the same SIP dialog. This is different from the conference level active talker events described earlier. The subscription and notifications for active talker events occur on the conference control leg, but the actual event detection occurs on one or more participant legs.

Subscriptions for call leg events are made by sending an MSCML <configure_leg> request on the desired SIP dialog. Call leg events may be used with the MSCML conferencing or IVR services. When used with the IVR service, the <configure_leg> request SHOULD NOT include any conference-related attributes. The media server MUST ignore these if present. Call leg event subscriptions MUST NOT be made on the conference control leg, since it has no actual RTP media to process for event detection. The media server MUST reject a <configure_leg> request sent on the conference control leg.

The <configure_leg> request contains the child elements <subscribe> and <events>. The <events> element may contain two child elements that control subscriptions to call leg events. These are <keypress> and <signal>. A <configure_leg> request MUST contain at most one <keypress> element but MAY contain multiple <signal> elements that request notification of different call progress events.

7.1. Keypress Events

Keypress events are used when the client wishes to receive notifications of individual DTMF events that are not tied to a specific <playcollect> request. One use of this facility is to monitor conference legs for DTMF inputs that require application intervention; for example, to notify the moderator that the caller wishes to speak. Keypress events are also used when the application desires complete control of grammars and timing constraints.

When used in a subscription context, the <keypress> element has two attributes, 'report' and 'maskdigits', which are detailed in the list below.

Keypress Subscription Attributes:

- report - required, no default value: Possible values are 'standard', 'long', 'both', and 'none'. 'Standard' means that detected digits should be reported. 'Long' means that long digits should be reported. 'Long' digits are defined as a single key press held down for more than one second, or two distinct key presses (a "double") of the same digit that occur within two seconds of each other with no other intervening digits. 'Both' means that both 'standard' and 'long' digit events should be reported. As a 'long' digit consists of one or more "normal" digits, a single long duration key press will generate one standard event and one 'long' event. A "double" will produce two standard events and one 'long' event. 'None' means that no keypress events should be reported; it disables keypress event reporting if enabled.

- maskdigits - optional, default value "no": Controls whether user DTMF inputs are captured in media server log files. The possible values for this attribute are "yes" and "no".

The media server sends an MSCML response to the subscription immediately upon receiving the request. Notifications are sent to the client when the specified events are detected.

When used in a notification context, the <keypress> element has several attributes that are used to convey details of the event that was detected. It also contains a child element, <status>, that provides information on any MSCML request that was in progress when the event occurred. The details of these notification attributes are described in the list below.

Keypress Notification Attributes:

- digit - required, no default value: Specifies the DTMF digit detected. Possible values are [0-9], [A-D], '#', or '*'.

- length - required, no default value: Specifies the duration class of the DTMF input. Possible values are 'standard' or 'long'.

- method - required, no default value: Specifies the keypress detection method that generated the notification. Possible values are 'standard', 'long', and 'double'.

- interdigittime - required, no default value: Specifies the elapsed time, as a time value (Section 4.2.1), between the current event detection and the previous one.

7.1.1. Keypress Subscription Examples

The following examples of MSCML payloads depict a subscription for standard keypress events and disabling keypress reporting.

Figure 21 shows a subscription for standard keypress events.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <configure_leg>
      <subscribe>
        <events>
          <keypress report="standard"/>
        </events>
      </subscribe>
    </configure_leg>
  </request>
</MediaServerControl>
```

Figure 21: Standard Digit Events Subscription

Figure 22 shows a client disabling keypress event notifications.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <configure_leg>
      <subscribe>
        <events>
          <keypress report="none"/>
        </events>
      </subscribe>
    </configure_leg>
  </request>
</MediaServerControl>
```

Figure 22: Disabling Keypress Event Reporting

7.1.2. Keypress Notification Examples

The following MSCML payloads depict keypress event notifications caused by various types of DTMF input.

Figure 23 shows a notification generated by the detection of a standard "4" DTMF digit. In this example, this is the first digit detected. Thus, the 'interdigittime' attribute has a value of '0'.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <notification>
    <keypress digit="4" length="standard" method="standard"
      interdigittime="0">
      <status command="play" duration="10"/>
    </keypress>
  </notification>
</MediaServerControl>
```

Figure 23: Standard Keypress Notification

Figure 24 shows a notification generated by detection of a long pound (#).

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <notification>
    <keypress digit="#" length="long" method="long"
      interdigittime="200">
      <status command="idle" duration="4"/>
    </keypress>
  </notification>
</MediaServerControl>
```

Figure 24: Long Keypress Notification

7.2. Signal Events

MSCML supports notification of certain call progress tones through the <signal> element. When used in a subscription context, the <signal> element has two attributes, 'type' and 'report', and no child elements. These attributes are detailed in the list below.

Signal Subscription Attributes:

- report - required, no default value: Controls whether the specified signal is reported. Possible values are 'yes' and 'no'. When set to 'yes', the media server invokes the required signal detection code and reports detected events. When it is set to 'no', the media server disables the associated signal detection code and does not report events.

- type - required, no default value: Specifies the type of call progress signal to detect. Possible values are 'busy', 'ring', 'CED', 'CNG', and '400', which correspond to busy tone, ring tone, fax CED, fax CNG, and 400 Hz tone, respectively.

  NOTE: The details of media server provisioning required to support country-specific variants of 'busy' and 'ring' is not covered by this specification.

As stated previously, a single <configure_leg> request MAY contain multiple <signal> elements that request notification of different call progress tones. A single <configure_leg> request SHOULD NOT contain multiple <signal> elements that have the same 'type' attribute value. If the media server receives such a request, it SHOULD honor the last element specifying that type that appears in the request.

The media server generates an immediate response to the <configure_leg> subscription request and sends notifications when the specified signals are detected. A single notification is sent as soon as the specified signal has been reliably detected. If the signal persists continuously, additional notifications will not be sent. If the signal is interrupted and then resumes, additional notifications will be sent.

Signal notifications have a single attribute, "type", as described in the list below.

Signal Notification Attribute:

- type - required, no default value: Specifies the type of call progress signal that was detected. Possible values are 'busy', 'ring', 'CED', 'CNG', and '400', which correspond to busy tone, ring tone, fax CED, fax CNG, and 400 Hz tone, respectively.

7.2.1. Signal Event Examples

The following MSCML payloads show a signal event subscription (Figure 25) and notification (Figure 26).

