Dialogic
Making Innovation Thrive™

Dialogic® IP Media Server Release 2.4.0

Application Developer's Guide

July 2008 939-028-32

www.dialogic.com

Copyright and Legal Disclaimer

All contents of this document are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries. Reasonable effort is made to ensure the accuracy of the information contained in the document. However, due to ongoing product improvements and revisions, Dialogic Corporation and its subsidiaries do not warrant the accuracy of this information and cannot accept responsibility for errors or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS EXPLICITLY SET FORTH BELOW OR AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at *www.dialogic.com*.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic Corporation or its subsidiaries may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic Corporation or its subsidiaries do not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic Corporation or its subsidiaries. More detailed information about such intellectual property is available from Dialogic Corporation's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9.

The software referred to in this document is provided under an End User Software License Agreement. Refer to the End User Software License Agreement that was provided with the software for details governing the use of the software.

Dialogic Corporation encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Any use case(s) shown and/or described herein represent one or more examples of the various ways, scenarios or environments in which Dialogic® products can be used. Such use case(s) are non-limiting and do not represent recommendations of Dialogic as to whether or how to use Dialogic products.

Dialogic, Dialogic Pro, Brooktrout, Cantata, SnowShore, Eicon, Eicon Networks, Eiconcard, Diva, SIPcontrol, Diva ISDN, TruFax, Realblocs, Realcomm 100, NetAccess, Instant ISDN, TRXStream, Exnet, Exnet Connect, EXS, ExchangePlus VSE, Switchkit, N20, Powering The Service-Ready Network, Vantage, Connecting People to Information, Connecting to Growth and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic.

Other names of actual companies and products mentioned herein are the trademarks of their respective owners.

Microsoft is a registered trademark of Microsoft Corporation in the United States and/or other countries.

This document discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Hardware Limited Warranty

Warranty for Hardware Products: Dialogic Corporation or its subsidiary that originally sold the hardware product ("Dialogic") warrants to the original purchaser of this hardware product, that at the time of delivery the hardware product supplied hereunder will be free from defects in material and workmanship. This warranty is for the standard period set out on Dialogic's website at http://www.dialogic.com/warranties and is void if the defect has resulted from accident, misuse, abuse or misapplication. Any hardware product which becomes defective during the warranty period and is returned by the original purchaser to Dialogic's Authorized Service Center with a Return Material Authorization (RMA) number (which must be obtained from Dialogic before any return) within thirty (30) days after discovery of the defect with a written description of the defect will be repaired or replaced at Dialogic's option. Freight charges will be paid by Dialogic only for shipment back to you.

Additional Exclusions: Dialogic will have no obligation to make repairs or replacements necessitated by your fault or negligence, improper or unauthorized use of the product, repairs or modifications made without Dialogic's prior written approval or by causes beyond the control of Dialogic, including, but not limited to, power or air conditioning failure, acts of God, improper interface with other units, or malfunction of any equipment or software used with the Dialogic product(s). If Dialogic is requested and agrees to make repairs or replacements necessitated by any such causes, you will pay for such service or replacement at Dialogic's then prevailing rates.

No Other Warranties: DIALOGIC DISCLAIMS AND YOU WAIVE ALL OTHER WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE AND ANY WARRANTY AGAINST LATENT DEFECTS, WITH RESPECT TO ANY DIALOGIC PRODUCT.

No Liability for Damages: IN NO EVENT SHALL DIALOGIC OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, INTERRUPTION OF ACTIVITIES, LOSS OF INFORMATION OR OTHER PECUNIARY LOSS AND DIRECT OR INDIRECT, CONSEQUENTIAL, INCIDENTAL, ECONOMIC OR PUNITIVE DAMAGES) ARISING OUT OF THE USE OF OR INABILITY TO USE ANY DIALOGIC PRODUCT.

Limitation of Liability: DIALOGIC'S MAXIMUM CUMULATIVE LIABILITY SHALL BE LIMITED TO THE AMOUNTS ACTUALLY PAID BY YOU TO DIALOGIC FOR THE SPECIFIC PRODUCT BEING THE OBJECT OF THE CLAIM. YOU RELEASE DIALOGIC FROM ALL AMOUNTS IN EXCESS OF THE LIMITATION. YOU ACKNOWLEDGE THAT THIS CONDITION IS ESSENTIAL AND THAT DIALOGIC WOULD NOT SUPPLY TO YOU IF IT WERE NOT INCLUDED.

Contents

List of Figures

List of Tables

About this Publication

The Dialogic® IP Media Server (which is also referred to herein as "IP Media Server" or "Media Server") is a standards-based SIP, VoiceXML, and MSCML server that performs a wide variety of media processing functions.

The IP Media Server is also is a cost-effective and scalable IP media option as it can power a broad range of voice and video services for next generation wireline, wireless, and broadband services.

This section describes this manual and the contents of the manual set and consists of the following sections:

Using this Publication

Audience and Purpose

This manual is for application developers who choose to use the Dialogic IP Media Server to in furtherance of deploying network announcements, conferences, and IVR in a Voice over IP (VoIP) environment.

Organization and Content

Chapter 1, "Introduction", introduces the Session Initiation Protocol (SIP) and XML-based APIs that allow the IP Media Server to perform in an IP network, explains the call-related resources, and defines the supported application services.

Chapter 2, "Session Initiation Protocol (SIP)", explains in detail the Session Initiation Protocol (SIP) methods, headers, and response codes that the IP Media Server currently supports.

Chapter 3, "Announcement Service API", covers the differences between simple and sequenced announcements and outlines the use of variables.

Chapter 4, "Conferencing API", covers simple and advanced conferencing and includes references to the Media Server Control Markup Language (MSCML) elements and attributes that enable advanced conferencing.

Chapter 5, "IVR with MSCML", explains Dialogic's implementation of Interactive Voice Response (IVR) using the ivr service indicator. This service indicator enhances SIP requests using MSCML message bodies for play, playcollect, and playrecord events.

Chapter 6, "Overview of VoiceXML 2.0", explains the basics of VoiceXML 2.0. For additional information about the supported VoiceXML 2.0 elements and attributes and ECMAScript language functionality, refer to the *VoiceXML 2.0 Reference Guide*. It is provided in HTML format on the IP Media Server documentation CD and provides descriptions of supported VoiceXML 2.0 tags and attributes.

Chapter 7, "Sample Code and Call Flows", provides examples and call flow diagrams for announcements.

Appendix A, "Audio Library", describes the IP Media Server's preconfigured sound library (audio files), which consists of phrases and messages, numbers, time and money, quantities, and miscellaneous words.

Appendix B, "VoiceXML Version 1.0 and Dialog Service", explains the basics of VoiceXML 1.0, and lists the supported VoiceXML 1.0 elements and attributes, and explains ECMAScript language functionality.

Appendix C, "MSCML Schema", provides the MSCML schema.

Appendix D, "RTP Codec Selection with VoiceXML <transfer/>", describes how the Real-Time Protocol (RTP) codec works with the transfer element.

Appendix E, "Using AMR-NB", describes how one can use the Adaptive Multi-Rate - Narrow Band (AMR-NB) codec with the IP Media Server.

Note: Using the AMR-NB resource in connection with a Dialogic® product does not grant the right to practice the AMR-NB standard. To seek a patent license agreement to practice the standard, contact the VoiceAge Corporation at http://www.voiceage.com/licensing.php. For a listing of the AMR codec "essential patents," visit: http://www.voiceage.com/ess_patent.php

---

Appendix F, "Dial Pulse Detection", describes the Dial Pulse Detection feature in the IP Media Server product.

Appendix G, "MRCP Session Management" describes how the IP Media Server can manage MRCP sessions, including a sample call flow.

Appendix H, "T.30 Fax Detection" describes how the IP Media Server can detect T.30 Fax CNG tones embedded in a G.711 RTP audio streams.

Appendix I, "T.38 Fax Detection, Termination, and Initiation" describes how the IP Media Server can receive a fax via T.38 transmissions carried in a G.711 audio RTP stream.

Dialogic® IP Media Server Documentation Set

The Dialogic® IP Media Server is documented in the following publications:

- The *Installation and Operations Guide* provides instructions for configuring, administering, and maintaining the IP Media Server.
- The *Application Developer's Guide* provides information for application developers who choose to use the IP Media Server to deploy network announcements, conferences, and Interactive Voice Response (IVR) in a voice over IP (VoIP) environment.
- *Installing Red Hat Enterprise Linuz 4.0 for the IP Media Server* describes how to install and configure Red Hat Enterprise Linux 4 if you are installing the licensed software version of the IP Media Server.
- The *License Activation Guide* describes how to activate the license for your Dialogic® IP Media Server.
- *Upgrading from Release 2.3.0 to 2.4.0 on Red Hat Enterprise Linux ES Platform* provides information and instructions for upgrading from IP Media Server Release 2.3.0 to IP Media Server Release 2.4.0 on platforms running Red Hat Enterprise Linux ES Platform. It also includes instructions for downgrading from 2.4.0 to 2.3.0 in the event that you need to restore your previous configuration.
- *VoiceXML 2.0 Reference* is provided in HTML format on the IP Media Server documentation CD and provides descriptions of supported VXML 2.0 tags and attributes.

Printed and Electronic Document Formats

The documentation package for the IP Media Server contains a printed copy of Release Notes and a CD containing electronic versions of the IP Media Server publications, Release Notes, hardware compliance documentation, and the Intel reference documentation, all in PDF format. The PDF files require the Adobe Acrobat reader, a free download from www.adobe.com.

Release notes are updated when certain software changes are made and are distributed by email and on the Dialogic Technical Support Web site: http://www.dialogic.com/support/.

Document Conventions

Notes, Cautions, and Warnings

Notes contain information of general interest, for example:

Cautions and warnings appear when appropriate throughout the manual.

Cautions alert you to situations that can make system administration less effective or compromise system performance or security. For example:

Before changing the configuration of a running system, always back up the current configuration using the System→Config Backups command.

Warnings alert you to situations that could cause physical harm to an operator or damage to the IP Media Server. For example:

If an interface is deactivated, all traffic on that interface will be dropped.

Links in PDF

Hypertext links in the PDF version of this manual use non-serif font. You can click on a cross reference to move to the information it references.

Index entries and Table of Contents listings are also clickable links in the PDF format. After you jump to a link, use the Back button on the Acrobat Reader toolbar to return to your prior location.

Contacting Dialogic Technical Support

For more information, refer to the Dialogic Technical Support site:

http://www.dialogic.com/support/

When reporting an issue to Technical Support, be prepared to provide the following information:

- Full description of the issue.
- Version of the IP Media Server software you are using.
- IP Media Server log files.
- Whether the issue is reproducible; the steps that you took.

Please note that the latest software update and release notes are available from the Dialogic Technical Support page.

Ordering Licenses

To use certain Dialogic® software products, you must have a license. To acquire applicable licenses, contact your Dialogic sales representative.

1 - Introduction

This chapter provides an overview of the features and functionality of the Dialogic® IP Media Server (which, as noted above, is also referred to in this document as "IP Media Server" or "Media Server").

This chapter includes the following sections:

Control Protocols and Services

This chapter introduces the Session Initiation Protocol (SIP) and XML-based APIs that allow the IP Media Server to perform in an IP network, explains the call-related resources, and defines the supported application services.

The IP Media Server can perform a specialized role in the IP network by providing high-capacity, real-time packet processing for network announcements, conferencing, and Interactive Voice Response (IVR) functions.

This chapter contains the following major sections:

Call Control Protocols

A control agent, such as a softswitch or an application server, can use a call control (signaling) protocol to request a service from the IP Media Server and to set up media streams between the IP Media Server and SIP-enabled endpoint devices. SIP (Session Initiation Protocol) is the signaling protocol currently used for call control on the IP Media Server.

In the course of signaling, the application server can perform the following:

- Requests the service
- Sets up the connection
- Modifies the connection, if necessary
- Tears down the connection

If such a connection is established, the IP Media Server is directly engaged with the endpoint devices in the media plane through third party call control. The IP Media Server, however, is not visible to the endpoint devices in the signaling plane.

Figure 1 illustrates the call control relationship.

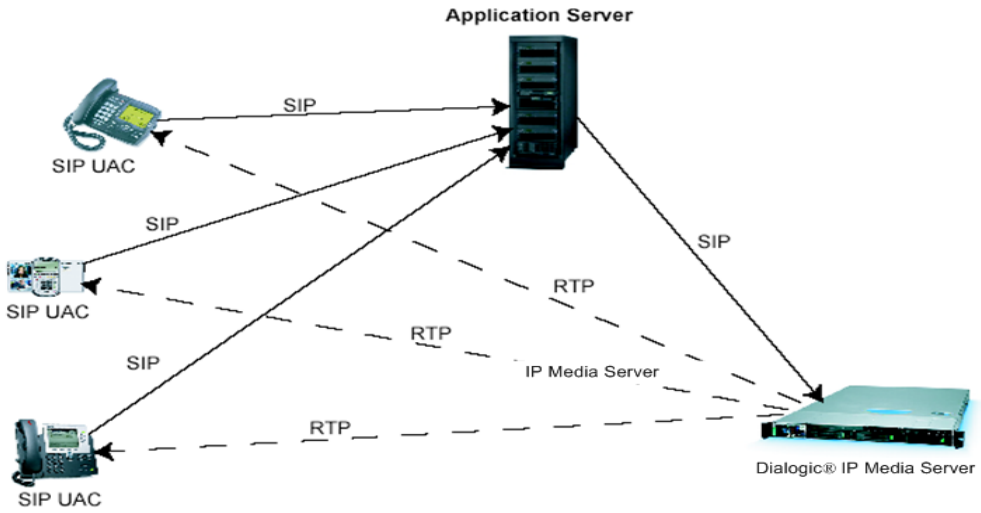

Figure 1. Call Control Architecture

The external application specifies what constitutes a call, not the IP Media Server. Upon receiving a request, the IP Media Server call control engages the correct application for processing. Control Protocols and Scripting Language

The IP Media Server is controlled by the following major control protocols and scripting languages:

- Session Initiation Protocol (SIP)
- Session Description Protocol (SDP)
- Media Server Control Markup Language (MSCML)

- VoiceXML

Session Initiation Protocol (SIP)

SIP is a peer-to-peer signaling protocol that employs a request and response model to create, manage, and tear down single or multiple concurrent sessions for the delivery of services.

Session

A session is defined as having at least two endpoints with one or more bi-directional IP streams to, from, or within the IP Media Server.

A control agent can initiate and control the following services with SIP messages:

- Network Announcements
- Conferences
- IVR
- VoiceXML

Service Indicators

The IP Media Server employs a service indicator in the user address portion of the Request URI. This indicator specifies the service to which a request is being addressed and can be one of the values listed in Table 1.

Table 1. SIP Service Indicators

Service Indicator	Definition
annc	Announcements
conf	Conferencing (simple and advanced with MSCML)
ivr	IVR with MSCML
dialog	VoiceXML

If the user address portion of the service indicator is not one of the supported service indicators, the default service, dialog, is invoked. You can configure the default application in the SIP parameters section of the MEDIA SERVER > SIP menu page on the Web UI.

The IP Media Server takes advantage of the inherent flexibility provided by the SIP URI conventions and by SIP's acceptance of message-body payloads. For example, when SIP is extended with the MSCML scripting language, SIP requests enable control of advanced conferences as well as Interactive Voice Response (IVR) through the ivr service.

For details concerning SIP, see Chapter 2, "Session Initiation Protocol (SIP)".

Session Description Protocol (SDP)

SIP uses the Session Description Protocol (SDP) to convey information about media streams. The IP Media Server supports parsing and formatting for SDP mandatory elements, as well as several optional elements.

For additional details concerning SDP, see "Session Description Protocol" (page 71).

Media Server Control Markup Language (MSCML)

Media Server Control Markup Language (MSCML) is a Dialogic-developed markup language used to extend SIP requests for IVR and advanced conferencing functions.

There are six types of MSCML requests which are grouped into two basic categories:

- Requests that support advanced conferencing (conf service)
- Requests that support interactive voice response (ivr service)

Table 2 lists both the MSCML conferencing requests and the MSCML IVR requests, and also requests the pages where these requests are defined.

Table 2. MSCML Conferencing and MSCML IVR Requests

MSCML Conferencing	MSCML IVR Requests
"configure_conference" (page 126)	"play" (page 154)
"configure_leg" (page 128)	"playcollect" (page 154)
	"playrecord" (page 155)
	"stop" (page 155)

For details concerning these MSCML requests and services, see Chapter 4, "Conferencing API".

SIP Methods with MSCML

The SIP `INVITE` method is used to convey the desired session parameters from the application server to the IP Media Server.

For mid-call changes, the XML payload is sent using SIP `INFO`.

Note: To provide that XML payloads are delivered in the correct order, the application server must support sequenced INFO messages for MSCML (see "CSeq" (page 63)).

Payload

The size of the MSCML payload is relatively small and is appropriate for inclusion in the SIP body. Each MSCML payload contains a single request or response, and each SIP `INVITE` or `INFO` carries, at most, one MSCML body part.

Most MSCML request attributes have default values or are defined as `#IMPLIED`. MSCML attributes can be omitted from the request if they are not needed.

For further details, see "MSCML Schema" (page 295).

The Multi-Internet Message Extension (MIME) type used to describe MSCML content is application/mediaservercontrol+xml.

Responses

MSCML responses are carried back to the application server in the body of the SIP request message. Responses to MSCML requests are also defined in the Document Type Description (DTD) using a simple form of XML. The response is easy to process and does not require a full XML parser on the application server.

Events

Applications subscribe to activetalker and asynchronous Dual Tone Multi-Frequency (DTMF) event notifications through MSCML directives. Events are reported within the SIP session that subscribed to the event.

Note: For details on creating advanced conferences with MSCML, see Chapter 4, "Conferencing API" and Chapter 5, "IVR with MSCML".

VoiceXML

Voice Extensible Markup Language (VoiceXML) is designed specifically for speech-based telephony and video applications. VoiceXML audio dialog scripts can include a mix of digitized audio clips and DTMF inputs.

A SIP `INVITE` request directed to the dialog service directs the VoiceXML interpreter to retrieve a URI-specified VoiceXML script and translate the script-directed commands.

The IP Media Server supports VoiceXML 1.0 and VoiceXML 2.0. For information about VoiceXML 2.0 tags and attributes, refer to the *VoiceXML 2.0 Reference*, included on the IP Media Server documentation CD.

The following table provides the IP Media Server functionality supported by VoiceXML 1.0 and VoiceXML 2.0.

Table 3. Functionality Supported by VoiceXML 1.0 and VoiceXML 2.0

VoiceXML1.0	VoiceXML 2.0
T.30 Fax Detection	MRCP 1.0 for Automated Speech Recognition (ASR) support
T.38 Fax Detection, Termination, and Initiation	MRCP 1.0 for Text to Speech (TTS) support

For further details concerning VoiceXML 1.0, see "VoiceXML Version 1.0 and Dialog Service" (page 247).

Media Storage, Processing, and Supported Codecs

The IP Media Server supports several audio and video codecs, and stores and retrieves content in several encodings and file formats. These supported codecs are summarized in Table 4.

Table 4. Supported Audio/Video Codecs

Codec	Audio/Video	RTP Payload Type	Bandwidth (Kb/s)	Supported Packet Time (ms)
AMR-NB	Audio	96...127	<table><tr><th>Mode</th><th>Bit Rate</th></tr><tr><td>0</td><td>4.75</td></tr><tr><td>1</td><td>5.15</td></tr><tr><td>2</td><td>5.9</td></tr><tr><td>3</td><td>6.7</td></tr><tr><td>4</td><td>7.4</td></tr><tr><td>5</td><td>7.95</td></tr><tr><td>6</td><td>10.2</td></tr><tr><td>7</td><td>12.2</td></tr></table>	20, 40
G.711 ulaw	Audio	0	64	10, 20, 30
G.711 alaw	Audio	8	64	10, 20, 30
G.726	Audio	2	32	10, 20, 30
G.729AB	Audio	18	8	10, 20, 30
H.263	Video	34	Various	N/A
H.263-1998	Video	96...127	Various	N/A
H.263-2000	Video	96...127	Various	N/A
H.264	Video	96...127	Various	N/A

Audio

For all services, the IP Media Server can receive and transmit RTP voice packets encoded as G.711μ-law, G.711a-law, G.726, G.729AB, and AMR-NB. See Table 4 (page 33).

Note: G.726, G.729AB, and AMR-NB codecs require an optional EdgeMedia EDP-10 DSP card. If you are planning to use one of these codecs, please inform your sales representative when you order your IP Media Server or IP Media Server software.

Note: Using the AMR-NB resource in connection with a Dialogic® product does not grant the right to practice the AMR-NB standard. To seek a patent license agreement to practice the standard, contact the VoiceAge Corporation at http://www.voiceage.com/licensing.php. For a listing of the AMR codec "essential patents," visit: http://www.voiceage.com/ess_patent.php

G.729AB

The IP Media Server supports G.729AB as an RTP codec.

The IP Media Server automatically converts (transcodes) stored content accessed via file:/// or http:// URLs from the content encoding to the RTP encoding. Any of the available content encodings can be played or recorded using any of the available RTP codecs.

G.726

The IP Media Server supports G.726 at 32kbps as an RTP codec.

AMR-NB

The IP Media Server supports AMR-NB as an RTP codec. AMR-NB (Adaptive Multi-Rate-Narrow Band) is a multi-mode codec that supports 8 narrow-band speech encoding modes, with bit rates between 4.75 and 12.2 Kbps.

Content Storage

The IP Media Server can retrieve and play audio files with content encoded as G.711 a-law/μ-law in the *.au, *.ulaw, *.alaw, and *.wav data formats. Raw (headerless) content is represented as *.ulaw and *.alaw. Content can also be encoded as Microsoft Global System Mobile (MSGSM) in *.wav format, with raw (headerless) content represented as *.msgsm or *.ms_gsm.

Table 5. G.711 Encoding and File Storage Formats

Content Encoding	File Storage Formats
G.711 μ-law	raw, .wav, .au
G.711 a-law	raw, .wav, .au
MSGSM	raw, .wav
3GPP (audio only)	.3gp

Note: If the file format is unknown or unspecified, the Dialogic® IP Media Server assumes headerless μ-law.

Video

The IP Media Server can transmit and receive H.263, H.263+, and H.264 RTP video packets. Video is currently supported only by the conferencing (conf) and VoiceXML (dialog) services.

Mixed Audio/Video

video/x-wav

Mixed audio and video content is supported using a proprietary WAV-based format identified by the MIME type video/x-wav. Files of this type can contain a G.711 audio track and an H.263, H.263+, or H.264 video track. This content type is currently only supported by the dialog service (VoiceXML).

3GPP

The IP Media Server also supports playback of an associated video and audio stream from the 3rd-Generation Partnership Project (3GPP) file format, provided that the video stream is stored in a track as H.263 or H.264 frames and the audio stream is stored in a separate track as AMR-encoded frames.

Note: Using the AMR-NB resource in connection with a Dialogic® product does not grant the right to practice the AMR-NB standard. To seek a patent license agreement to practice the standard, contact the VoiceAge Corporation at http://www.voiceage.com/licensing.php. For a listing of the AMR codec "essential patents," visit: http://www.voiceage.com/ess_patent.php

---

On playback, the IP Media Server packetizes the H.263 or H.264 video frames into RTP streams, and the audio stream into RTP packets in one of the supported RTP encodings (G.711, G.726, G.729, AMR-NB). Note that the IP Media Server does not send/receive RTP/AMR packets as part of the 3GPP file support feature unless you have installed the optional EdgeMedia EDP-10 DSP card.

The following table shows the encodings that are currently supported for storage and transmission by the IP Media Server for the 3GPP file format.

Table 6. 3GPP Supported Encodings and Streams

Inbound or Outbound Network Stream	Encoding of Stream in File	Supported (Yes/No)
RTP/AMR	AMR	Yes (with EDP-10)
RTP/G.711	AMR	Yes
RTP/G.726	AMR	Yes (with EDP-10)
RTP/G.729	AMR	Yes (with EDP-10)
RTP/H.263	H263	Yes
RTP/H.263-1998	H263-1998	Yes
RTP/H.263-2000	H263-2000	Yes
RTP/H.264	H264	Yes

The 3GPP file format uses the AMR codec for speech encoding and H.263 codec for video encoding. This format is currently supported only by the dialog service (VoiceXML).

File Storage and Retrieval

The IP Media Server can process files using either NFS or HTTP.

NFS

The IP Media Server can retrieve, plays, stores, and records data files using the Network File System (NFS) protocol (RFC 1094 and RFC 1813).

HTTP

The IP Media Server can retrieve and play data files using HTTP/1.0 (RFC 1945) and HTTP/1.1 (RFC 2068) servers.

DTMF

The Dialogic® IP Media Server supports the in-band tone detection of the DTMF digit set (0-9, #, *, A-Z), as defined in ITU recommendations Q.23 and Q.24, and according to RFC 2833, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals".

For further information concerning DTMF and its use with MSCML, see “IVR Operations on Participant Legs” (page 118).

In-Band Busy Tone Detection and Reporting

The IP Media Server supports in-band busy tone detection and reporting. This feature allows you to monitor outbound calls that can terminate on a PBX.

Services

The IP Media Server supports the following SIP services:

- Network Announcements
- Conferences
- IVR
- Dialog (VoiceXML)

Network Announcements

Announcement play-out happens when the IP Media Server retrieves stored media and plays it out in an RTP audio stream.

Announcement files are stored at a location accessible by the IP Media Server. The SIP `INVITE` message that plays an announcement includes the path for the audio file in the `play=` parameter. For example:

```
INVITE sip:annc@MS_IP;
   play=file:////opt/snowshore/prompts/generic/
   en_us/susan/circuit_busy.ulaw SIP/2.0
```

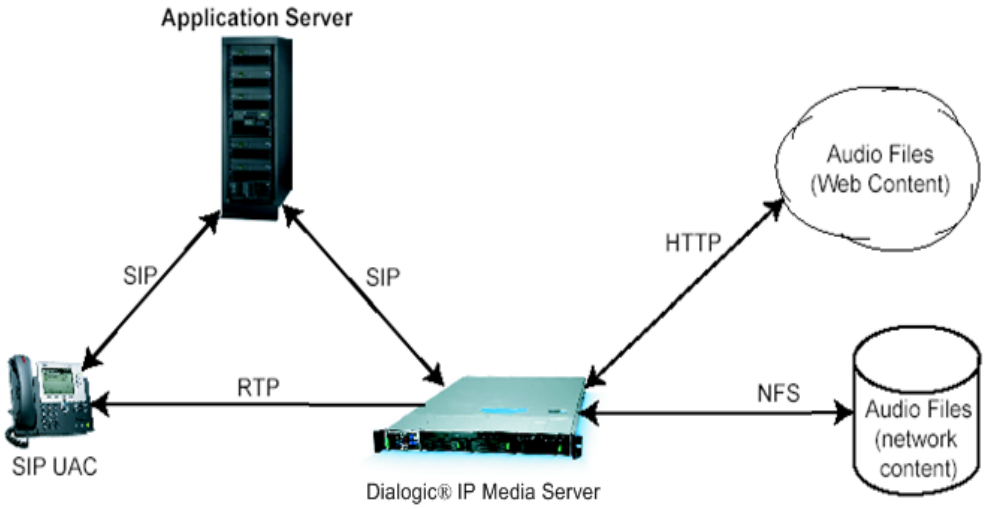

Figure 2. Network Announcements

The IP Media Server is factory-equipped with a library of generic audio segments (see "Audio Library" (page 233)).

Simple Announcements

Simple announcements are audio files of fixed content that require no user interaction. (For example, a message such as: “All circuits are busy. Please try your call again later.”)

Simple announcements can be stored in a location that is accessible to the IP Media Server and can be in either the `file:////` scheme retrieved by NFS or the `http://` scheme retrieved by HTTP.

You can use the Web UI to configure a default location for audio files that play as simple announcements. Once you configure this setting, the IP Media Server prepends it to the play= parameter in the SIP message when there is no scheme (`file` or `http`) specified between play= and the filename.

For example, if you set your default URL to this:

```
file:////opt/snowshore/prompts/generic/
```

then the IP Media Server interprets this request:

```
INVITE sip:annc@MS_IP;play=circuit_busy.ulaw SIP/2.0
```

as:

```
INVITE
   sip:annc@MS_IP;play=file:////opt/snowshore/prompts/
   generic/circuit_busy.ulaw SIP/2.0
```

Announcement Sequences and Variable Content Announcements

Announcement sequences are multiple audio files delivered in a single play-out. Each portion of the sequence is a separate file that can be reused in another sequence. In the following example, each sentence can be separate file that can also be used in other sequences:

”Thank you for calling Media Services. The office is currently closed. Our normal business hours are Monday through Saturday, 10AM to 5PM.”

Sequences can also contain variables that are evaluated at run-time. For example, the series of variables

<num_dial.wav>, <var1> <changed.wav> <new_num.wav> <var2>

becomes

“The number you have dialed, 555-122-2222, has been changed. The new number is 555-122-3333.”

Conferences

Conferences are categorized in the following way:

- Simple
- Advanced

Conferencing requires that two or more RTP streams are mixed so that speakers are heard in the output RTP stream. Input and output channels may be selected from any supported RTP codec. The number of input channels per session ranges from two to the available IP Media Server capacity. The output channels generate a mix representing the inputs for a maximum of three active speakers.

The conference sessions mix the input media streams, creating a set of output streams of mixed media. Advanced conference sessions additionally allow operations on individual inputs and outputs to and from the conference mixer. These operations include:

- Tone detection
- Tone generation
- Tone suppression

Simple Conferences

The SIP interface supports simple conferencing with no control or special mixing directives. Standard SIP methods create the conference, manage participant access, and delete the conference. Based on IP Media Server capacity, participants can listen and talk, with a maximum of three active talkers.

The number of participants per simple conference is dynamically allocated.

The URI in the SIP `INVITE` contains a conference service indicator (`conf`) with a unique instance identifier. For example:

```
sip:conf=conference1@MS_IP SIP/2.0
```

The conference service creates a conference for the first instance of the unique service instance identifier. The IP Media Server places subsequent requests with the same service instance identifier into that conference.

Advanced Conferences

Enhanced SIP (inclusion of MSCML), supports the following:

- Advanced conferencing with call control
- IVR prompting
- Mixing directives
- Event notification

For example, you can set parameters on a per-conference basis to record the conference and play announcements to all participants.

IVR

The IVR service supports interactive voice response applications with functions for playing prompts, collecting DTMF digits, and recording. This service uses Media Server Control Markup Language (MSCML) to enable application control of these functions.

Dialog (VoiceXML)

The dialog service supports interactive voice response and video applications written in VoiceXML. VoiceXML 1.0 and 2.0 are supported on the IP Media Server.