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <configure_leg>
      <subscribe>
        <events>
          <signal type="busy" report="yes"/>
        </events>
      </subscribe>
    </configure_leg>
  </request>
</MediaServerControl>
```

Figure 25: Signal Event Subscription

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <notification>
    <signal type="busy"/>
  </notification>
</MediaServerControl>
```

Figure 26: Signal Event Notification

8. Managing Content <managecontent>

MSCML uses the <managecontent> request to move recorded content from the media server to remote locations using the HTTP protocol. This is a store-and-forward model, which requires the completion of local temporary recording before the media server can send it to the web server. This facility is useful in applications such as voice messaging, where a message may be reviewed by the caller prior to being committed to persistent storage.

The <managecontent> request contains no child elements and has the attributes described in the list below.

Managecontent Attributes:

- src - required, no default value: Specifies the local source URL of the content. The URL scheme MUST be "file://".

- dest - required (see note), no default value: Specifies the destination URL. The URL scheme MUST be "http://". Note: If the selected action is 'delete', this attribute is optional; otherwise it is required.

- action - optional, default value "move": Specifies the operation for the media server to execute. Values can be either 'move' or 'delete'. The 'delete' action operates on the local source file. After a successful move or delete, the media server deletes the source file from its local storage. If the request is unsuccessful, the source file is not deleted, which gives the client complete control of the retry strategy.

- httpmethod - optional, default value "post": HTTP protocol method for the media server to use in the HTTP request. The only values are 'post' or 'put'.

- name - required (see note), no default value: Specifies the field name for the content in the form when using the 'post' method. This is not to be confused with the "src" or "dest" attributes. Note: This attribute is required when the "htttpmethod" has the value "post" and is optional otherwise.

- fetchtimeout - optional, default value "10000ms": Specifies the maximum time allowed for the transfer to complete. Expressed as a time value (Section 4.2.1) from 1ms onwards.

- mimetype - required (see note), no default value: Specifies the MIME type that the media server will use for the content transfer. If it is not provided, the media server MUST try to infer it from the content file extension based on a platform specific mapping table. A non-normative, example mapping table is shown in Table 3. To avoid ambiguity, we RECOMMEND that clients explicitly set this attribute. Note: If the MIME type of the content cannot be inferred from the file extension, this attribute is required.

Table 3 shows common audio and video MIME types and possible file extension mappings.

Extension	MIME Type
alaw	audio/x-alaw-basic
ulaw	audio/basic
msgsm	audio/ms-gsm
wav	audio/x-wav
tif	image/tiff
tiff	image/tiff
mov	video/quicktime
qt	video/quicktime
3gp	video/3gpp
3gpp	video/3gpp

Table 3: Example File Extension to MIME Type Mappings

<Managecontent> is purely a transport operation; the underlying content is not changed by it. Therefore clients MUST ensure that the source and destination file name extensions and MIME types are the same. Failure to do so could result in content that is unreadable.

The ability to move or delete any local file presents a potential risk to the security of the media server system. For this reason, we STRONGLY RECOMMEND that implementers limit local file system access when using <managecontent>. For example, we encourage limiting access as based on file ownership and/or specific directories.