Refer to "VoiceXML Version 1.0 and Dialog Service" (page 247) for information about VoiceXML 1.0.

Refer to the *VoiceXML 2.0 Reference* for information about VoiceXML 2.0. It is provided in HTML format on the IP Media Server documentation CD and provides descriptions of all supported VoiceXML 2.0 tags and attributes.

2 - Session Initiation Protocol (SIP)

This chapter explains in detail the Session Initiation Protocol (SIP) methods, headers, and response codes that the IP Media Server currently supports.

This chapter covers the following major sections:

Conformance

The IP Media Server currently conforms to the following SIP and SDP (Session Description Protocol) specifications:

[1] RFC 2327, “SDP: Session Description Protocol”,
M. Handley et al., April 1998.

[2] RFC 2543, “SIP: Session Initiation Protocol”,
M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg, March 1999.

[3] RFC 2543bis-03 “SIP: Session Initiation Protocol”,
M. Handley, H Schulzrinne, E. Schooler, J. Rosenberg, March 1999. (Partial conformance for methods, headers, and return codes, as indicated on pages 52, 60, and 67 respectively.)

[4] RFC 2976, “The SIP INFO Method”,
S. Donovan, October 2000.

[5] draft-ietf-sip-100rel-03, “Reliability of Provisional Responses in SIP”,
J. Rosenberg, H. Schulzrinne, March 2001, work in progress.

[6] draft-ietf-sip-session-timer-04, “SIP Session Timer”,
S. Donovan, J. Rosenberg, November 2000.

[7] draft-ietf-sipping-netann-06, “Basic Network Media Services with SIP”,
Burger, E. and Van Dyke, J., July 2003, work in progress.

[8] RFC 1341, “MIME (Multipurpose Internet Mail Extensions)”,
Borenstein and Freed, June 1992.

[9]RFC 4722, “Media Server Control Markup Language (MSCML) and Protocol”,
Van Dyke, J., Burger, E., November 2006.

[10] RFC 2833, “RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals”,
H. Schulzrinne, S. Petrack, May 2000.

Media Server Availability

The IP Media Server supports SIP OPTIONS, which allows an endpoint to query the supported methods, content types, extensions, and codecs without establishing a call with the IP Media Server. The IP Media Server extends the SIP OPTIONS response to return IP Media Server specific process availability as well as a snapshot of the current application sessions in use on the IP Media Server. This information is made available when endpoint includes in the SIP OPTIONS request a Content-Type specific only for the IP Media Server.

The IP Media Server support a specific Content-Type in a SIP OPTIONS request that instructs the IP Media Server to return a set of Media Server Availability information in one XML response from the IP Media Server. This information will be in two Content Types:

- Licensed Features on the IP Media Server and usage statistics
- IP Media Server Process Table

Refer to OPTIONS on page 56 for more information on the OPTIONS method and Content-Type (c) on page 63 for more information on this Content-Type.

Example

The following is an example of a SIP OPTIONSrequest soliciting the application/media_server_usage+xml information from an IP Media Server. The response follows the request below.

Request

```
OPTIONS sip:192.168.12.160 SIP/2.0
Via: SIP/2.0/UDP 10.128.41.35:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Hammer001 <sip:Hammer001>;tag=9fxced76sl
To: <sip:192.168.12.160>
Call-ID: 3848276298220188511
CSeq: 1 OPTIONS
Accept: application/media_server_usage+xml
Content-Length: 0
```

Response

Note: The SIP OPTIONS response will be sent using UDP and will exceed the 1500 byte limit referenced in RFC 3261 - SIP: Session Initiation Protocol.

```
SIP/2.0 200 Ok
Contact: sip:192.168.12.160:5060
Server: SnowShoreMediaServer/0.0.0-0
Via: SIP/2.0/UDP 10.128.41.35:5060;branch=z9hG4bK74bf9
To: <sip:192.168.12.160>
From: Hammer001 <sip:Hammer001>;tag=9fxced76sl
Call-ID: 3848276298220188511
CSeq: 1 OPTIONS
Allow: OPTIONS, INVITE, ACK, INFO, BYE
Accept: application/sdp, multipart/mixed,
   application/mediaservercontrol+xml,
   application/media_server_usage+xml
Supported: timer, 100rel
Content-Type: application/media_server_usage+xml
Content-Length: 3497


<?xml version="1.0" encoding="iso-8859-1"
   ?><media_server_usage><features><feature
   name="ports" current="0" peak="0" total="750"
   /><feature name="mrcp" current="0" peak="0"
   total="400" /><feature name="g726" current="0"
   peak="0" total="300" /><feature name="g729"
   current="0" peak="0" total="300" /><feature
   name="h263" current="0" peak="0" total="150"
   /><feature name="h264" current="0" peak="0"
   total="100" /></features><processes><process
   command="fido_start" status="S" parent_pid="18978"
   pid="19020"/><process command="vxmld_start"
   status="S" parent_pid="18981" pid="19021"/><process
   command="sipd" status="S" parent_pid="18983"
   pid="19026"/><process command="logServ" status="S"
   parent_pid="18977" pid="19057"/><process
   command="monitord" status="S" parent_pid="18984"
   pid="19106"/><process command="vxmld" status="S"
   parent_pid="19021" pid="19138"/><process
   command="logI" status="S" parent_pid="19021"
   pid="19139"/><process command="msinit" status="S"
   parent_pid="18980" pid="19160"/><process
   command="recoveryd" status="S" parent_pid="18985"
   pid="19192"/><process command="uad" status="S"
   parent_pid="18982" pid="19216"/><process
   command="msprovider" status="S" parent_pid="19160"
   pid="19238"/><process command="fido" status="S"
   parent_pid="19020" pid="19239"/><process
   command="logI" status="S" parent_pid="19020"
   pid="19240"/><process command="fido" status="S"
```

```
parent_pid="19239" pid="19241"/><process
command="fido" status="S" parent_pid="19239"
pid="19244"/><process command="fido" status="S"
parent_pid="19239" pid="19245"/><process
command="fido" status="S" parent_pid="19239"
pid="19246"/><process command="fido" status="S"
parent_pid="19239" pid="19248"/><process
command="fido" status="S" parent_pid="19239"
pid="19250"/><process command="fido" status="S"
parent_pid="19239" pid="19252"/><process
command="fido" status="S" parent_pid="19239"
pid="19254"/><process command="fido" status="S"
parent_pid="19239" pid="19255"/><process
command="fido" status="S" parent_pid="19239"
pid="19257"/><process command="fido" status="S"
parent_pid="19239" pid="19263"/><process
command="fido" status="S" parent_pid="19239"
pid="19266"/><process command="fido" status="S"
parent_pid="19239" pid="19267"/><process
command="fido" status="S" parent_pid="19239"
pid="19270"/><process command="fido" status="S"
parent_pid="19239" pid="19271"/><process
command="vxmld" status="S" parent_pid="19138"
pid="19289"/><process command="vxmld" status="S"
parent_pid="19138" pid="19290"/><process
command="vxmld" status="S" parent_pid="19138"
pid="19292"/><process command="mserv" status="S"
parent_pid="19160" pid="20114"/><process
command="dms_start" status="S" parent_pid="18979"
pid="20148"/><process command="dms" status="S"
parent_pid="20148" pid="20168"/><process
command="logI" status="S" parent_pid="20148"
pid="20169"/><process command="ps" status="R"
parent_pid="19238"
pid="20950"/></processes></media_server_usage>
```

SIP Description

SIP (Session Initiation Protocol) is an IP telephony signaling protocol developed by the IETF. Generally used for voice over IP (VoIP) calls, SIP can also be used for video or any media type; for example, SIP has been used to set up multi-player games.

SIP is a text-based protocol based on HTTP and MIME, which makes it suitable for integrated voice-data applications. Its addressing scheme uses URLs and is human readable; for example: `sip:john.doe@company.com`.

SIP relies on the Session Description Protocol (SDP) for session description and the Real-time Transport Protocol (RTP) for actual transport.

SIP handles the initiation, modification, and teardown of multimedia sessions through the transmission of messages between clients and servers. Messages specify the network path and session description and can also include other payloads, and expected or required options.

SIP allows for the inclusion of parameters and encoded documents in the INVITEmessage as well as content in the message body. See "SIP Message Body Types" (page 49) for the list of accepted message body types supported in this release.

SIP User Agents

SIP User Agents are participants in a SIP session.

- A User-Agent Client (UAC) is a logical entity that issues requests and waits for responses.
- A User-Agent Server (UAS) is a logical entity that listens for and responds to a SIP request by either accepting or rejecting the request.

UAC and UAS functions change during the course of a call. For example, when playing an announcement, the application server acts as the UAS when it is directly engaged in SIP signaling with UACs such as SIP phones that use media services.

SIP Message

A SIP message is either a request from a UAC to a UAS identifying a desired service and providing a context for interpreting the request, or a response (status message) from the UAS to the UAC.

Messages employ a text-based protocol, using ISO 10646 character set in UTF-8 encoding. Unless otherwise stated, parameter names and values are case-insensitive.

SIP Transaction

A SIP transaction consists of the messages from the first request sent by the UAC to the corresponding, final response sent from the UAS. A sequence number (for example, CSeq:2) defines the transactions.

URLs

Universal Resource Locators (URLs) are used in SIP messages to indicate the originator (`From`), the routed destination (Request-URI), and final recipient (`To`) of a SIP request, and also to specify redirection addresses (`Contact`).

Both dotted IP addresses and fully-qualified domain names are accepted network addresses.

A SIP URL follows the guidelines of RFC 2396 and takes a form similar to a mail to or Telnet URL; for example, user@host.

Service Indicators

Service indicators in the initial SIP INVITE request direct how the IP Media Server processes the request.

In the context of SIP control, the IP Media Server substitutes a service indicator for the user component on the left-hand side of the URI. For example, a SIP request for an announcement takes the form sip:annc@MS_IP.

The IP Media Server, upon receiving the INVITE, notes the service indicator and interprets the request accordingly.

Table 7 lists the SIP services and associated indicators.

Table 7. Application Service Indicators

Service	Indicator	Example*	Page
Announcements	annc	`INVITE sip:`**`annc`**`@MS_IP;` `play=(etc.) SIP/2.0`	84
Conferencing	conf	`INVITE sip:`**`conf=`**`confid@MS_IP` `SIP/2.0`	98
IVR	ivr	`INFO sip:`**`ivr`**`@MS_IP SIP/2.0`	85

VoiceXML	dialog	`INVITE sip:`**`dialog`**`@MS_IP;`
		`voicexml=http://path/`
		`filename.vxml SIP/2.0`
			85 and 170

Default Application Service

If no service indicator appears in the SIP message, the default application is VoiceXML (the dialog service indicator). You can change this default application through the web interface as follows:

1 Select MEDIA SERVER > SIP menu

2 Set the Default Application under the SIP Parameters section.

3 The default version of VXML is 2.0 which you can configure from the MEDIA SERVER > VOICEXML menu.

SIP Message Body Types

Table 8 lists the valid message body content types.

Table 8. Valid Content (MIME) Types

Service / Return Type	Content Type
All services	application/sdp
Advanced conferences (conf= with MSCML) ivr service	application/mediaservercontrol+xml
Advanced conferences (conf= with MSCML)	multipart/mixed
Two pieces to the Return Type: Media Server Licensed Feature Set Media Server Processes and Status	application/media_server_usage+xml

The maximum inbound SIP message size is 2900 bytes. This size prevents buffer overflow caused by proprietary SIP message headers used by some third-party network elements.

SIP Requests

A SIP Request begins with a request line, followed by one or more headers. An empty line [a carriage return line feed (CRLF) followed by another carriage return line feed] marks the end of the header lines, and can be followed by an optional message body, which can contain a Session Description Protocol (SDP) body.

Example 1. SIP Message

SIP Request

```
INVITE sip:annc@192.168.12.155:5060;
play=file:////opt/snowshore/prompts/generic/
   circuit_busy.ulaw;early=no SIP/2.0
```

SIP Headers

```
Via: SIP/2.0/UDP 192.168.1.150:6100
To: <sip:annc@192.168.12.155:5060>
From: <sip:test0@192.168.12.153:5060>
Call-ID: 27103@192.168.1.150
Contact: sip:192.168.1.150:6100
CSeq: 2 INVITE
Content-Type: application/sdp
Supported: timer
Supported: 100rel
Session-Expires: 60
Content-Length: 15
```

Message Body

```
v=0
o=SnowShoreUaV1 14250 3757 IN IP4 192.168.1.150
s=SnowShore Sdp
t=0 0
m=audio 6000/1 RTP/AVP 0
c=IN IP4 192.168.12.154
a=sendrecv
a=ptime:10
```

Request Line

The request line consists of the following elements:

- Method, encoded by ASCII characters, which describes the action requested (`INVITE`, shown in the above example)
- Request-URI, which indicates the user or service receiving the request (sip:annc@192.168.12.155:5060, shown in the example)
- SIP version (SIP/2.0)
- Parameters and attributes (such as play= and early =), shown in the following example.

Example 2. SIP Parameters and Attributes

```
INVITE sip:annc@192.168.12.155:5060;
   play=file:///opt/snowshore/prompts/generic/en_us/
     susan/circuit_busy.ulaw;
   early=no SIP/2.0
```

Headers

A SIP header consists of the header name, followed by a colon, followed by a space and the header value, for example, `Supported: 100rel`.

The are four types of SIP headers:

- General
- Request
- Response
- Entity

The order of headers is not significant in the SIP Request. See "Session Description Headers" (page 71) for more information on SDP headers.

Session Descriptor

The session descriptor is encoded in conformance with the Session Description Protocol (RFC 2327) and contains a value or a range of values, such as a media description field and the connection-type attribute.

For details concerning the Session Description Protocol, see "Session Description Protocol" (page 71).

SIP Methods

The IP Media Server supports five methods from the SIP core protocol, as well as two methods—INFO and PRACK—that have been defined as extensions. Table 9 lists the SIP methods supported by the IP Media Server.

Table 9. Supported SIP Methods

Method	Description	Page
INVITE	Establishes a session or modifies a session.	52
ACK	Acknowledges final responses to INVITE requests.	54
CANCEL	Cancels a pending INVITE that terminates an unestablished call.	55
OPTIONS	Queries the IP Media Server on capabilities and current availability.	56
BYE	Ends a session.	57
INFO	Used for mid-call application requests, responses, and events. (RFC 2976, “SIP INFO Method”)	57
PRACK	Acknowledges receipt of a provisional response. (draft-ietf-sip-100rel-03, “Reliability of Provisional Responses in SIP”)	59

These methods are described in the following sections, including a list of Dialogic-supported headers for each one. (The headers are described in “SIP Headers” (page 60).)

INVITE

The INVITE method initiates or modifies a session. For example, an INVITE is sent to the IP Media Server requesting that the server participate in a call. An INVITE is followed by an ACK.

SIP INVITEs can also be initiated by the IP Media Server by a VoiceXML script. The IP Media Server typically does not initiate SIP INVITEs, but can act as a refresher for SIP session timers. When the IP Media Server acts as a refresher (user agent client), it sends session timer update messages during a SIP session. (The only other exception is VoiceXML <transfer/>.) For details on the Session Timer, refer to “Basic Network Media Services with SIP” listed in “Conformance” (page 43).

Note: The IP Media Server supports session timers in compliance with “Basic Network Media Services with SIP”. There is no session timer if early=yes, as a stable session is not established.

The INVITE can include the SDP offer. If not present in the INVITE, the IP Media Server provides the SDP in the final response and the client must answer in the ACK.

A re-INVITE renegotiates the media capabilities of an existing session and supports session timers. The re-INVITE must have a higher CSeq than any previous request from the UAC to the IP Media Server.

See the SIP request parameters "SIP Request Parameters for Announcements" (page 92) for more information.

Headers in an INVITE

Table 10 lists the mandatory and optional headers in a SIP INVITE method.

Table 10. SIP Headers in an INVITE

	Header	Page
Mandatory	Call-ID (i)	62
	CSeq	63
	From (f)	63
	To (t)	65
	Via (v)	66
	Contact (m)	62
Optional	Accept	61
	Content-Length (l)	62
	Record-Route	64
	Require	64
	Route	64
	Session-Expires	64
	Subject	65
	Supported	65
	Unsupported	66

ACK

The ACK method acknowledges the receipt of a final response to an INVITE. An ACK functions as follows:

- Used only with INVITE requests
- Issued by the party that sent the INVITE
- Does not generate a response

The ACK request has the same CSeq number as the corresponding INVITE request but comprises a transaction of its own.

The ACK can include SDP information only if the INVITE request did not. A UAC must not send an updated session description in an ACK response, if it has already sent one in the INVITE request.

Headers in an ACK

Table 11 lists the mandatory and optional headers in a SIP ACK method.

Table 11. SIP Headers in an ACK Method

	Header	Page
Mandatory	Call-ID (i)	62
	CSeq	63
	From (f)	63
	To (t)	65
	Via (v)	66
Optional	Content-Length (l)	62
	Content-Type (c)	63
	Record-Route	64
	Require	64
	Route	64
	Supported	65
	Unsupported	66

CANCEL

The CANCEL method is a request from the UAC to cancel a pending request with the same Call-ID, To, From, Via header, Request-URI, and CSeq (sequence number only). The IP Media Server ignores a CANCEL request if it arrives after the final response has been sent.

Headers in CANCEL

Table 12 lists the mandatory and optional headers in a SIP CANCEL method.

Table 12. SIP Headers in CANCEL Method

	Header	Page
Mandatory	Call-ID (i)	62
	CSeq	63
	From (f)	63
	To (t)	65
	Via (v)	66
Optional	Accept	61
	Content-Length (l)	62
	Record-Route	64
	Require	64
	Supported	65
	Unsupported	66

OPTIONS

The OPTIONS method is a request from the UAC for information from the IP Media Server about the SIP methods supported. The response contains supported methods, content types, and extensions.

Headers in OPTIONS

Table 13 lists the mandatory and optional headers in a SIP OPTIONS method.

Table 13. OPTIONS Method Headers

	Header	Page
Mandatory	Call-ID (i)	62
	CSeq	63
	From (f)	63
	To (t)	65
	Via (v)	66
Optional	Accept	61
	Content-Length (l)	62
	Record-Route	64
	Require	64
	Supported	65
	Unsupported	66

BYE

The BYE method terminates a session by indicating that either the application server or the IP Media Server wants to end the connection.

Note: BYEs are sent only when a stable session exists.

Headers in BYE

Table 14 lists the mandatory and optional headers in a SIP BYE method.

Table 14. BYE Method Headers

	Header	Page
Mandatory	Call-ID (i)	62
	CSeq	63
	From (f)	63
	To (t)	65
	Via (v)	66
Optional	Accept	61
	Content-Length (l)	62
	Content-Type (c)	63
	Record-Route	64
	Require	64
	Route	64
	Supported	65
	Unsupported	66

INFO

The INFO method is used during a session to convey application-specific information and events and to communicate mid-session signaling information that does not affect session state.

Headers in INFO

Table 15 lists the mandatory and optional headers in a SIP INFO method.

Table 15. INFO Method Headers

	Header	Page
Mandatory	Call-ID (i)	62
	CSeq	63
	From (f)	63
	To (t)	65
	Via (v)	66
Optional	Accept	61
	Content-Length (l)	62
	Content-Type (c)	63
	Record-Route	64
	Require	64
	Supported	65
	Unsupported	66

PRACK

The PRACK method acknowledges receipt of a provisional response. Like ACK, it is used only with INVITE requests and is issued by the party that sent the INVITE. The IP Media Server does not issue this request, but responds with 200 OK to UACs who require or support [5] draft-ietf-sip-100rel-04, "Reliability of Provisional Responses in SIP".

Figure 3 (page 80) shows a call flow for a provisional response.

Headers in a PRACK

Table 16 lists the SIP headers in a PRACK Method.

Table 16. Headers in PRACK Method

	Header	Page
Mandatory	Call-ID (i)	62
	CSeq	63
	From (f)	63
	To (t)	65
	Via (v)	66
Optional	Accept	61
	Content-Length (l)	62
	Record-Route	64
	Require	64
	Supported	65
	Unsupported	66

SIP Headers

There are four types of SIP headers:

- General headers, used in both requests and responses.
- Request headers, used for request messages to provide additional information about the request or the UAC.
- Response headers, which are added to responses to give information that supplements the response code and reason phrase.
- Entity headers, used to provide additional information about the message body or resource requested.

Table 17 lists the SIP headers that Dialogic supports. The headers are described in "Supported Headers" (page 61). Unsupported headers are parsed syntactically, but then discarded.

Table 17. Supported SIP Headers

SIP Header	Description	Page
Accept	Indicates acceptable message types.	61
Call-ID (i) (Mandatory)	Identifies a session or a call.	62
Contact (m)	URL where the user can be reached.	62
Content-Length (l)	size of message body.	62
Content-Type (c) (Mandatory)	Media type in the message.	63
CSeq (Mandatory)	Orders different requests within the same session and matches requests against responses.	63
From (f) (Mandatory)	Initiator of the request (name and SIP URL).	63
Max-Forwards (Mandatory)	Upper limit on the number of intermediary proxies.	64
Record-Route	Forces routing through a proxy.	64
Require	Options that must be supported.	64
Route	Forces routing.	64
Session-Expires	Serves as watchdog time out.	64
Server	Provides MS identifying information in IP Media Server request messages.	65

Table 17. Supported SIP Headers

SIP Header	Description	Page
Subject	Indicates client or server.	65
Supported	One or more options.	65
To (t) (Mandatory)	Recipient of the request (name and SIP URL).	65
Unsupported (Mandatory)	Features not supported by the server.	66
User-Agent	Provides IP Media Server identifying information in IP Media Server response messages.	66
Via (v) (Mandatory)	Records the route taken by a request.	66

Format and Syntax

SIP headers appear after the request line in Requests and after the status line in Responses.

A header line takes the syntax `headername: headervalue`. Allowed values can be a single numeric value, a hex string, or a comma-separated list of values. Values are not case sensitive.

Unsupported headers, malformed headers, and disallowed characters in supported headers do not generate a specific response. They may be ignored, or they may generate a 4xx error (final response indicating client error).

Supported Headers

This section lists and describes the supported headers in alphabetical order.

Common headers also have a compact form, which appears in parentheses after the header name; for example, From (f), Contact (m).

Accept

The Accept Header is used in the OPTIONS message to indicate to the Media Server that it would like to request the MS Usage. The format is as follows:

Accept: application/media_server_usage+xml

Optional in BYE, CANCEL, INVITE, and OPTIONS requests.

Optional in response to BYE, CANCEL, INVITE, and OPTIONS requests.

Values

Internet media (MIME types of the format type/subtype. See the accepted MIME types in Table 8 (page 49).

Call-ID (i)

This general header is created by the UAC to identify a call. It must be globally and chronologically unique across all calls, because it is used to keep track of the SIP session.

The combination of the To, From, and Call-ID headers completely defines a peer-to-peer SIP relationship (referred to in RFC 2543bis-03, "SIP: Session Initiation Protocol") as a call leg and in [4b] as a dialog.

The Call-ID is never modified by the IP Media Server.

Mandatory in all requests and all responses.

Values

A random identifier (case-sensitive string) that is globally and chronologically unique.

A common method of generating Call-IDs is to append the current time and host IP address to a random number.

Example This identifier is not guaranteed to be unique in the network: Call-ID: 123456 .

Example This identifier is guaranteed to be unique in the network, for example: Call-ID: 20031120105449192.168.15.91@my.com (random number:20031120105; time: 4:49; IP address: 192.168.15.91).

Contact (m)

This general header contains the URL to which the IP Media Server directs requests in a dialog.

Mandatory for ACK, INVITE, and OPTIONS requests.

Mandatory in response to INVITE and OPTIONS requests.

Values

All URL parameters are allowed.

Content-Length (l)

This entity header indicates the size of the message-body sent to the recipient. This header is optional. However, if Content Type is present, the octet count of the message body value must be correct.

Optional for all except CANCEL requests (for which it is N/A).

Optional for all responses.

Values

Decimal number of octets. Any value greater than or equal to 0 is valid.

Content-Type (c)

This entity header specifies the Internet media (MIME) type in the message body. If an Accept header was listed in the request, the response Content-Type must be listed in the Accept.

Mandatory when a message body is present in the method. Otherwise, optional in all requests and responses and not applicable for CANCEL requests.

Values

Internet media types of the format type/subtype:

- application/sdp for all services.
- application/mediaservercontrol+xml for advanced conferences and IVR.
- multipart/mixed when both application/sdp and application/mediaservercontrol+xml are present.
- application/mediaserverusage+xml for IP Media Server licensed feature set and IP Media Server processes and status

CSeq

This general header serves to uniquely identify and order SIP transactions within the same session.

This header contains a 32-bit unsigned integer that increases with every request, except for ACK and CANCEL requests, which use the CSeq number of the INVITE to which they correlate.

A response must contain the CSeq value from the request.

Mandatory in all requests and all responses.

Values

A decimal number followed by the method name.

From (f)

This general header indicates the originator of the message and is one of two addresses used to identify the call leg (the other is To:).

Per [3], requests and responses must contain a From general-header field indicating the initiator of the message.

Mandatory in all requests and all responses.

Values

All URL parameters are allowed.
URL is enclosed in <> when a display name is present.

Max-Forwards

This request header specifies an upper limit on the number of intermediary proxies.

Record-Route

This request header forces routing through a proxy. The proxy puts a Record-Route in the request, so that the response is routed back to the proxy.

Optional in all requests and all responses.

Values

The URL for the required route.

Require

This request header specifies options the UAC expects the IP Media Server to support. The IP Media Server can also use the Require header in a response.

Optional in all requests and all responses.

Values

Takes the value of the option(s).

Route

This request header describes routing for a request.

Optional in all requests and all responses.

Values

URL parameters for the route.

Session-Expires

This request header defines a session timer refresh interval.

Optional in INVITE requests.

Optional in 200 OK responses to INVITE requests.

Values

Takes a timer value (in seconds).

Server

This general header field contains information about the software used by the UAS to handle the request.

Subject

This general header field provides a summary or indicates the nature of the call and allows call filtering without having to parse the session description. The session description does not have to use the same subject indication as the invitation.

Optional in all responses.

Supported

This general header enumerates the capabilities (options) of the client or server.

Optional in BYE, CANCEL, INVITE, and OPTIONS requests.

Optional in response to BYE, CANCEL, INVITE, and OPTIONS requests.

Values

Takes the values of the supported options.

To (t)

This general header indicates the final destination (recipient) of the message.

Per [3] RFC 2543bis-03, requests and responses must contain a To: general-header field, indicating the desired recipient of the request.

Note: This header is not used for routing; the Request-URI is used for routing.

Mandatory in all requests and all responses.

Values

All URL parameters are allowed.
URL is enclosed in <> when a display name is present.

Unsupported

This general response header indicates features that are not supported by the server.

Optional in all responses.

Values

Takes the values of the unsupported options.

User-Agent

This general header contains information about the UAC originating the request. The purpose of User-Agent is to collect statistics, trace protocol violations, and automatically recognize user agents for tailoring responses to avoid particular user-agent limitations. User-agents should include this field with requests. The field can contain multiple product tokens and comments identifying the agent and any subproducts that form a significant part of the user-agent.

Via (v)

This general header records the route taken by a request. It contains the protocol name and version, type of transport, host name and address, and port number.

Mandatory in all requests and all responses.

Values

All URL parameters are allowed.
URL is enclosed in <> when a display name is present.

SIP Responses

Responses are returned for most requests. They consist of a status line, several headers, an empty line, and an optional message body.

A response status line consists of three elements:

- Protocol version
- Return code
- Reason phrase (The phrase associated with each code, for example: “SIP/2.0 100 Trying”.)

SIP Provisional Response Configuration

The IP Media Server can be configured to generate 180 or 183 SIP provisional responses. This feature provides flexibility to address the interoperability requirements of various SIP implementations.

Note: A provisional response configuration item does not affect the behavior of early media announcements, which by definition, require a 183 to be sent.

SIP Return Codes

All SIP requests other than ACKs are acknowledged with a return code in the response header. Codes are grouped in the following classes:

- 100-199 indicates a provisional response.
- 200-299 is a final response indicating success.
- 400-499 is a final response indicating client error.
- 500-599 is a final response indicating server error.
- 600-699 is a final response indicating a global error.

Note: The IP Media Server does not send redirection responses (class 3xx).

Table 18 lists the return codes that can be sent in responses from the IP Media Server, and notes when and why they are sent.

Table 18. SIP Return Codes Generated by the Media Server

Code	Code Classes	Reason Phrase	Examples of When / Why
100	Provisional	Trying	INVITE received.
183	Provisional	Session progress	Early media requested in INVITE for announcement (annc with early=yes).
200	Success	OK	• Request accepted. • CANCEL and BYE received.
400	Client Error	Bad request	• Error detected with a message body (such as the failure to execute an XML script). • Request received to create a conference that already exists. • Request received, but the session is not stable. • Request cannot be decoded by the SIP parser. • INVITE is improperly formatted. Response is set to the destination indicated by the Via header, if it can be parsed. Otherwise, it is sent to the source IP address:port that generated the request. The Request-URI cannot contain any errors in order for the response to be generated.
401	Client Error	Unauthorized	Request missing required headers.
403	Client Error	Forbidden	Requested SDP is not supported; for example, G.729a is requested for a conference.
404	Client Error	File not Found	Announcement content not found
405	Client Error	Method not allowed	Stable session established, but UAC attempts to perform an unsupported method.
415	Client Error	Unsupported media type	Requested SDP contains unsupported media type.

Table 18. SIP Return Codes Generated by the Media Server (continued)

Code	Code Classes	Reason Phrase	Examples of When / Why
420	Client Error	Bad Extension	INVITE contains Require header specifying an unsupported extension. The only supported extensions are “100rel” (Reliable Provisional Response) and “timer” (SIP Session Timer). The response contains a Supported header that lists these extensions.
480	Client Error	Temporarily unavailable	• Resource cannot be allocated. • Layer of stack not available for processing. • The IP Media Server exceeded the number of RTP licenses.
486	Client Error	Busy Here	• Contact was successful, but the callee is not willing or able to take the call. • Cannot add leg to valid conference.
487	Client Error	Request terminated	• Early media requested in an INVITE when announcement has completed. (This is the default response code to end of announcement when `early=yes`.) • Re-INVITE sent while session is being torn down. • INVITE sent when CANCEL has been received and processed with a non-stable session.