8.1. Managecontent Example

The following is an example (Figure 27) showing a local file on the media server being transferred to an HTTP URL using the "put" method. The client sends the following request.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <managecontent id="102"
    src="file:////var/mediaserver/rec/6A5GH49B.ulaw"
    dest="http://www.example.com/recordings/myrecording.ulaw"
    mimetype="audio/basic" action="move" httpmethod="put"
    fetchtimeout="5000"/>
  </request>
</MediaServerControl>
```

Figure 27: Managecontent Example

Note that the client can change the temporary file name assigned by the media server as part of this operation as shown.

If the request is ambiguous, the media server MUST return a status code of "400" and text "Bad Request." If the media server is unable to execute a syntactically correct and unambiguous request, it MUST return a "500" status code with the text "Server Error." For example, the local file system access restrictions may prevent deletion of the specified file. In this case, the "reason" attribute in the response conveys additional details on the server error that occurred. If there is a network or remote server error, the media server provides detailed error information in the <error_info> element contained in the media server response. Additional information regarding <managecontent> responses is provided in Section 10.7.

9. Fax Processing

9.1. Recording a Fax <faxrecord>

The <faxrecord> request directs the media server to process a fax in answer mode. The reason for a request separate from <playrecord> is that the media server needs to know to process the T.30 [18] or T.38 [19] fax protocols.

The <faxrecord> request has multiple attributes and one child element, <prompt>. Its attributes are described in the list below.

Attributes of <faxrecord>:

- lclid - optional, default value "" (the empty string): A string that identifies the called station.

- prompturl - optional, no default value: The URL of the fax content to be retrieved and played. The target may be a local or remote (NFS) "file://" scheme URL or an "http://" or "https://" scheme URL. NOTE: Use of this attribute is deprecated.

- promptencoding - optional, no default value: Specifies the content encoding for files that do not have a 'tif' or 'tiff' extension. The only allowable value is 'tiff'. This attribute only affects "file://" scheme URLs. NOTE: Use of this attribute is deprecated.

- recurl - optional, no default value: Specifies the target URL for the recorded content.

- rmtid - optional, no default value: Specifies the calling station identifier of the remote terminal. If present, the media server

MUST reject transactions with the remote terminal if the remote terminal's identifier does not match the value of 'rmtid'.

Clients SHOULD use the more flexible <prompt> mechanism for specifying fax content. Use of the 'prompturl' attribute is deprecated and may not be supported in future MSCML versions. The <prompt> element is described in Section 6.1.1. A <prompt> element sent in a <faxrecord> request MUST NOT contain <variable> elements.

Media servers MUST support local and remote (NFS) "file://" scheme URLs in the "recurl" attribute. MSCML supports "http://" and "https://" scheme URLs indirectly through the <managecontent> (Section 8) request.

The <faxrecord> request operates in one of three modes: receive, poll, and turnaround poll. The combination of <prompt> or 'prompturl' attribute and 'recurl' attribute define the mode. Table 4 describes these modes in detail. The 'prompt' column in the table has the value 'yes' if the request has either a <prompt> element or a 'prompturl' attribute.

prompt	recurl	Mode	Operation
no	no	Invalid	Request fails.
no	yes	Receive	Record the fax to the target URL specified in 'recurl'.
yes	no	Poll	Send fax from source specified in the <prompt> element or 'prompturl' attribute. If there is a 'rmtid', it MUST match the remote terminal's identifier, or the request will fail.
yes	yes	TP	Turnaround Poll (TP) mode. If the remote terminal wishes to transmit, the media server records the fax to the target URL specified in 'recurl'. If the remote terminal wishes to receive, the media server sends the fax from the source URL contained in <prompt> or 'prompturl'. If there is a 'rmtid', it MUST match remote terminal's identifier, or the send request will fail. A receive operation will still succeed, however.

Table 4: Fax Receive Modes

In receive mode, the media server receives the fax and writes the fax data to the target URL specified by the 'recurl' attribute.

In poll mode, the media server sends a fax, but as a polled (called) device.

In turnaround poll mode, the media server will record a fax that the remote machine sends. If the remote machine requests a transmission, then the media server will send the fax.

When transmitting a fax, the media server will advertise that it can receive faxes in the DIS message. Likewise, when receiving a fax, the media server will advertise that it can send faxes in the DIS message.

The media server MUST flush any quarantined digits when it receives a <faxrecord> request.

9.2. Sending a Fax <faxplay>

The <faxplay> request directs the media server to process a fax in originate mode. The reason for a request separate from <play> is that the media server needs to know to process the T.30 [18] or T.38 [19] fax protocols.

The <faxplay> request has multiple attributes and one child element, <prompt>. Its attributes are described in the list below.

Attributes of <faxplay>:

- lclid - optional, default value "" (the empty string): A string that identifies the called station.

- prompturl - optional, no default value: The URL of the content to be retrieved and played. The target may be a local or remote (NFS) "file://" scheme URL or an "http://" or "https://" scheme URL. NOTE: Use of this attribute is deprecated.

- promptencoding - optional, no default value: Specifies the content encoding for files that do not have a 'tif' or 'tiff' extension. The only allowable value is 'tiff'. This attribute only affects "file://" scheme URLs. NOTE: Use of this attribute is deprecated.

- recurl - optional, no default value: Specifies the target URL for the recorded content.

- rmtid - optional, no default value: Specifies the calling station identifier of the remote terminal. If present, the media server

MUST reject transactions with the remote terminal if the remote terminal's identifier does not match the value of 'rmtid'.

Clients SHOULD use the more flexible <prompt> mechanism for specifying fax content. Use of the 'prompturl' attribute is deprecated and may not be supported in future MSCML versions. The <prompt> element is described in Section 6.1.1. A <prompt> element sent in a <faxrecord> request MUST NOT contain <variable> elements.

Media servers MUST support local and remote (NFS) "file://" scheme URLs in the "recurl" attribute. MSCML supports "http://" and "https://" scheme URLs indirectly through the <managecontent> (Section 8) request.

The <faxplay> request operates in one of three modes: send, remote poll, and turnaround poll. The combination of <prompt> or 'prompturl' attribute and 'recurl' attribute define the mode. Table 5 describes these modes in detail. The 'prompt' column in the table has the value 'yes' if the request has either a <prompt> element or a 'prompturl' attribute.

prompt	recurl	Mode	Operation
no	no	Invalid	Request fails.
yes	no	Send	Send fax from source specified in the <prompt> element or 'prompturl' attribute. If there is a 'rmtid', it MUST match the remote terminal's identifier, or the request will fail.
no	yes	Poll	Send fax from source specified in the <prompt> element or 'prompturl' attribute, assuming the remote terminal specifies it can receive a fax in its DIS message. If the remote terminal does not support reverse polling, the request will fail. If 'rmtid' is specified, it MUST match remote terminal's identifier, or the request will fail.
yes	yes	TP	Turnaround Poll (TP) mode. If the remote terminal wishes to transmit, the media server records the fax to the target URL specified in 'recurl'. If the remote terminal wishes to receive, the media server sends the fax from the source URL contained in <prompt> or 'prompturl'. If there is a 'rmtid', it MUST match remote terminal's identifier, or the send request will fail. A receive operation will still succeed, however.

Table 5: Fax Send Modes

In send mode, the media server sends the fax.

In remote poll mode, the client places a call on behalf of the media server. The media server requests a fax transmission from the remote fax terminal.

In turnaround poll mode, the media server will record a fax that the remote machine sends. If the remote machine requests a transmission, then the media server will send the fax.

When transmitting a fax, the media server will advertise that it can receive faxes in the DIS message. Likewise, when receiving a fax,

the media server will advertise that it can send faxes in the DIS message.

The media server MUST flush any quarantined digits when it receives a <faxplay> request.

10. MSCML Response Attributes and Elements

10.1. Mechanism

The media server acknowledges receipt of a client MSCML request sent in a SIP INVITE by sending a response of either 200 OK or 415 Bad Media Type. The media server responds with 415 when the SIP request contains a content type other than "application/sdp" or "application/mediaservercontrol+xml".

The media server acknowledges receipt of a client MSCML request sent in a SIP INFO with a 200 OK or 415 Bad Media Type. The media server responds with 415 if the INFO request contains a content type other than "application/mediaservercontrol+xml".

The media server transports the MSCML <response> message in a SIP INFO request.

If there is an error in the request or the media server cannot complete the request, the media server sends the <response> message very shortly after receiving the request. If the request is able to proceed, the <response> contains final status information as described below.