Table 18. SIP Return Codes Generated by the Media Server (continued)

Code	Code Classes	Reason Phrase	Examples of When / Why
488	Client Error	Not acceptable here	Received SDP information that cannot be processed. The 488 response contains a warning header that describes the details of the SDP negotiation issues, such as an SDP offer containing codecs that are unsupported by the IP Media Server or requests for a media type other than audio.
500	Server Error	Internal error	• Hardware or software error, or system resources are fully utilized. • Internal error processing a request (such as memory allocation or errors from SIP parser API). Maximum concurrent sessions are in use.
503	Server Error	Service Unavailable	• System is shutting down. • The system has been placed in an operational state in which no new calls are accepted.
606	Global Error	Not acceptable	The user wants to communicate, but cannot adequately support the session described. Even though the user's agent was contacted successfully, some aspects of the session description, such as requested media, bandwidth, or addressing style were not acceptable. The 606 response might contain a list of reasons in a warning header field, indicating why the session cannot be supported.

Session Description Protocol

The Session Description Protocol (SDP) provides a text-based format (field=value) for describing how a session is encoded.

SDP supports many types of sessions, from simple audio conversations to multi-media conferences with video and whiteboard support.

SDP content includes session-level information beginning with the v= header line, followed by media-level information, which begins with the m= header.

SDP requires that fields appear in a certain order. Table 19 lists and defines the SDP headers in the required order. Values are case-sensitive.

Examples of uses of SDP in SIP include the following:

- Version
- Origin
- Session
- Connection
- Media attributes

Because the type of media session and codec are part of connection negotiation, SIP uses the SDP to specify multiple media types, and can selectively accept or decline each type.

Note: All SDP must be compliant with ABNF as defined in RFC 2327. Syntax errors are rejected, even when they are present in SDP headers that the IP Media Server ignores.

Session Description Headers

Table 19 describes the SDP description headers and the action taken when they are either received or sent. Headers with a * next to them are defined as optional in RFC 2327. Actions are described further in Table 22 (page 74).

Table 19. SDP Description Headers

Header	Description	Action (When Received/When Sent)
v	Protocol version which is indicated by 'v=0'.	Ignored/Static Protocol version header is ignored when received in an offer and set to zero on a created offer or response.

Table 19. SDP Description Headers (continued)

Header	Description	Action (When Received/When Sent)
o	Owner/creator and session identifier. o=<username> <session id> <version> <network type> <address type> <address> Typical values for variables:	Ignored/Static The origin header is ignored when received as an offer and set to static and created values.
s	Session name (one per session description)	Ignored/Static Ignored when offered. Static value set to "SnowShore SDP" when created.
*i	Session information	Ignored
*u	URI of description	Ignored
*e	Email address	Ignored
*p	Phone number	Ignored
*c	Connection information. c=<network type> <address type> <connection address> Not required if included in all media descriptions. See “c= (Connection data)” (page 75) for details.	Extracted/Static/Created Extracted when offered. Static when not. Created for late media.
*b	Bandwidth information	Ignored
*z	Time zone adjustments	Ignored
*k	Encryption key	Ignored

Table 19. SDP Description Headers (continued)

Header	Description	Action (When Received/When Sent)
*a	Zero or more attribute lines. See page 76 for details.	Extracted/Static/Created

Time Description Headers

Table 20 lists and defines the SDP time description headers.

Table 20. SDP Time Description Headers

Header	Description	Action (When Received/When Sent)
t	Time the session is active.	Ignored
*r	Zero or more repeat times.	Ignored

Media Description Headers

Table 21 lists and defines SDP media description headers.

Table 21. SDP Media Description Headers

Header	Description	Action (When Received/When Sent)
m	Media name and transport address. m=<media> <port> <transport> <fmt list> See “m= (Media information)” (page 74) for details.	Extracted/Static/Created
*I	Media title	Ignored
*c	Connection information. Optional if included at session level. See “c= (Connection data)” (page 75) for details.	Extracted/Static/Created
*b	Bandwidth information	Ignored
*k	Encryption key	Ignored
*a	Zero or more attribute lines. See page 76 for details.	Extracted/Static/Created

Header Action Classes

Table 22 lists and defines the four different types of SDP header action classes associated with the SDP headers.

Table 22. Action Classes

Header	Definition
Ignored	The header and/or header fields are ignored.
Static	The header and/or header fields are static within the session control stack.
Extracted	The header and/or header fields are extracted from the parsed SIP method or SDP.
Created	The header and/or header fields are dynamic.

Header Definitions

The following sections describe the SDP headers and their respective attributes.

m= (Media information)

m=<media> <port> <transport> <fmt list>

Attribute	Description
<media> <port>	Media type is set to audio or video when offered, followed by the starting port number. All other media types are ignored.
<transport>	Transport protocol is always set to RTP/AV.

Attribute	Description
<fmt list>	Static or dynamic payload codec type(s). **Note:** For a list of supported media formats, see "Media Storage, Processing, and Supported Codecs" (page 33) and Table 4 (page 33). Multiple codecs offered to the IP Media Server will result in a single codec accepted. The choice normally is made from left to right, based on what the IP Media Server can accept. There must be at least one matching codec. Every code can be augmented by RFC 2833 payload type. The IP Media Server-offered codecs (one audio, one video) are configurable through the Web User Interface. G.711 μ-law is the default audio code. Use the SIP parameters: Prefer Offer Codec or Require Offer Codec to modify the audio codec selection algorithm. Refer to the Dialogic® IP Media Server *Installation and Operations Guide* to configure these SIP parameters. The payload type for telephone events in the SDP offer is configurable through the Web User Interface. The default payload type is 101.

c= (Connection data)

c=<network type> <address type> <connection_address>

Attribute	Description
Network type	IN
Address type	IP4
Connection address	IP address is set to the address designated for the media stream

Connection address setting of 0.0.0.0 is used to specify Hold (suspend audio) on an RTP session. See "IP Media Server Behavior When Hold Media is Presented" (page 83) for additional information on Hold.

Note: The IP Media Server does not support multicast.
A Time-to-Live (TTL) value in the c= field generates an error.

a= (Attribute Lines)

Optional attribute headers enable the inclusion of additional information and can be specified at the session level, media level, or both. A single SDP section can contain multiple attribute fields. The meaning of an attribute is position-dependent:

- If the a= appears before the first m= field, it is a session-level attribute.
- If the a= appears after a given m= field, it applies to that media type.

The following attributes are extracted from the SDP body and can also be defined for a session:

Attribute	Description
direction	The direction of a media stream. Possible values include: sendrecv, sendonly, recvonly. Parsed and reported when present and included within formatted SDP with a default value of sendrecv. The offered direction is configurable through the Web UI.
packet time (ptime)	The length of time in milliseconds per packet. Always included with formatted SDP and defaults to 20 ms if not present. The offered ptime is configurable through the Web UI.
dynamic payload type (rtpmap)	Defines a mapping between a codec and a dynamic payload type number. Only supported for RFC 2833 events. All other codecs must use the associated static payload type.

Ports

SIP

The IP Media Server listens for incoming SIP messages on the default port, 5060. This port is configurable through the Web User Interface.

RTP

You can set a starting port number, which is the low value in the range allowed for RTP port negotiation on the IP Media Server. The starting port number is set using the parameter RtpPortLow in the snowshore.cfg file. RtpPortLow can take even-number values from 6000 (the default) to a maximum of 40,000. RTP Ports are allocated on even number ports so you must allow for the addition of two times the number of licensed ports available – not to exceed 40,000. If the parameter is set to a value that falls outside the allowable range, RrpPortLow will be set to the default value (6000).

Note: Changes take effect only after an IP Media Server reset.

Reliability of Provisional Responses

Reliability of Provisional Responses, known as 100 reliability [`100rel`], is the exchange of acknowledgements for 1xx (greater than 100) class responses (provisional).

The Session Control Stack (SC) asserts 100rel when an initial INVITE method contains the Supported header with the value 100rel, as illustrated in Example 3.

Example 3. Supported Header with 100 Reliability

```
INVITE
   sip:annc@192.168.1.150:5079;
   play= file:////net/server/hello.wav SIP/2.0
Via: SIP/2.0/UDP 192.168.1.150:5078
To: <sip:annc@192.168.1.150:5079>
From: <sip:abc@192.168.1.150:5078>
Call-ID: 1944@192.168.1.150
Contact: sip:192.168.1.150:5078
CSeq: 18042 INVITE
Content-Type: application/sdp
Supported: timer
Supported: 100rel
Session-Expires: 60
Content-Length: 147

v=0
o=SnowShoreUaV1 12411 30414 IN IP4
   192.168.1.150
s=SnowShore Sdp
t=0 0
m=audio 8000/1 RTP/AVP 0
c=IN IP4 1.1.1.1
a=sendrecv
a=ptime:20
```

The IP Media Server responds with the Require and RSeq headers in any 1xx class method (excluding the 100 Trying method).

```
SIP/2.0 183 Session Progress
Contact: sip:192.168.1.150:5079
Via: SIP/2.0/UDP 192.168.1.150:5078
To:
   <sip:annc@192.168.1.150:5079>;tag=10013564
   03
From: <sip:abc@192.168.1.150:5078>
Call-ID: 1944@192.168.1.150
CSeq: 18042 INVITE
Require: 100rel
RSeq: 32163
Content-Length: 0
```

Once 100rel is negotiated, all subsequent 1xx class methods are acknowledged by the User Agent Client (UAC) with the PRACK method, to which the User Agent Server (UAS) responds with a 200 OK.

Figure 3 shows the call flow with a SIP provisional response. The numbers that follow indicate their respective sections listed inside Figure 3.

4 Initial INVITE request contains Supported header with 100rel.

5 1xx response contains required 100rel and Rseq headers.

6 Each 1xx response requires the UAC to return a PRACK request which makes the Media Server send a 200 OK response.

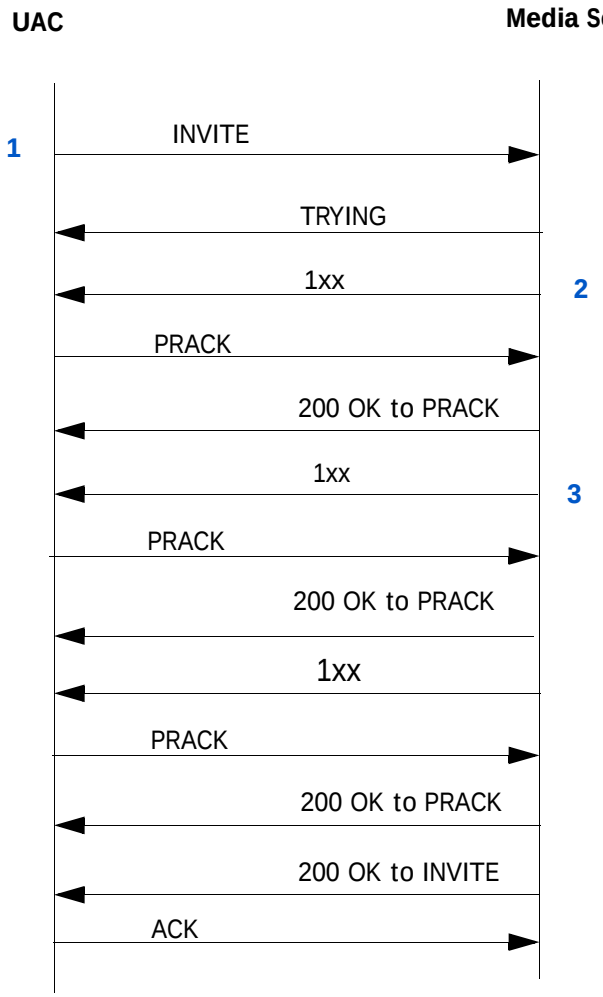

Figure 3. Call Flow: SIP Provisional Response

Syntax and Escaping

Various sections of the SIP message are permitted to contain different sets of characters. See the SIP specification (RFC 2543) reference for the full description of reserved and unreserved characters allowed in each field. All fields allow the full alphanumeric character set. For convenience, the following tables show which non-alphanumeric characters are considered to be unreserved in the respective fields and headers. Unreserved characters do not need to be escaped.

Table 23 lists the characters that are unreserved in a SIP request, as defined in RFC 2976.

Table 23. Unreserved Characters in SIP Request

Symbol	Name
-	hyphen
_	underscore
.	period
!	exclamation point
~	tilde
*	asterisk
‘	single quote
( )	parentheses
/	forward slash

Table 24 lists the characters that are unreserved in the user portion (before the @ sign).

Table 24. Unreserved Characters in User Portion

Symbol	Name
&	ampersand
=	equal sign
+	plus sign
$	dollar sign
,	comma
;	semi-colon
?	question mark
/	forward slash

Table 25 lists the characters that are unreserved for parameters.

Table 25. Unreserved Characters for Parameters

Symbol	Name
[ ]	brackets
/	forward slash
:	colon
&	ampersand
+	plus sign
$	dollar sign

Syntax and MSCML Body

In compliance with RFC 1341 (MIME), there must be a blank line (CRLF) following the boundary marker. A CRLF must also separate each line in the multipart/mixed MSCML body. Example 4 represents the correct syntax.

Example 4. Syntax for MSCML Multi-Part, Mixed Body

```
INVITE sip:conf=123456@10.10.140.1 SIP/2.0
Call-Id: call-12351-1.4.snowshore_conf.5@10.10.221.21
Contact: sip:snow_conf@10.10.221.25
CSeq: 1 INVITE
Expires: 180
From: sip:1.4.snowshore_conf.5@10.10.221.21
Record-Route: <sip:10.10.221.22>
To: sip:conf=123456@10.10.140.1
Via: SIP/2.0/UDP 10.10.221.21:5060
Content-Type: multipart/mixed;boundary=snowbound

--snow-bound
Content-Type: application/mediaserver+xml

<?xml version=”1.0”?>
<MediaServerControl version=”1.0”>
<request>
<configure_conference/>
</request>
</MediaServerControl>

--snow-bound
Content-Type: application/sdp

v=0
```

```
o=Pingtel 5 5 IN IP4 10.10.221.21
s=phone-call
c=IN IP4 10.10.221.21
t=0 0
m=audio 8766 RTP/AVP 0 96 8
a=rtpmap:0 pcmu/8000/1
a=rtpmap:96 telephone-event/8000/1
a=rtpmap:8 pcma/8000/1

--snow-bound--
```

IP Media Server Behavior When Hold Media is Presented

About Hold SDP

A SIP INVITE/ACK that contains a Hold SDP is a request for the other participant in the call (in this context, the IP Media Server) to cease sending RTP. The IP Media Server recognizes Hold SDP whether it arrives as an SDP offer in the INVITE request or as an SDP response in the ACK request.

Note that, the IP Media Server does not transmit any media streams while on hold.

The IP Media Server remains on hold until a subsequent INVITE/ACK arrives that contains SDP other than Hold SDP.

Hold SDP (as defined in RFC 2543) contains a connection address consisting of (c=0.0.0.0).

Example 5. Hold SDP

```
v=0
o=SnowShoreUaV1 12853 17864 IN IP4 192.168.12.155
s=SnowShore Sdp
t=0 0
c=IN IP4 0.0.0.0
m=audio 4556 RTP/AVP 0
a=sendrecv
a=ptime:20
```

Each participant in the call negotiates Hold independently. The IP Media Server does not automatically answer with Hold SDP when the IP Media Server receives the Hold.

However, when the IP Media Server responds to the initial INVITE to establish a conference Control Leg for an advanced conference, it automatically answers with a HOLD SDP. See "Advanced Conferencing (SIP with MSCML)" (page 84).

For all services, if the initial INVITE/ACK handshake that establishes a session contains Hold SDP, the IP Media Server does not start any media-related activity until a subsequent INVITE/ACK arrives that contains other than Hold SDP.

Hold Behavior for the Various Media Services

The following sections describe the Hold behavior for various media services.

Announcement Service (annc)

After receiving a re-INVITE with Hold SDP, the IP Media Server does not play the announcement immediately. When a subsequent re-INVITE with a valid destination IP address and port number arrives, the IP Media Server plays the announcement.

Note: This behavior is reliable only when playing content from NFS servers. HTTP retrievals timeout if the Hold duration value exceeds the HTTP's timeout value (fetch value). HOLD is not allowed during early media (`early=yes`) announcement requests.

Simple Conferencing (SIP)

A participant leg can accept Hold SDP in the initial INVITE/ACK. The session will not be connected to the conference until a subsequent INVITE/ACK is received with other than Hold SDP.

After receiving a re-INVITE with Hold SDP, the IP Media Server removes the session from the conference and stops sending RTP packets.

When the IP Media Server receives a subsequent re-INVITE with a valid destination IP address and port number, RTP is re-established as indicated in a new SDP section and the session is reconnected to the conference.

Advanced Conferencing (SIP with MSCML)

The initial INVITE that establishes a conference control leg and creates a conference should contain Hold SDP because the control leg has no associated RTP streams. This INVITE has an MSCML <configure_conference/> request in the SIP body. The IP Media Server always returns Hold SDP on the control leg, regardless of whether SDP was sent in the offer.

A participant (or a non-control) leg can accept Hold SDP in the initial INVITE/ACK. The session will not be connected to the conference until a subsequent INVITE/ACK is received with other than Hold SDP.

After receiving a re-INVITE with Hold SDP, the IP Media Server sends a response to any currently executing <play>, <playcollect> or <playrecord> request, indicating reason=stopped.

If the IP Media Server receives a subsequent re-INVITE with a valid IP address and port number, the IP Media Server restores the connection to the conference and resumes sending RTP packets to the IP address and port specified.

During Hold, the behavior of any media-related MSCML request received by the IP Media Server is undefined. The configure_leg request is independent of the state of the media streams and operates normally while on Hold. The subscription state for asynchronous DTMF and busy events is maintained across Hold/Retrieve transitions.

IVR Service (SIP with MSCML)

If a re-INVITE with Hold SDP is sent, the IP Media Server sends a response to any currently executing <play>, <playcollect> or <playrecord> request, indicating reason=stopped.

If the IP Media Server receives a subsequent re-INVITE with a valid IP address and port number, the IP Media Server resumes sending RTP packets to the IP address and port specified.

During Hold, the behavior of any media-related MSCML request received by the IP Media Server is undefined. The configure_leg request is independent of the state of the media streams and operates normally while on Hold. The subscription state for asynchronous DTMF and busy event is maintained across Hold/Retrieve transmissions.

Dialog Service (VoiceXML)

For the dialog (VoiceXML) service, the initial script is not retrieved until the IP Media Server receives an INVITE/ACK with non-hold SDP. The startup script is then invoked and VoiceXML processing starts.

The Hold function in a re-INVITE causes the IP Media Server to stop any media-related activity. This function causes the VoiceXML script to block at the next media-related request. The script is paused during Hold and all timers and event processing are disabled.

If the IP Media Server receives a re-INVITE with non-Hold SDP, VoiceXML script execution is restarted at the point where the Hold was received. Any play, record, or collect operation in progress when the Hold was received is also restarted from the point of interruption.

3 - Announcement Service API

This chapter describes the differences between simple and sequenced announcements, and outlines the use of variables.

For general information on SIP, see Chapter 2, "Session Initiation Protocol (SIP)". For examples and call flows, see Chapter 7, "Sample Code and Call Flows".

This chapter covers the following major sections:

- Overview
- Simple Announcements
- Announcement Sequences
- SIP Request Parameters for Announcements

Overview

Upon receipt of a SIP request from an application server, softswitch, or proxy server, the IP Media Server uses the Announcement Server API to deliver network announcements.

The Announcement Server API performs the following steps:

1. Listens for incoming SIP messages on port 5060. (The default port is configurable through the Web User Interface.)
2. Retrieves the files referenced by the play= parameter in the SIP request.
3. Encodes the retrieved data in an RTP stream.
4. Sends the RTP data stream to the destination indicated in the offered SDP.

File Retrieval

The URI identifies audio files, and these files can be stored anywhere accessible by the IP Media Server, using NFS or HTTP.

Network File Server (NFS)

The Dialogic® IP Media Server stores and retrieves data files on the Network Filer Server (RFC 1094 and RFC 1813).

All audio files retrieved by NFS must use the `file:///` scheme.

To access Dialogic prompts, the request must reference the following directory on the IP Media Server: /opt/snowshore/prompts/generic/.

HTTP

The Dialogic® IP Media Server retrieves data from HTTP/1.0 (RFC 1945) and HTTP/1.1 (RFC 2068) servers. Audio files retrieved by HTTP must use the `http://` scheme.

Announcement Types

The Dialogic® IP Media Server can process two types of announcements:

- Simple Announcements, which are single audio files with fixed content: "Your call did not go through. Please hang up and try again".
- Announcement Sequences, which are a grouping of audio files identified by a single name. They comprise a series of fixed audio files delivered in a single play-out: "Welcome to Dialogic Corporation. <welcome.wav> How may I direct your call? <direct.wav>".

Sequences can also contain variables that are evaluated at run-time. For example, "The number you have dialed <num_dial.wav>, 555-122-2222 <var1>, has been changed. <changed.wav> The new number is <new_num.wav> 555-122-3333 <var2>".

Announcement Service Indicator and Request URI

The announcement service indicator is annc.

```
INVITE sip:annc@MS_IP;play=URLname
```

For simple announcements, the Request-URI in the SIP INVITE points directly to the audio file to retrieve and play.

```
INVITE sip:annc@MS_IP;play=file:///opt/snowshore/
   prompts/generic/circuit_busy.ulaw
```

If you have configured a default location for all audio files for simple announcements, the SIP Request-URI will point directly to the file, as shown

```
INVITE sip:annc@MS_IP;play=wrong_number.ulaw
```

Simple Announcements

Simple announcements are single audio files that have no variables to resolve; for example:

```
INVITE sip:annc@MS_IP;play=http://server/path/
   hangup.wav
```

A simple announcement can be any length. It is termed simple because this type of announcement only requires a basic fetch-and-play operation. For examples and call flows, see Chapter 7, "Sample Code and Call Flows".

The SIP INVITE that initiates a simple announcement is directed to the announcement service on the IP Media Server, as indicated by the presence of `annc` (the announcement service indicator) in the Request-URI.

There are several "SIP Request Parameters for Announcements" (page 92). The only required parameter is play, which points to the audio file. Other parameters are early, repeat, delay, and duration.

Announcement Sequences

An announcement sequence is a series of audio files played back-to-back without interruption. Each part of a sequence is a unit of recorded information that can be reused in another context.

These units can be a single sound or word (“two”, “hello”), a phrase (“You have selected...”), or one or more sentences (“Our office will be closed today, Friday July 4th. We reopen for business on Monday the 7th.”)

Variable-Content Announcements

Variable-content announcements are a special type of sequenced announcements that incorporate variables, evaluated at run-time. In addition to listing the audio files to be included in the data stream, the specification for a variable-content announcement references the manner for rendering the variables.

Implementation

The IP Media Server can process both single audio segments and lists of audio segments which are identified by the MIME type `text/uri-list`. Announcement sequences rely on the IP Media Server's ability to play these arbitrary lists.

The application server (or another application component) provides a Web interface for generating the announcement sequence and returning it to the Dialogic® IP Media Server. The following example shows an application having a small audio segment library which consists of the following files:

File	Contents
welcome.wav	Welcome to...
reached.wav	You have reached...
hours.wav	Our normal business hours are 9AM to 6PM.
name.wav	The Museum of Transportation
closed.wav	The museum is currently closed.
current.wav	We are now offering an exhibit of vintage motorcycles.
website.wav	For a preview of this exhibit, visit our Website: www.transport.com.

When a call arrives after business hours, the following message plays: “You have reached the Museum of Transportation. The Museum is currently closed. Our normal business hours are 9 AM to 6 PM.” This message can be delivered by the sequence:

reached.wav
name.wav
closed.wav
hours.wav

For explanatory purposes, this portion of the code can be termed the afterhours sequence. The SIP request required to play this announcement would look like:

```
sip:annc@MS_IP;play=http://appserver.carrier.net;prompt=afterhours
```

The IP Media Server first retrieves the sequence of URLs from the application server and then fetches the audio files. The audio file list must include the URL scheme and the full path to the files. For example, if the files are stored on a centralized NFS server in the /var/prompts directory and accessed via NFS, the audio file list would look like:

```
file:////net/nfsserver.carrier.net/var/prompts/reached.wav
file:////net/nfsserver.carrier.net/var/prompts/name.wav
file:////net/nfsserver.carrier.net/var/prompts/closed.wav
file:////net/nfsserver.carrier.net/var/prompts/hours.wav
```

The following diagram illustrates an example of a sequenced announcement process. The IP Media Server includes an NFS automounter, so there is no need to explicitly set mount points for prompt directories.

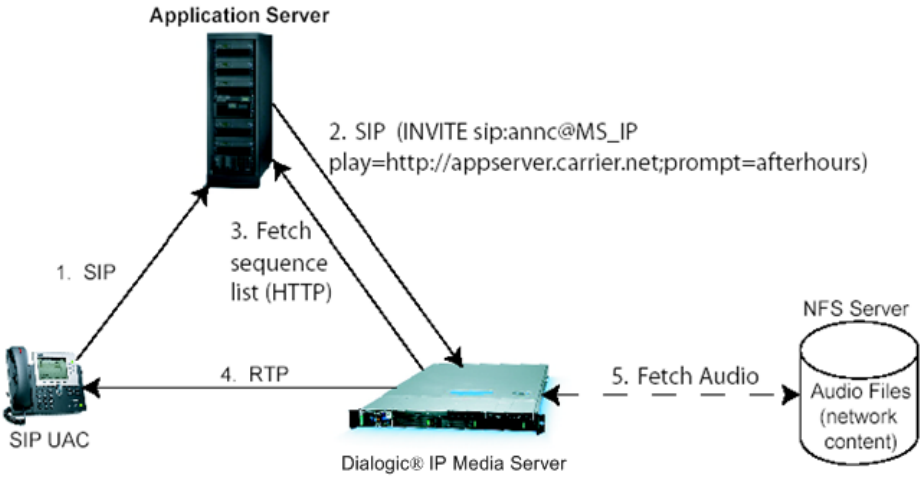

Figure 4. Playing an Announcement Sequence

Sequences can also contain variables by including URL(s) to the IP Media Server's onboard phrase server in the returned list. These URLs are of the form:

```
http://localhost/snowshore/phrase.cgi;
    locale=xxxxx&type=yyy&subtype=zzz&value=1234
```

where the URL parameters control what is spoken. For details on supported variables, see "Variable Types and Subtypes" (page 94).

SIP Request Parameters for Announcements

The Request-URI contains parameters specifying the audio segment to play and the associated parameter values. Parameters can appear in any order and are separated by semi-colons.

Play is required to reference the audio file. The `early` parameter is optional, but is also the default.

`Repeat`, `delay`, and `duration` are optional parameters and can be used when appropriate.

play

This parameter specifies the audio resource as a URL. The URL must resolve to a physical audio file or to an HTTP server process, for example servlet that returns a list of audio files using the MIME type `text/uri-list`. This parameter is required. Files are not retrieved unless this parameter is present.

Values

Value	Description
URI	The complete path for the audio prompt file or servlet, including parameters. Accepts NFS syntax (file:////) or HTTP syntax (http://).

Examples

```
play=http://path/audio/allcircuitsbusy.ulaw

play=http://audioserver.carrier.com;prompt=18
```

early

This optional parameter specifies whether or not the announcement is to play before the session is completely established.

Values

Value	Description
yes (default)	The message is played when the media streams are connected, but before the final response (487) is sent. See "Call Flow for an Early Media Announcement" (page 176) for a call flow example.
no	The announcement plays only after 200 OK.

Example

```
early=no
```

SIPAnnc404

If the Early parameter above is set to No, you must configure the following parameter in the snowshore.cfg file.

Values

Value	Description
yes	For regular announcements, if the announcement file is not found, the IP Media Server will return a SIP 404 Response to the INVITE method.
no (default)	Usual behavior

repeat

This optional parameter specifies the number of times the announcement or the elements of a sequence should be played in a loop.

Values

The allowed range is 1 to 250. (Default: 1.)

Example

```
repeat=2
```

delay This optional parameter defines the delay between repeat plays in milliseconds. It has meaning to the IP Media Server only if the `repeat` parameter is present; otherwise, it is ignored.

Values

The allowed range is 0 to 30,000 ms in 100 ms increments. (Default: 0 [no delay].)

Example

```
delay=20000
```

duration This optional parameter defines the maximum duration for the sequence in seconds. Once this amount of time has elapsed, the sequence stops playing. If this is not defined (duration=0), the sequence plays to completion.

Values

The allowed range is 0 to 6000 seconds in increments of 1 second. (Default: 0 [no duration limit].)

Example

```
duration=1000
```

Variable Types and Subtypes

Many standard phrases or items can be referred to using variables. Standard variables are described below in this section.

The URI of the desired variable can be specified explicitly in the `play` parameter or can be included as an item in a `text/uri-list` returned by a Web server process.

date The value is spoken as a date in the form specified by the subtype. The value is always specified as YYYYMMDD (per ISO 8601, International Date and Time Notation).

Subtypes

Value	Description
mdy	20021015 is spoken as “October Fifteenth Two Thousand Two”.
dmy	20021015 is spoken as “Fifteen October Two Thousand Two”.
ymd	20021015 is spoken as “Two Thousand Two October Fifteen”.

digit

The value is spoken as a string of digits, one at a time, with one of two phrasings.

Subtypes

Value	Description
ndn	is spoken with North American dialing phone number phrasing (NPA-NXX-XXXX), with appropriate pauses.
gen	is spoken as generic digits (one, five, zero).

duration

Duration is specified in seconds and is spoken in one or more units of time as appropriate. For example:

Value	Description
3600	is spoken as “one hour”.
3660	is spoken as “one hour and one minute”.
3661	is spoken as “one hour, one minute, and one second”.

money

Currency value in USD, spoken in dollars and cents.

Subtype

USD (Format: $$¢¢)

Examples

Value	Description
2500	is spoken as “25 dollars”.
25	is spoken as “25 cents”.
1	is spoken as “1 cent”.
100	is spoken as ”one dollar”.
101	is spoken as “one dollar and one cent”.
1025	is spoken as “ten dollars and twenty five cents”.

month The specified month in MM format, with 01 denoting January, 02 denoting February, 10 denoting October, and so forth.

number A number in cardinal or ordinal form, spoken with one of two phrasings.

Subtypes

Value	Description
crd	1 is spoken as “one”.
ord	1 is spoken as “first”.

silence Plays a specified period of silence as indicated by the duration value in milliseconds. For example, the following generates one second of silence:

```
silence:duration=1000
```

string Each character of the string is spoken. For example, “a34bc” is spoken as “A, three, four, B, C.”

Valid characters are a-z, A-Z, 0-9, #, and *.

time Spoken as a time of day in either twelve or twenty-four hour HHMM format according to ISO 8601, International Data and Time.

Subtypes

Value	Description
t12	1700 is spoken as “five p.m.”
t24	1700 is spoken as “seventeen hundred hours”.

weekday Spoken as the day of the week. Days are specified as single digits, with 1 denoting Sunday, 2 denoting Monday, and so forth.