10.2. Base <response> Attributes

All MSCML responses have the basic attributes defined in the list below.

Basic MSCML Response Attributes:

- id - optional, no default value: Echoes the client-defined ID contained in the request.
- request - required, no default value: Specifies the MSCML request type that generated the response. Allowable values are "configure_conference", "configure_leg", "play", "playcollect", "playrecord", "stop", "faxplay", "faxrecord", and "managecontent".

- code - required, no default value: The final status code for the request. MSCML uses a subset of the status classes defined in RFC 3261 [4]. In MSCML, 2XX responses indicate success, 4XX responses indicate client error, and 5XX responses indicate an error on the media server. There are no 1XX, 3XX, or 6XX status codes in MSCML.

- text - required, no default value: The human readable reason phrase associated with the status code.

Responses to <configure_conference> and <stop> requests contain only the attributes above. MSCML responses to other requests MAY contain additional request-specific attributes and elements. These are described in the following sections.

10.3. Response Attributes and Elements for <configure_leg>

Responses to <configure_leg> requests have only the base response attributes defined in Section 10.2. However, when the request contains a <configure_team> element, the response includes a <team> element describing the teammate configuration for that leg. The attributes of the <team> element are shown in the list below.

Attributes of <team>:

- id - required, no default value: The client-defined unique identifier for the conference leg.

- numteam - required, no default value: The number of team members for the leg.

Additional information on each team member is conveyed by child <teammate> elements contained within <team>. Each teammate is represented by a single element in the list. The <teammate> element has a single attribute, as described below.

Attributes of <teammate>:

- id - required, no default value: The client-defined unique identifier for the teammate leg.

10.4. Response Attributes and Elements for <play>

In addition to the base response attributes defined in Section 10.2, responses to <play> requests have the additional attributes described in the list below.

MSCML Response Attributes for <play>:

- reason - optional, no default value: For requests that are not completed immediately, the "reason" attribute conveys additional information regarding why the command was completed. Possible values are "stopped", indicating that an explicit or implicit <stop> request was received, and "EOF", indicating that the end of the specified sequence of URLs was reached.

- playduration - required, no default value: A time value (Section 4.2.1) that returns the duration of the associated content playout.

- playoffset - required, no default value: A time value (Section 4.2.1) that returns the time offset into the specified content sequence where play was terminated. If the initial "offset" value in the sequence was "0", then "playduration" and "playoffset" are equal. However, if the initial offset had some other value, "playoffset" serves as a bookmark for the client to resume play in a subsequent request.

10.4.1. Reporting Content Retrieval Errors

If the associated request set "stoponerror=yes" in <prompt> and an error occurred while retrieving the specified content the response will include an <error_info> element detailing the problem. This element contains the response information received from the remote content server. The <error_info> element has the attributes described in the list below.

Attributes of <error_info>:

- code - required, no default value: The status code returned by the remote content server. For example, a web server might return 404 to indicate that the requested content was not found.

- text - required, no default value: The human-readable reason phrase returned by the remote content server. For example, the reason phrase "Not Found" would be returned if the requested content was not found.

- context - required, no default value: Contains the content URL that was being fetched when the retrieval error occurred. This enables the client to know precisely which URL in a sequence caused the problem.

An <error_info> element MAY be present in the response to any request that contains a child <prompt> element.

10.5. Response Attributes and Elements for <playcollect>

In addition to the base response attributes defined in Section 10.2, responses to <playcollect> requests have the additional attributes described in the list below.

MSCML Response Attributes for <playcollect>:

- reason - optional, no default value: For requests that are not completed immediately, the "reason" attribute conveys additional information regarding why the command was completed. Possible values are "stopped", indicating that an explicit or implicit <stop> request was received; "match", meaning that a DTMF grammar was matched; "timeout", indicating that no DTMF input was received before one of the collection timers expired; and "returnkey" or "escapekey", meaning the DTMF digit mapped to that key was detected and the return key or escape key terminated the operation, respectively.

- playduration - required, no default value: A time value (Section 4.2.1) that returns the duration of the associated content playout. If the caller barged the prompt, this value will reflect the play duration up to that event.

- playoffset - required, no default value: A time value (Section 4.2.1) that returns the time offset into the specified content sequence where play was terminated. If the initial "offset" value in the sequence was "0", then "playduration" and "playoffset" are equal. However, if the initial offset had some other value, "playoffset" serves as a bookmark for the client to resume play in a subsequent request. If the caller barged the prompt this value will reflect the time offset at which barge-in occurred.

- digits - required, no default value: Contains the collected DTMF input characters. If no DTMF input was collected, this attribute is set to the empty string ("").

- name - required (see note), no default value: The client-defined name of the DTMF grammar that was matched. Note: This attribute is required if the "name" attribute was set in the matching DTMF grammar.

Responses to <playcollect> requests MAY include an <error_info> element, as described in Section 10.4.1.

10.6. Response Attributes and Elements for <playrecord>

In addition to the base response attributes defined in Section 10.2, responses to <playrecord> requests have the additional attributes described in the list below.

- reason - optional, no default value: For requests that are not completed immediately, the "reason" attribute conveys additional information regarding why the command was completed. Possible values are "stopped", indicating that an explicit or implicit <stop> request was received; "digit", meaning that a DTMF digit was detected and that the prompt phase was barged; "init_silence", meaning the recording terminated because of no input; "end_silence", meaning that the recording was terminated because the "endsilence" timer elapsed; "max_duration", indicating that the maximum time for the recording was reached; "escapekey", indicating that the DTMF input mapped to "escapekey" was detected, thus terminating the recording; and "error", indicating a general operation failure.

- playduration - required, no default value: A time value (Section 4.2.1) that returns the duration of the associated content playout. If the caller barged the prompt, this value will reflect the play duration up to that event.

- playoffset - required, no default value: A time value (Section 4.2.1) that returns the time offset into the specified content sequence where play was terminated. If the initial "offset" value in the sequence was "0", then "playduration" and "playoffset" are equal. However, if the initial offset had some other value, "playoffset" serves as a bookmark for the client to resume play in a subsequent request. If the caller barged the prompt this value will reflect the time offset at which barge-in occurred.

- digits - required, no default value: Contains the DTMF digit that terminated the recording. If no DTMF input was detected, this attribute is set to the empty string ("").

- reclength - required, no default value: The length of the recorded content, in bytes.

- recduration - required, no default value: A time value (Section 4.2.1) indicating the elapsed duration of the recording.

Responses to <playrecord> requests MAY include an <error_info> element, as described in Section 10.4.1.

10.7. Response Attributes and Elements for <managecontent>

Responses to <managecontent> requests have only the base response attributes defined in Section 10.2. If a content transfer error occurs while executing the request the response will also contain an <error_info> element as described in Section 10.4.1.

10.8. Response Attributes and Elements for <faxplay> and <faxrecord>

In addition to the base response attributes defined in Section 10.2, responses to <faxplay> and <faxrecord> requests have the additional attributes described in the list below.

- reason - required, no default value: For requests that are not completed immediately, the "reason" attribute conveys additional information regarding why the command was completed. Possible values are "stopped", indicating that an explicit or implicit <stop> request was received; "complete", indicating successful completion, even if there were bad lines or minor negotiation problems (e.g., a DCN was received); "disconnect", meaning that the session was disconnected; and "notfax", indicating that no DIS or DCS was received on the connection.

- pages_received - required (see note), no default value: Indicates the number of fax pages received. Note: This attribute is required if any pages were received.

- pages_sent - required (see note), no default value: Indicates the number of fax pages sent. Note: This attribute is required if any pages were sent.