4 - Conferencing API

This chapter provides information about simple and advanced (enhanced) conferencing.

Simple conferences are controlled and released with standard SIP INVITE and BYE requests. In contrast, advanced conferences are developed by extending SIP with Media Server Control Markup Language (MSCML). This chapter also includes information about MSCML elements and attributes that enable advanced conferencing.

This chapter contains the following sections:

Note: If you opt to perform the operations in this chapter, familiarize yourself with the SIP concepts and operations explained in "Session Initiation Protocol (SIP)"

The conference service indicator is conf. An equals sign (=) separates the conference service indicator from a URI or ID that uniquely identifies a conference session, for example: conf=confid@MS_IP.

SIP request URIs identify individual conferences. The URI can be any value compliant with the SIP URI specification, as detailed in RFC 2543. The conference control application must be configured to make sure that the identifier is unique within the scope of any potential conflict.

If the `INVITE` is directed to a URI that does not already exist on the IP Media Server and if the requested resources are available, the IP Media Server creates a new conference.

If the `INVITE` is directed to an existing URI, the IP Media Server interprets the INVITE as a request to join the conference represented by that URI.

Note: The conference URI shared between the application server and IP Media Server can be different from the conference URI visible to participants. This difference provides enhanced security and also simplifies resource management by allowing a specific IP Media Server to be assigned.

Simple Conferencing

The standard SIP Interface (with no MSCML) supports simple conferencing, suitable for calls that do not require the following:

- In-conference Interactive Voice Response (IVR)
- Explicit control of mixing mode
- Event reporting

How SIP Manages Conferences

Using SIP to manage conferences involves the following methods:

- The INVITE method to create the conference and join participants to it.
- The BYE method to remove participants from a conference and delete the conference.

Table 26 on page 103 lists the attributes and default values for simple conferences.

Creating a Simple Conference

A simple conference is created dynamically if the IP Media Server receives a SIP INVITE request to create a conference using the request URI. For example.

```
INVITE sip:conf=confid@MS_IP
```

If the SIP INVITE lacks an MSCML payload, the IP Media Server sets up a simple conference.

Adding a Participant to a Simple Conference

To add a participant to a simple conference, direct the `INVITE` request to the existing conference URI.

While a conference is in progress, participants can leave and rejoin the conference.

The IP Media Server imposes no restrictions on the size of a conference other than the total capacity of the server. If a request to join a conference exceeds the capacity of the IP Media Server, the IP Media Server sends the response: 480 temporarily not available.

Ending a Simple Conference

A simple conference remains allocated as long as there is at least one SIP session joined to it. When all participants have left the conference, as indicated by the SIP BYE method, the IP Media Server terminates the conference.

Attributes for Simple Conferences and Participants

Table 26 describes the attributes that are used to create simple conferences for either the entire conference or an individual participant of a conference.

Note: In simple conferences, these attributes take only their default values.

Table 26. Simple Conference Attributes

Attribute	Default Value	Description
type	talker	All participants (legs) are talkers.
mixmode	full	All audio is mixed.
toneclamp	no	Audio is not removed (clamped) when the IP Media Server detects certain repetitive, non-voice signals.
dtmfclamp	no	Detected DTMF digits are not removed from audio for all participants. See the following section for a detailed description concerning dtmfclamp used in a conference leg.

DTMF Clamping

In a conference leg with DTMF clamping set to no, detected 2833 events are regenerated (mixed) to all other conference participants. To receive a 2833 event payload, you must first set up a channel for 2833 mode.

Transmitting 2833 events through conferences is implemented with the concept of a single 2833 talker; only one leg can send 2833 at a time. The leg that first detects 2833 and is not clamped becomes the 2833 talker and has that 2833 event transmitted to completion. Once that event completes, another leg can become the 2833 talker. This scheme prevents mixing of simultaneous 2833 events from different sources thereby confusing the recipients. However, it allows for any leg to send 2833 in the conference, but not simultaneously.

In addition, if a non-2833 leg detects DTMF (and is not clamped), a matching 2833 is sent for each 2833 leg in that conference. However, the reverse is not the case—non-2833 legs will not have DTMF tone inserted on them when 2833 is in the mix.

Managing Video Switching

The video conferencing service operates as a simple conference which has both audio and video or just audio. The service indicator (conf) used for a video conference is the same as for a simple audio conference. An equal sign (=) separates conf from a URI or ID that uniquely identifies an instance of a conference session, for example:

```
conf=confid@192.168.12.153
```

where 192.168.12.153 is the IP address or the fully qualified domain name of the IP Media Server.

The only difference in an `INVITE` between joining an audio-only participant to the conference and an audio/video participant is in the SDP section. The audio/video participant's SDP contains a description of both the audio stream and the video stream, for example, `m=video 4028 RTP/AVP 34`.

Switching Rules

Switching in the conference is based on audio streams. The IP Media Server detects the three loudest talkers from the participants in the conference. The audio mix for the conference is a combination of these three active talkers. The video stream is selected and switched based on these audio active talkers as follows:

- The first participant to enter a conference is sent video silence (currently an image of a spinning snowflake).
- After other participants join, the video stream of the loudest talker (with some hysteresis) is sent to all other compatible participants (those using the same video codec). The loudest talker continues to receive the video stream he/she was previously receiving.
- When a conference is starting up and only one participant has been the loudest talker, every other participant sees the loudest participant. As long as the first participant remains the loudest talker, he/she sees video silence, because there is no previous loudest talker.
- When a second participant becomes the loudest talker, the first talker then sees that participant, and the new loudest talker sees the previous loudest talker.

Refresh Frames

Video streams switch only when a special type of frame referred to as a refresh frame ('intra' frame in H.263 and H.263+; 'IDR' frame in H.264) is received by the IP Media Server from the new loudest talker. A refresh frame contains all of the video information required to decode that frame (there is no dependency on data from previous video frames).

Different Client Types

The IP Media Server supports conferences among clients of different types.

Conferencing between video-enabled devices and audio-only devices. Video is never sent to the audio-only device. If an audio-only device becomes the active talker in a conference, each video-enabled devices receives the "video silence" stream (currently a looping video clip of a spinning snowflake) in its encoding (H.263, H.263+, H.264).

Conferencing between H.263, H.263+, and H.264 participants. Participants only receive a video stream when that stream is of the same encoding type. That is, H.263 endpoints receive only H.263 video streams, H.264 endpoints receive only H.264 video streams, and H.263+ endpoints receive only H.263+ video streams. If the video stream normally sent to an endpoint is not compatible, that endpoint receives only an audio stream.

Note: For some video clients, when the Media Server stops sending video packets (for example, when an incompatible endpoint becomes the active talker), the video displayed may freeze on the last frame received.

Call Flow and Sample Code Examples

For many types of code samples and call flows for simple conferences, see "Sample Code and Call Flows" (page 173).

Figure 5 shows an example of the call flow for creating a simple conference (normal media).

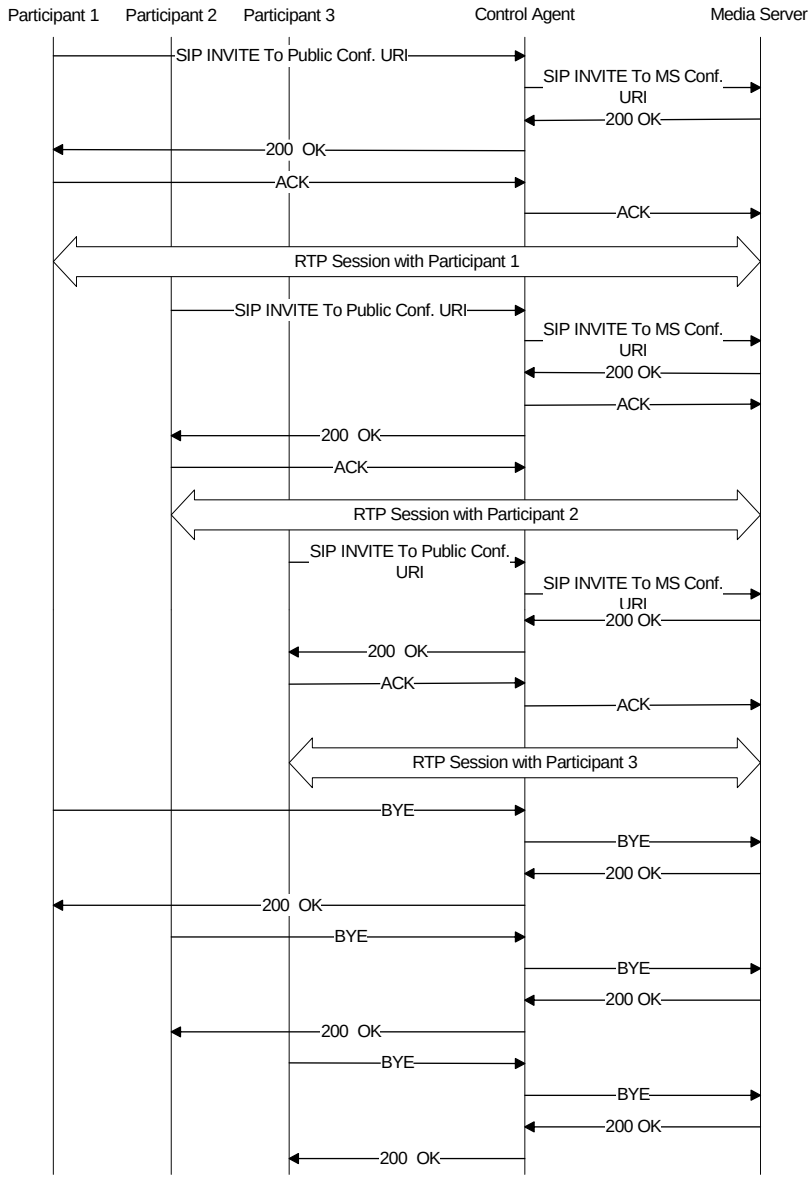

Figure 5. Call Flow for Simple Conference (Normal Media)

Code for Creating a Simple Conference

Example 6 illustrates a simple conference.

Example 6. Creating a Simple Conference

```
INVITE sip:conf=1234@192.168.12.153 SIP/2.0
From: sip:threepcc@192.168.1.126;tag=1as8ut
To: sip:conf=1234@192.168.12.153
Call-ID: 1031579120515@192.168.1.126
CSeq: 1327320033 INVITE
```

```
Content-Length: 201
Content-Type: application/sdp
Contact: <sip:192.168.1.126:5060;
   transport=udp>
Via: SIP/2.0/UDP 192.168.1.126:5060

v=0
o=Pingtel 5 5 IN IP4 192.168.12.109
s=phone-call
c=IN IP4 192.168.12.109
t=0 0
m=audio 8770 RTP/AVP 0 96 8
a=rtpmap:0 pcmu/8000/1
a=rtpmap:96 telephone-event/8000/1
a=rtpmap:8 pcma/8000/1


SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060
To: sip:conf=1234@192.168.12.153;tag=1031665657
From: sip:threepcc@192.168.1.126;tag=1as8ut
Call-ID: 1031579120515@192.168.1.126
CSeq: 1327320033 INVITE
Contact: sip:192.168.12.153:5060
Content-Type:application/sdp
Session-Expires:120
Content-Length: 153
a=ptime:20

v=0
o=SnowShoreUaV1 22263 30720 IN IP4 192.168.12.153
s=SnowShore Sdp
t=0 0
m=audio 4202 RTP/AVP 0
c=IN IP4 192.168.12.154
a=sendrecv
```

Advanced Conferencing

Extending SIP with MSCML allows developers to create complex conferencing applications.

Advanced (also called enhanced) conferencing allows an application to do the following:

- Set unique attributes for each participant
- Play announcements to the conference as a whole
- Create personalized mixes for each participant
- Record the conference
- Subscribe to conference events

Using MSCML for Advanced Conferencing

Advanced conferences are controlled by MSCML payloads sent in SIP requests. Each SIP INVITE/INFO method contains at most one MSCML body.

The Multipurpose Internet Messaging Extension (MIME) type used to describe the MSCML content is application/mediaservercontrol+xml. See "MSCML Schema" (page 295) for the formal definition of the MSCML grammar.

The two major MSCML elements for advanced conference functions are the following:

- configure_conference
- configure_leg

configure_conference

When sent in an `INVITE` message to conf=confid@MS_IP, this payload:

- Creates the conference
- Determines whether the conference subscribes to Active Talker events (Default: No)

When sent in an `INFO` message to an existing conference, a configure_conference message modifies the properties and event subscriptions for that conference.

configure_leg

When sent in an INVITE message to conf=confid@MS_IP, this payload joins a participant when the default attributes for the conference are not suitable for that participant.

When sent in an INFO message to the Call ID of a leg, a "configure_leg" message modifies attributes for that participant.

MSCML Attributes and Elements for <configure_conference>

The <configure_conference/> element can control the following elements and attributes:

- reservedtalkers and reserveconfmedia are set in the initial <configure_conference> INVITE message and are the only attributes specified in that message.
- reservedtalkers and subscribe for the entire conference can be modified in subsequent <configure_conference> INFO requests directed to the URI representing the conference.
- subscribe element can also be sent in the initial INVITE.

Table 27 describes the attributes and default values for advanced conferences.

Table 27. Advanced Conferencing Attributes

Attribute	Default Value	Description	See Page
reserveconfmedia	yes	Allocates resources to play or record audio to or from the entire conference.	141
reservedtalkers	N/A	Indicates the maximum number of conference participants (legs).	142

Table 28 describes the subscribe element for advanced conferencing.

Table 28. Subscribe Element

Element	Default Description
subscribe	Empty, meaning no event subscriptions are active.

Note: Configurable input gain is not currently supported on the IP Media Server.

MSCML Attributes and Elements for <configure_leg>

Table 29 describes the attributes and their default values for participant legs in an advanced conference. You can modify the following attributes and element for individual participants using the configure_leg INVITE/INFO request.

Table 29. Attributes for Participant Legs in Advanced Conference

Attribute	Default Value	Description	See Page
mixmode	full	Audio from all talker participants is mixed.	140
dtmfclamp	yes	Detected DTMF digits are removed from audio for all participants.	139
toneclamp	yes	Audio is removed (clamped) when the IP Media Server detects certain repetitive, non-voice signals.	142
type	talker	All participants are talker participants.	142

Table 30 describes the subscribe element for participant legs.

Table 30. Subscribe Element for Participant Legs

Element	Default Description
subscribe	Empty, meaning no event subscriptions are active.

MSCML Attributes for <managecontent>

The <managecontent/> element is used to move recorded content from the IP Media Server to remote locations using the HTTP protocol. This is a store and forward model, which requires the local temporary recording to be completed before it can be sent to the Web server (i.e., <playrecord/>). There is currently no provision for streaming the recording to the Web server while it is being recorded.

Table 31 describes the attributes of the <managecontent/> element.