- faxcode - required, no default value: The value of the "faxcode" attribute is the binary-or of the bit patterns defined in Table 6.

Mask	description
0	Operation Failed
1	Operation Succeeded
2	Partial Success
4	Image received and placed in recurl
8	Image sent from specified source URL
16	rmtid did not match
32	Error reading source URL
64	Error writing recurl
128	Negotiation failure on send phase
256	Negotiation failure on receive phase
512	Reserved
1024	Irrecoverable IP packet loss
2048	Line errors in received image

Table 6: Faxcode Mask

Responses to <faxplay> and <faxrecord> requests MAY include an <error_info> element, as described in Section 10.4.1.

11. Formal Syntax

The following syntax specification uses XML Schema as described in XML [7].

11.1. Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="MediaServerControl">
    <xs:complexType>
      <xs:choice>
        <xs:element name="request">
          <xs:complexType>
            <xs:choice>
              <xs:element name="configure_conference"
                type="configure_conferenceRequestType"/>
              <xs:element name="configure_leg"
                type="configure_legRequestType"/>
              <xs:element name="play" type="playRequestType"/>
              <xs:element name="playcollect"
                type="playcollectRequestType"/>
              <xs:element name="playrecord"
                type="playrecordRequestType"/>
```

```
              <xs:element name="managecontent"
                type="managecontentRequestType"/>
              <xs:element name="faxplay"
                type="faxRequestType"/>
              <xs:element name="faxrecord"
                type="faxRequestType"/>
              <xs:element name="stop" type="stopRequestType"/>
            </xs:choice>
          </xs:complexType>
        </xs:element>
        <xs:element name="response" type="responseType"/>
        <xs:element name="notification">
          <xs:complexType>
            <xs:choice>
              <xs:element name="conference"
                type="conferenceNotificationType"/>
              <xs:element name="keypress"
                type="keypressNotificationType"/>
              <xs:element name="signal"
                type="signalNotificationType"/>
            </xs:choice>
          </xs:complexType>
        </xs:element>
      </xs:choice>
      <xs:attribute name="version" use="required"/>
    </xs:complexType>
  </xs:element>
  <!-- Definitions for base and concrete MSCML requests -->
  <!-- and embedded types. -->
  <xs:complexType name="base_requestType" abstract="true">
    <xs:attribute name="id" type="xs:string"/>
  </xs:complexType>
  <xs:complexType name="playRequestType">
    <xs:complexContent>
      <xs:extension base="base_requestType">
        <xs:sequence>
          <xs:element name="prompt" type="promptType"
            minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="prompturl" type="xs:string"/>
        <xs:attribute name="offset" type="xs:string"/>
        <xs:attribute name="promptencoding" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="configure_conferenceRequestType">
    <xs:complexContent>
      <xs:extension base="base_requestType">
```

```
          <xs:sequence>
            <xs:element name="subscribe"
              type="conference_eventsubscriptionType" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="reservedtalkers"
            type="xs:positiveInteger"/>
          <xs:attribute name="reserveconfmedia" type="yesnoType"
            default="yes"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="configure_legRequestType">
      <xs:complexContent>
        <xs:extension base="base_requestType">
          <xs:sequence>
            <xs:element name="inputgain" type="gainType"
              minOccurs="0"/>
            <xs:element name="outputgain" type="gainType"
              minOccurs="0"/>
            <xs:element name="configure_team"
              type="configure_teamType" minOccurs="0"/>
            <xs:element name="subscribe"
              type="leg_eventsubscriptionType" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="type">
            <xs:simpleType>
              <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="talker"/>
                <xs:enumeration value="listener"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="mixmode">
            <xs:simpleType>
              <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="full"/>
                <xs:enumeration value="mute"/>
                <xs:enumeration value="preferred"/>
                <xs:enumeration value="parked"/>
                <xs:enumeration value="private"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute name="dtmfclamp" type="yesnoType"/>
          <xs:attribute name="toneclamp" type="yesnoType"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
```

```
<xs:complexType name="configure_teamType">
  <xs:sequence>
    <xs:element name="teammate" type="teammateType" minOccurs="0"
      maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string"/>
  <xs:attribute name="action" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="add"/>
        <xs:enumeration value="delete"/>
        <xs:enumeration value="query"/>
        <xs:enumeration value="set"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="teammateType">
  <xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="playcollectRequestType">
  <xs:complexContent>
    <xs:extension base="base_requestType">
      <xs:sequence>
        <xs:element name="prompt" type="promptType"
          minOccurs="0"/>
        <xs:element name="pattern" type="dtmfGrammarType"
          minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="prompturl" type="xs:string"/>
      <xs:attribute name="offset" type="xs:string"/>
      <xs:attribute name="barge" type="yesnoType" default="yes"/>
      <xs:attribute name="promptencoding" type="xs:string"/>
      <xs:attribute name="cleardigits" type="yesnoType"
        default="no"/>
      <xs:attribute name="maxdigits" type="xs:string"/>
      <xs:attribute name="firstdigittimer" type="xs:string"
        default="5000ms"/>
      <xs:attribute name="interdigittimer" type="xs:string"
        default="2000ms"/>
      <xs:attribute name="extradigittimer" type="xs:string"
        default="1000ms"/>
      <xs:attribute name="interdigitcriticaltimer"
        type="xs:string"/>
      <xs:attribute name="skipinterval" type="xs:string"
        default="6s"/>
      <xs:attribute name="ffkey" type="DTMFkeyType"/>
      <xs:attribute name="rwkey" type="DTMFkeyType"/>
```

```
        <xs:attribute name="returnkey" type="DTMFkeyType"
          default="#"/>
        <xs:attribute name="escapekey" type="DTMFkeyType"
          default="*"/>
        <xs:attribute name="maskdigits" type="yesnoType"
          default="no"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="playrecordRequestType">
    <xs:complexContent>
      <xs:extension base="base_requestType">
        <xs:sequence>
          <xs:element name="prompt" type="promptType"
            minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="prompturl" type="xs:string"/>
        <xs:attribute name="promptencoding" type="xs:string"/>
        <xs:attribute name="offset" type="xs:string" default="0"/>
        <xs:attribute name="barge" type="yesnoType" default="yes"/>
        <xs:attribute name="cleardigits" type="yesnoType"
          default="no"/>
        <xs:attribute name="escapekey" type="xs:string" default="*"/>
        <xs:attribute name="recurl" type="xs:string" use="required"/>
        <xs:attribute name="mode" default="overwrite">
          <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
              <xs:enumeration value="append"/>
              <xs:enumeration value="overwrite"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="recencoding" type="xs:string"/>
        <xs:attribute name="initsilence" type="xs:string"/>
        <xs:attribute name="endsilence" type="xs:string"/>
        <xs:attribute name="duration" type="xs:string"/>
        <xs:attribute name="beep" type="yesnoType" default="yes"/>
        <xs:attribute name="recstopmask" type="xs:string"
          default="01234567890*#"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="managecontentRequestType">
    <xs:complexContent>
      <xs:extension base="base_requestType">
        <xs:attribute name="fetchtimeout" type="xs:string"
          default="10000"/>
        <xs:attribute name="mimetype" type="xs:string"/>
```

```
        <xs:attribute name="name" type="xs:string"/>
        <xs:attribute name="httpmethod">
          <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
              <xs:enumeration value="put"/>
              <xs:enumeration value="post"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="action">
          <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
              <xs:enumeration value="move"/>
              <xs:enumeration value="delete"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="dest" type="xs:string"/>
        <xs:attribute name="src" type="xs:string" use="required"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="stopRequestType">
    <xs:complexContent>
      <xs:extension base="base_requestType"/>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="faxRequestType">
    <xs:complexContent>
      <xs:extension base="base_requestType">
        <xs:sequence>
          <xs:element name="prompt" type="promptType" minOccurs="0"/>
        </xs:sequence>
        <xs:attribute name="lclid" type="xs:string"/>
        <xs:attribute name="prompturl" type="xs:string"/>
        <xs:attribute name="recurl" type="xs:string"/>
        <xs:attribute name="rmtid" type="xs:string"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="dtmfGrammarType">
    <xs:choice>
      <xs:element name="regex" type="dtmfPatternType"
        maxOccurs="unbounded"/>
      <xs:element name="mgcpdigitmap" type="dtmfPatternType"/>
      <xs:element name="megacodigitmap" type="dtmfPatternType"/>
    </xs:choice>
  </xs:complexType>
```

```
    <xs:complexType name="dtmfPatternType">
      <xs:attribute name="value" type="xs:string" use="required"/>
      <xs:attribute name="name" type="xs:string"/>
    </xs:complexType>
    <!-- Definitions for base and concrete MSCML responses -->
    <!-- and embedded types. -->
    <xs:complexType name="base_responseType" abstract="true">
      <xs:attribute name="request" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="configure_conference"/>
            <xs:enumeration value="configure_leg"/>
            <xs:enumeration value="play"/>
            <xs:enumeration value="playcollect"/>
            <xs:enumeration value="playrecord"/>
            <xs:enumeration value="managecontent"/>
            <xs:enumeration value="faxplay"/>
            <xs:enumeration value="faxrecord"/>
            <xs:enumeration value="stop"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="id" type="xs:string"/>
      <xs:attribute name="code" type="xs:string" use="required"/>
      <xs:attribute name="text" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="responseType">
      <xs:complexContent>
        <xs:extension base="base_responseType">
          <xs:sequence>
            <xs:element name="error_info"
              type="stoponerrorResponseType" minOccurs="0"/>
            <xs:element name="team" type="configure_teamResponseType"
              minOccurs="0"/>
          </xs:sequence>
          <xs:attribute name="reason" type="xs:string"/>
          <xs:attribute name="reclength" type="xs:string"/>
          <xs:attribute name="recduration" type="xs:string"/>
          <xs:attribute name="digits" type="xs:string"/>
          <xs:attribute name="name" type="xs:string"/>
          <xs:attribute name="playduration" type="xs:string"/>
          <xs:attribute name="playoffset" type="xs:string"/>
          <xs:attribute name="faxcode" type="xs:string"/>
          <xs:attribute name="pages_sent" type="xs:string"/>
          <xs:attribute name="pages_recv" type="xs:string"/>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
```

```
    <xs:complexType name="stoponerrorResponseType">
      <xs:attribute name="code" type="xs:string" use="required"/>
      <xs:attribute name="text" type="xs:string" use="required"/>
      <xs:attribute name="context" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="configure_teamResponseType">
      <xs:sequence>
        <xs:element name="teammate" type="teammateType" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="id" type="xs:string" use="required"/>
      <xs:attribute name="numteam" type="xs:integer" use="required"/>
    </xs:complexType>
    <!-- Definitions for MSCML event subscriptions and -->
    <!-- embedded types -->
    <xs:complexType name="conference_eventsubscriptionType">
      <xs:sequence>
        <xs:element name="events">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="activetalkers"
                type="activetalkersSubscriptionType"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="activetalkersSubscriptionType">
      <xs:attribute name="report" type="yesnoType" use="required"/>
      <xs:attribute name="interval" type="xs:string" default="60s"/>
    </xs:complexType>
    <xs:complexType name="leg_eventsubscriptionType">
      <xs:sequence>
        <xs:element name="events">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="keypress"
                type="keypressSubscriptionType" minOccurs="0"
                  maxOccurs="1"/>
              <xs:element name="signal" type="signalSubscriptionType"
                minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="keypressSubscriptionType">
      <xs:attribute name="report" use="required">
```

```
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="standard"/>
            <xs:enumeration value="long"/>
            <xs:enumeration value="both"/>
            <xs:enumeration value="none"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="maskdigits" type="yesnoType" default="no"/>
    </xs:complexType>
    <xs:complexType name="signalSubscriptionType">
      <xs:attribute name="type" type="xs:NMTOKEN" use="required"/>
      <xs:attribute name="report" type="yesnoType" use="required"/>
    </xs:complexType>
    <!-- Definitions for MSCML event notifications and -->
    <!-- embedded types. -->
    <xs:complexType name="conferenceNotificationType">
      <xs:sequence>
        <xs:element name="activetalkers"
          type="activetalkersNotificationType" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="uniqueid" type="xs:string" use="required"/>
      <xs:attribute name="numtalkers" type="xs:string"
        use="required"/>
    </xs:complexType>
    <xs:complexType name="activetalkersNotificationType">
      <xs:sequence minOccurs="0">
        <xs:element name="talker" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="callid" type="xs:string"
             use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="keypressNotificationType">
      <xs:sequence>
        <xs:element name="status" type="statusType"/>
      </xs:sequence>
      <xs:attribute name="digit" type="DTMFkeyType" use="required"/>
      <xs:attribute name="length" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="standard"/>
            <xs:enumeration value="long"/>
          </xs:restriction>
        </xs:simpleType>
```

```
      </xs:attribute>
      <xs:attribute name="method" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="standard"/>
            <xs:enumeration value="long"/>
            <xs:enumeration value="double"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="interdigittime" type="xs:string"
        use="required"/>
    </xs:complexType>
    <xs:complexType name="statusType">
      <xs:attribute name="command" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="idle"/>
            <xs:enumeration value="play"/>
            <xs:enumeration value="collect"/>
            <xs:enumeration value="record"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="duration" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:complexType name="signalNotificationType">
      <xs:attribute name="type" use="required" fixed="busy"/>
    </xs:complexType>
    <!-- Definitions for miscellaneous embedded, helper data types -->
    <xs:complexType name="promptType">
      <xs:choice maxOccurs="unbounded">
        <xs:element name="audio" type="promptcontentType"/>
        <xs:element name="variable" type="spokenvariableType"/>
      </xs:choice>
      <xs:attribute name="locale" type="xs:string"/>
      <xs:attribute name="baseurl" type="xs:string"/>
      <xs:attribute name="stoponerror" type="yesnoType" default="no"/>
      <xs:attribute name="gain" type="xs:string" default="0"/>
      <xs:attribute name="gaindelta" type="xs:string" default="0"/>
      <xs:attribute name="rate" type="xs:string" default="0"/>
      <xs:attribute name="ratedelta" type="xs:string" default="0"/>
      <xs:attribute name="repeat" type="xs:string" default="1"/>
      <xs:attribute name="duration" type="xs:string"
        default="infinite"/>
      <xs:attribute name="offset" type="xs:string" default="0"/>
      <xs:attribute name="delay" type="xs:string" default="0"/>
    </xs:complexType>
```

```
    <xs:complexType name="promptcontentType">
      <xs:attribute name="url" type="xs:string" use="required"/>
      <xs:attribute name="encoding" type="xs:string"/>
      <xs:attribute name="gain" type="xs:string" default="0"/>
      <xs:attribute name="gaindelta" type="xs:string" default="0"/>
      <xs:attribute name="rate" type="xs:string" default="0"/>
      <xs:attribute name="ratedelta" type="xs:string" default="0"/>
    </xs:complexType>
    <xs:complexType name="spokenvariableType">
      <xs:attribute name="type" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="dat"/>
            <xs:enumeration value="dig"/>
            <xs:enumeration value="dur"/>
            <xs:enumeration value="mth"/>
            <xs:enumeration value="mny"/>
            <xs:enumeration value="num"/>
            <xs:enumeration value="sil"/>
            <xs:enumeration value="str"/>
            <xs:enumeration value="tme"/>
            <xs:enumeration value="wkd"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="subtype">
        <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
            <xs:enumeration value="mdy"/>
            <xs:enumeration value="dmy"/>
            <xs:enumeration value="ymd"/>
            <xs:enumeration value="ndn"/>
            <xs:enumeration value="t12"/>
            <xs:enumeration value="t24"/>
            <xs:enumeration value="USD"/>
            <xs:enumeration value="gen"/>
            <xs:enumeration value="ndn"/>
            <xs:enumeration value="crd"/>
            <xs:enumeration value="ord"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="value" type="xs:string" use="required"/>
    </xs:complexType>
    <xs:simpleType name="yesnoType">
      <xs:restriction base="xs:NMTOKEN">
        <xs:enumeration value="yes"/>
        <xs:enumeration value="no"/>
```

```
        <xs:enumeration value="1"/>
        <xs:enumeration value="0"/>
        <xs:enumeration value="true"/>
        <xs:enumeration value="false"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:simpleType name="DTMFkeyType">
      <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]"/>
        <xs:pattern value="[A-D]"/>
        <xs:pattern value="[a-d]"/>
        <xs:pattern value="#"/>
        <xs:pattern value="\*"/>
      </xs:restriction>
    </xs:simpleType>
    <xs:complexType name="gainType">
      <xs:choice>
        <xs:element name="auto" type="autogainType"/>
        <xs:element name="fixed" type="fixedgainType"/>
      </xs:choice>
    </xs:complexType>
    <xs:complexType name="autogainType">
      <xs:attribute name="startlevel" type="xs:string"/>
      <xs:attribute name="targetlevel" type="xs:string"/>
      <xs:attribute name="silencethreshold" type="xs:string"/>
    </xs:complexType>
    <xs:complexType name="fixedgainType">
      <xs:attribute name="level" type="xs:string"/>
    </xs:complexType>
  </xs:schema>
```

12. IANA Considerations

12.1. IANA Registration of MIME Media Type application/mediaservercontrol+xml

MIME media type name: application
MIME subtype name: mediaservercontrol+xml
Required parameters: none
Optional parameters: charset

charset This parameter has identical semantics to the charset parameter of the "application/xml" media type, as specified in XML Media Types [8].

Encoding considerations: See RFC 3023 [8].
Interoperability considerations: See RFC 2023 [8] and RFC 4722.
Published specification: RFC 4722

Applications that use this media type: Multimedia, enhanced conferencing and interactive applications.
Personal and email address for further information: eburger@cantata.com [31]
Intended usage: COMMON

13. Security Considerations

Because media flows through a media server in a conference, the media server itself MUST protect the integrity, confidentiality, and security of the sessions. It should not be possible for a conference participant, on her own behalf, to be able to "tap in" to another conference without proper authorization.

Because conferencing is a high-value application, the media server SHOULD implement appropriate security measures. This includes, but is not limited to, access lists for application servers. That is, the media server only allows a select list of application or proxy servers to create conferences, to invite participants to sessions, etc. Note that the mechanisms for such security, like private networks, shared certificates, MAC white/black lists, are beyond the scope of this document.

Security concerns are one important reason MSCML limits requests with conference scope to a separate control leg per conference. MSCML uses the simple, proven, Internet-scale security model of SIP to determine if a client is who they say they are (authentication) and if they are allowed to create and manipulate a conference. However, the security model to enable a control leg to manipulate arbitrary conferences on the media server is extremely difficult. Not only would one need to authenticate and authorize the basic conference primitives, but privacy considerations require policies for one client to access another client's conferences, even if the two clients are on the same host. For example, if the media server allowed any control leg to control any conference, an authorized but unrelated client could maliciously attach itself to an existing session and record or tap the conversation without the participant's knowledge or consent.

Participants give implicit authorization to their applications by virtue of the INVITE to the application. However, there is no trust, explicit or implicit, between the users of one service and a distinct client of another service.

All MSCML messages are sent within an INVITE-created SIP dialog. As a result, it would be difficult for an entity other than the original requestor to interfere with an established MSCML session, as this would require detailed information on the dialog state. This allows

multiple applications to utilize the resources of a single media server simultaneously without interfering with one another.

Because of the sensitive nature of collected data, such as credit card numbers or other identifying information, the media server MUST support sips: and TLS. Clients, who presumably know the value of the information they collect, as well as the privacy expectations of their users, are free to use clear text signaling or encrypted secure signaling, depending on the application's needs. Likewise, the media server SHOULD support Secure Realtime Transport Protocol (SRTP) [9]. Again, the clients are free to negotiate the appropriate level of media security.

The media management facilities of MSCML, such as the <managecontent> (Section 8) request, assume a trust relationship between the media server and file server. This scenario is similar to the one addressed by URLAUTH [20]. Namely, the media server is acting on behalf of a given user, yet the media server does not have credentials for that user. One might be tempted to use the user:pass facility of the HTTP URI to offer per-user security, but that also requires that the media server be secure, as the media server would need to know the user credentials in a form that is easily compromised (clear text passwords).

The IETF is investigating methods for providing per-user or per-instance authorization of third-party http writing, as is needed for other protocols as well, such as WEBDAV [21]. However, until that work is completed, media server implementations MUST be prepared to authenticate themselves to file and web servers using appropriate authentication means. At a minimum, the media server MUST support HTTPS basic authentication. Implementers should note that the media server will need to respond appropriately to whatever authentication mechanism the file server requires.

As this is an XML markup, all the security considerations of RFC 3023 [8] apply.

14. References

14.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[2] Burger, E., Van Dyke, J., and A. Spitzer, "Basic Network Media Services with SIP", RFC 4240, December 2005.

[3] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[4] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[5] "Network call signalling protocol for the delivery of time-critical services over cable television networks using cable modems", ITU-T J.162, March 2001.

[6] Groves, C., Pantaleo, M., Anderson, T., and T. Taylor, "Gateway Control Protocol Version 1", RFC 3525, June 2003.

[7] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures", W3C REC REC-xmlschema-1-20010502, May 2001.

[8] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[9] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

14.2. Informative References

[10] Rosenberg, J., "A Framework for Conferencing with the Session Initiation Protocol (SIP)", RFC 4353, February 2006.

[11] Carter, J., Danielsen, P., Hunt, A., Ferrans, J., Lucas, B., Porter, B., Rehor, K., Tryphonas, S., McGlashan, S., and D. Burnett, "Voice Extensible Markup Language (VoiceXML) Version 2.0", W3C REC REC-voicexml20-20040316, March 2004.

[12] International Packet Communications Consortium, "IPCC Reference Architecture V2", June 2002.

[13] European Telecommunications Standards Institute, "Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); IP Multimedia Subsystem (IMS); Stage 2 (3GPP TS 23.228 version 7.2.0 Release 7)", December 2005.

[14] Hollenbeck, S., Rose, M., and L. Masinter, "Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols", BCP 70, RFC 3470, January 2003.

[15] Jacobs, I., Lie, H., Bos, B., and C. Lilley, "Cascading Style Sheets, level 2 (CSS2) Specification", W3C REC REC-CSS2-19980512, May 1998.

[16] Rosenberg, J., Schulzrinne, H., and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State", RFC 4575, August 2006.

[17] Cable Television Laboratories, Inc., "Audio Server Protocol", January 2005.

[18] "Procedures for document facsimile transmission in the general switched telephone network", Recommendation T.30, April 1999.

[19] "Procedures for real-time Group 3 facsimile communication over IP networks", Recommendation T.38, March 2002.

[20] Crispin, M., "Internet Message Access Protocol (IMAP) - URLAUTH Extension", RFC 4467, May 2006.

[21] Goland, Y., Whitehead, E., Faizi, A., Carter, S., and D. Jensen, "HTTP Extensions for Distributed Authoring -- WEBDAV", RFC 2518, February 1999.

[22] Institute of Electrical and Electronics Engineers, "Information Technology - Portable Operating System Interface (POSIX) - Part 1: Base Definitions, Chapter 9", IEEE Standard 1003.1, June 2001.

[23] Burger, E. and M. Dolly, "A Session Initiation Protocol (SIP) Event Package for Key Press Stimulus (KPML)", RFC 4730, November 2006.

[24] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[25] Campbell, B., Ed., Mahy, R., Ed., and C. Jennings, Ed., "The Message Session Relay Protocol", Work in Progress, June 2006.

URIs

[26] <http://www.ietf.org/html.charters/sip-charter.html>

[27] <http://www.ietf.org/html.charters/sipping-charter.html>

[28] <http://www.ietf.org/html.charters/mmusic.html>

[29] <http://www.ietf.org/html.charters/xcon-charter.html>

[30] <http://www.3gpp.org/ftp/Specs/html-info/23228.htm>

[31] <mailto:eburger@cantata.com>

Appendix A. Regex Grammar Syntax

The regular expression syntax used in MSCML is a telephony-oriented subset of POSIX Extended Regular Expressions (ERE) [22] termed Digit REGular EXpression (DRegex). This syntax was first described in KPML [23].

DRegex includes ordinary characters, special characters, bracket expressions, and interval expressions. These entities are defined in the list below.

character matches digits 0-9, *, #, and A-D (case insensitive)
* matches the * character
# matches the # character
[character selector] matches any character in selector
[range1-range2] matches any character in range from range1 to range2, inclusive
x matches any digit 0-9
{m} matches m repetitions of the previous pattern
{m,} matches m or more repetitions of the previous pattern
{,n} matches at most n (including zero) repetitions of the previous pattern
{m,n} at least m and at most n repetitions of the previous pattern
L the presence of 'L' in any regex expression causes the media server to enable "long" digit detection mode. See Section 7.1 for the definition of "long" digits.

Table 7 illustrates DRegex usage through examples.

Example	Description
1	Matches the digit 1
[179]	Matches 1, 7, or 9
[2-9]	Matches 2, 3, 4, 5, 6, 7, 8, 9
[02-46-9A-D]	Matches 0, 2, 3, 4, 6, 7, 8, 9, A, B, C, D
x	Matches 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
*6[179#]	Matches *61, *67, *69, or *6#
x{10}	Ten digits (0-9)
011x{7,15}	011 followed by seven to fifteen digits
L*	Long star

Table 7: DRegex Examples

Appendix B. Contributors

Jeff Van Dyke and Andy Spitzer did the concept, development, documentation, and execution for MSCML at SnowShore Networks, Inc., which is now part of Cantata Technology, Inc. Andy Spitzer's original work at The Telephone Connection, Inc., influenced the IVR implementation. Mary Ann Leekley implemented the personalized mix feature and several other enhancements.

Cliff Schornak of Commetrex and Jeff Van Dyke developed the facsimile service.

Jai Cauvet, Rolando Herrero, Srinivas Motamarri, and Ashish Patel contributed greatly by testing MSCML.

Appendix C. Acknowledgements

The following individuals provided valuable assistance in the direction, development, or debugging of MSCML:

- Brian Badger and Phil Crable from Verizon Business
- Stephane Bastien from BroadSoft
- Peter Danielsen of Lucent Technologies
- Kevin Flemming, formerly of SnowShore Networks, Inc.
- Wesley Hicks and Ravindra Kabre, formerly from Sonus Networks
- Jon Hinckley from SkyWave/Sestro
- Terence Lobo, formerly of SnowShore Networks, Inc.
- Kunal Nawale, formerly of SnowShore Networks, Inc.
- Edwina Nowicki, formerly of SnowShore Networks, Inc.
- Diana Rawlins and Sharadha Vijay, formerly of WorldCom
- Gaurav Srivastva and Subhash Verma from BayPackets
- Kevin Summers from Sonus Networks
- Tim Wong from at&t

The authors would like to thank Cullen Jennings and Dan Wing from Cisco Systems for their helpful review comments.

The authors would also like to thank Scotty Farber for applying her technical writing expertise to the documentation of MSCML.

Authors' Addresses

Jeff Van Dyke
Cantata Technology, Inc.
18 Keewaydin Dr.
Salem, NH 03079
USA

EMail: jvandyke@cantata.com

Eric Burger (editor)
Cantata Technology, Inc.
18 Keewaydin Dr.
Salem, NH 03079
USA

EMail: eburger@cantata.com

Andy Spitzer
Pingtel Corporation
400 West Cummings Park
Woburn, MA 01801
USA

EMail: woof@pingtel.com

Full Copyright Statement

Intellectual Property

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.