Table 31. Attributes for <managecontent> in an Advanced Conference

<table>
<tr><th>Attribute</th><th>Description</th></tr>
<tr><td>src</td><td>Source to be copied, then deleted by the IP Media Server. The URL scheme must be file:////. This attribute is mandatory. It must be in the directory /var/snowshore and must be owned by the user.</td></tr>
<tr><td>Dest</td><td>Destination URL. The URL scheme must be http://. This attribute is not needed when the Action attribute =’delete’.</td></tr>
<tr><td>Action</td><td>Operation to be carried out. This option can be either ‘move’ or ‘delete’; ‘move’ is the default. The ‘delete’ operates on the local source file. After a successful move or delete, the source file is deleted from the IP Media Server If the request is unsuccessful, the source file is not deleted.</td></tr>
<tr><td>Httpmethod</td><td>HTTP protocol method to be used in request. Only ‘post’ or ‘put’ are allowed. The ‘post’ is the default.</td></tr>
<tr><td>Name</td><td>Field name for the content in the form when using the ‘post’ method. This is not to be confused with the <code>src</code> or <code>dest</code> attributes. It must be provided when using ‘post’. There is no default.</td></tr>
<tr><td>mimetype</td><td>MIME type of content being transferred. If not provided, the IP Media Server tries to infer it based on the following mappings:

<table><tr><th>Extension</th><th>MIME Type</th></tr><tr><td>alaw</td><td>audio/x-alaw-basic</td></tr><tr><td>.ulaw</td><td>audio/basic</td></tr><tr><td>.msgsm</td><td>audio/ms-gsm</td></tr><tr><td>.wav</td><td>audio/x-wav</td></tr></table>
If the content does not match one of the above, the mimetype attribute must be populated.</td></tr>
<tr><td>fetchtimeout</td><td>Maximum time permitted for operation in milliseconds. (Default: 10000 ms.)</td></tr>
</table>

<managecontent> Examples

When the caller is satisfied with the recording, the application commits it to persistent remote storage using <managecontent/>.

- The application sends:

```
   .
   .
   .
<managecontent
   id="102"
   src="file:////var/snowshore/rec/6A5GH49B.ulaw"
   dest="http://server.carrier.net/recordings/
      myrecording.ulaw"
   action="move"
   method="put"
/>
   .
   .
   .
```

Note: The application can change the temporary file name assigned by the IP Media Server as part of this operation, as shown.

- To which the IP Media Server responds:

```
   .
   .
   .
<response>
   id="102"
   request="managecontent"
   code="200"
   text="OK"
</response>
```

- If the request is ambiguous, the IP Media Server returns `code="4nn"`.
- If the IP Media Server is unable to perform the request, it returns `code="5nn"`. For example, if the file to be deleted either is not in `/var/snowshore` or is not writable by the IP Media Server, the IP Media Server returns the following:

```
<response>
   id="102"
   request="managecontent"
   code="500"
   text="Bad Request"
   reason=" Delete not permitted."
</response>.
```

- If there is a network or remote server error, the response shows the remote error within the successful IP Media Server response.

```
<response>
   id="102"
```

```
      request="managecontent"
      code="200"
      text="OK"
      <error_info>
         code="503"
         text="Service Unavailable"
         context="http://server.carrier.net/recordings/
            myrecording.ulaw"
      </error_info>
   </response>
```

Creating an Advanced Conference

All advanced conferences contain a control participant (leg). The SIP session, which creates the conference, is known as the control leg.

The control leg performs the following:

- Creates the conference and determines its lifetime. (An advanced conference exists as long as the control leg is present. If the control leg is disconnected, the IP Media Server sends BYEs to all remaining legs, terminating the conference.)
- Records audio from the entire conference.
- Plays audio to the entire conference.

If reserveconfmedia is set to yes, the control leg consumes one leg resource.

The conference control leg is used for conference-wide functions such as active talker reports and playing to or recording from the full conference. It is not used for conference participants, therefore RTP streams are not associated with the conference control leg. Because full conference play and record functionality are internally handled by the media sever, RTP is not involved.

If the application establishes the conference control leg with a SIP INVITE, it must contain "hold" SDP or no SDP. Either of these explicitly states that no RTP streams are associated with the control leg SIP session.

Note: No RTP is expected on the conference control leg. Therefore, the application should send a hold request (c=0.0.0.0), if it prefers sending any SDP. The application does not need to send any SDP.

For further details concerning Hold, see "IP Media Server Behavior When Hold Media is Presented" (page 83).

As with simple conferences, advanced conferences are created dynamically by sending a SIP INVITE to the URI representing the conference. The form is: sip:conf=confid@MS_IP .

When the initial SIP INVITE contains an MSCML configure_conference payload in the message body, the IP Media Server creates an advanced conference that can be modified with MSCML requests.

If the URI does not already exist on the IP Media Server, but resources are available, the request creates a new conference.

The configure_conference message contains the following attributes and elements that control characteristics of the entire conference:

- reservedtalkers (attribute), which sets the maximum number of conference participants.
- reserveconfmedia (attribute), which allocates resources for playing or recording audio to or from the entire conference. (Default: Yes.)
- subscribe (element), which is used if the conference subscribes to activetalker events.

The configure_conference payload sets the reserveconfmedia attribute in the initial `INVITE` message. This attribute cannot be changed.

To create a conference with 20 reserved talkers, the application specifies the `reservedtalkers` attributes in the initial `INVITE`.

```
<request>
   <configure_conference reservedtalkers= "20"
      reserveconfmedia="no">
   </configure_conference>
</request>
```

Note: The `reservedtalkers` attribute is currently ignored by the IP Media Server; so the conference size is not limited by the value of `reservedtalkers`.

Modifying an Advanced Conference

The initial INVITE with a <configure_conference/> request defines conference-wide attributes.

The server can change the event subscriptions for the conference. To do this, send a SIP INFO to the URI representing the existing conference. In that SIP request, include a <configure_conference> message body with values for the element `subscribe`. For a detailed definition of the subscribe element, see page 138.

Example 7 shows a request to modify an activetalkers event subscription by changing the default to `yes` with an interval value of `10` seconds.

Example 7. MSCML Request to Modify Conference Event Subscription

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
   <configure_conference>
      <subscribe>
         <events>
            <activetalkers report="yes interval=10”/>
```

```
            </events>
         </subscribe>
      </configure_conference>
      </request>
   </MediaServerControl>
```

Example 8 shows the response to the subscription request.

Example 8. Response to Modify Request

```
<?xml version="1.0"?>
   <MediaServerControl version="1.0">
      <response request="configure_conference"
         code="200" text="OK"/>
   </MediaServerControl>
```

Ending an Advanced Conference

The IP Media Server ends the conference when the control leg receives a BYE method from the controlling application.

Normally, an application server removes all conference participants before deleting the conference.

If one or more participants are still joined to the conference when the control leg receives a BYE, the IP Media Server removes the participants by issuing BYEs on any remaining participant legs to the application server, and then ends the conference.

Joining Participants (Legs) to an Advanced Conference

To join participants to an existing conference, the application directs a SIP INVITE to that conference URI.

If the default attributes listed in Table 32 are acceptable, the message body can consist of SDP only.

If additional features are required for a participant, the application server sends an INVITE to the conference URI with an MSCML <configure_leg> request that specifies the desired features.

Table 32 describes the attributes and their default values in the <configure_leg> message.

Table 32. Configure_Leg Attributes

Attribute	Value	Default	Description
dtmfclamp	{yes | no}	Yes	Removes the audio when a DTMF digit is detected.

Table 32. Configure_Leg Attributes (continued)

Attribute	Value	Default	Description
mixmode	{full | private | preferred | listen | parked}	Full	Specifies the mode for mixing audio from this leg.
type	{talker | listener | internal}	talker	Identifies the participant for this conference leg as a talker.
toneclamp	{yes | no}	Yes	Removes (clamps) audio when the IP Media Server detects non-voice signals.

Example 9 shows a configure_leg message body that changes the defaults for mixmode and toneclamp to parked and no, respectively.

Example 9. Joining a Participant with Non-standard Attributes

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_leg mixmode="parked" toneclamp="no"/>
   </request>
</MediaServerControl>
```

Modifying a Conference Participant

Make changes to an existing participant by using a SIP INFO request on the selected call leg that contains a <configure_leg> request, indicating the desired attribute. Example 10 shows a configure_leg request that mutes the leg.

Example 10. Modifying a Conference Leg

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_leg mixmode="mute"/>
   </request>
</MediaServerControl>
```

Removing Participants From a Conference

Remove conference participants from a conference when a SIP BYE is received on the corresponding call leg. No MSCML message body is required or expected. The IP Media Server ignores any message body in a SIP BYE request.

Conference Subsetting

To transfer participants from Conference A to Conference B, while still holding their place in Conference A, create a subset of the existing conference.

To subset an existing conference:

1 Create a second conference.

2 Put the selected participants on hold.

3 Re-INVITE the selected participants to the second conference.

Conference subsetting requires SIP third-party call control operations, but does not require explicit support in MSCML.

To return the participants to the original conference, the application reverses the operation.

Active Talker Events

The IP Media Server supports active talker events. These are conference-level events requested using the <activetalkers/> element, which takes two attributes:

- `report`
- `interval`

By default, the report attribute is set to No, and advanced conferences do not subscribe to active talker events.

To subscribe or turn on active talker events, the application sends a SIP INFO message with `report` set to a yes value. For a detailed example, see "Modifying Conference Using Subscribe" (page 189).

With report set to yes, the active talker event contains the number of talkers in the mix. The event identifies talker participants by their SIP Call IDs, which must be globally unique per SIP [3].

The supported notification method is SIP INFO. Example 11 shows how the XML is generated by the active talker event.

Example 11. Notification of Active Talker Events

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <notification>
      <conference uniqueID="ab34h76z" numtalkers="3"
         <activetalkers>
            <talker callID="123"/>
            <talker callID="456"/>
            <talker callID="789"/>
         </activetalkers>
      </conference>
   </notification>
</MediaServerControl>
```

IVR Operations during a Conference

Media Server Control Markup Language supports Interactive Voice Response (IVR) operations such as prompting, DTMF digit collection, and recording. These operations can be used, for example in conjunction with other advanced conferencing features to create complete conferencing applications.

For a detailed description of IVR features and how they are used with MSCML, see "IVR with MSCML" (page 143).

The application can perform IVR operations on either the conference control leg or an individual participant leg. To do this, the application sends INFO requests with the following MSCML commands:

- <play/>
- <playcollect/>
- <playrecord/>
- <managecontent/>

If sending an IVR request within the context of a conference, for example when sending a <play/> or <playcollect/> message to a participant leg, the application must first send an INFO to that leg, changing its mixmode to parked. After the IVR operation, the application resends an INFO to the leg, changing its mixmode back to full.

In this situation, the application server must wait for the IP Media Server's response to one INFO before sending the next.

Note: In the above-mentioned scenario, the application server must wait for the IP Media Server's <playcollect/> response before sending the <configure_leg/> request. Otherwise, the <playcollect/> operation may be stopped prematurely.

Note: The MSCML message bodies for play, playcollect, playrecord, and stop can be interrupted.This can occur as a result of an explicit <stop/> or if a new MSCML request is sent while the previous one is still executing.

Note: It is not necessary to `park` a conference leg for asynchronous DTMF reporting to function.

Playing and Recording Within the Entire Conference

To play announcements to the entire conference or record the conference output, enable the reserveconfmedia attribute and set it to yes (default). You set this default value in the <configure_conference> request that creates the conference in the initial INVITE.

In every conference:

- The conference control leg is configured for media handling/recording.
- If you wish to play media into a conference you are also recording, you can create a separate internal leg for that purpose. For information on how to do this, see "Simultaneous Play and Record" (page 125)
- Only the conference control leg or a special internal leg can play or record audio in the conference. The conference control leg is counted when calculating the capacity of the IP Media Server.

Set the `mixmode` attribute to preferred for this type of playing. Set the mixmode to preferred to make sure that the audio placed on the conference control leg is always mixed into the conference output.

Playing to the Conference

The <play/> request is typically used to inject audio into a conference; for example, to announce the name of a new participant.

The client application issues this request in a SIP INFO request, using the <prompt/> element to specify the audio files to be played.

Recording the Conference Output

To record, use the <playrecord> request. The key attribute of this request is the recurl, a URL reference to the target location for the recorded audio. The recording can be preceded as can occur by optional prompt.

There are three timer attributes for the <playrecord> request that control when and if a recording is terminated by the absence of speech. These timer attributes are as follows:

- initsilence
- endsilence
- duration

Note: When recording the output of a conference, these timers generally are disabled, by setting their values to -1.

You can also use the <managecontent> element to move recorded content from the IP Media Server to remote locations using the HTTP protocol.

Video Conferencing Enhancements

This release provides application developers with the ability to have added greater control over the media mix that is delivered to conference participants. Specifically, a new lecture mode feature has been added which provides that input media from the lecturer is always sent to all participants. Developers may choose whether the lecture properties are applied to the audio input, video input or both.

In addition, it is now possible to mute the audio and video inputs from a participant independently. Both features apply equally to live input or pre-recorded content. These enhancements can be utilized, for example in so-called "video sharing" applications where a pre-recorded video clip is played to the conference participants.

Refer to "mixmode" (page 140) for the Configure_Leg mixmode variations that support these enhancements.

MSCML Changes

Application developers may now control the output gain on conference legs through MSCML. See the mixmode variants highlighted below.

configure_leg Attribute - Mixmode

```
<!-- The list of current talkers is used only when
   sending -->
 <!-- notifications to the calling application. It
   should never -->
 <!-- be set when subscribing. -->
 <!ELEMENT configure_leg (inputgain?, outputgain?)>
 <!ATTLIST configure_leg
  id CDATA #IMPLIED
  type (talker | listener | internal ) #IMPLIED
  mixmode (full | mute | preferred | parked | private |
   mute_video | mute_audio | lecture | lecture_video |
   lecture_audio) #IMPLIED
  dtmfclamp (yes | no) #IMPLIED
```

IVR Operations on Participant Legs

Interactive Voice Response (IVR) operations on participant legs include the following requests:

- <play/>
- <playrecord/>
- <playcollect/>

Detecting DTMF Digits On A Conference Leg

In most cases, a participant leg must be parked before an IVR operation can be executed on that leg. The two exceptions to this are asynchronous DTMF and <playcollect/> when no prompt attribute is specified.

Using Playcollect

Conferencing applications often want to detect DTMF input on a conference leg to trigger some feature, such as dial-out or help. The <playcollect/> request collects DTMF digits with or without prompting.

This request causes the IP Media Server to process any digits previously input and possibly wait for additional digits, depending on the attributes supplied. One can use the cleardigits attribute to indicate whether the application should remove buffered digits.

Setting the firstdigittimer attribute of <playcollect/> to -1 causes the IP Media Server to wait forever for user input. Any buffered digits that were issued prior to the request will be processed.

When digits matching the request specification are detected, a <response> is sent to the application. The response contains the collected digits. If the application wants to detect additional DTMF input, another <playcollect/> request must be issued.

Using Asynchronous DTMF

For applications where immediate digit-by-digit notification is desired, the IP Media Server provides asynchronous DTMF event reporting. If using asynchronous DTMF, the application must manage all timers, pattern matching, etc. The asynchronous DTMF feature is used instead of <playcollect/>.

Note: Dialogic does not recommend mixing asynchronous DTMF and <playcollect/> for DTMF reporting.

To enable asynchronous `keypress` reporting, send the following MSCML payload (shown in Example 12) to the IVR or conference leg on the IP Media Server in a SIP INFO message. This payload is sent for each leg on which digit reporting is desired. Unlike activetalker reports, keypress reports are not directed to the conference control leg.

Example 12. Asynchronous Keypress Reporting

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_leg>
         <subscribe>
            <events>
               <keypress report="standard|long|both|none"
                  maskdigits=”yes|no”/>
```

```
            </events>
         </subscribe>
      </configure_leg>
   </request>
</MediaServerControl>
```

The `report` values are the following:

Value	Description
None	Do not report anything (disable reporting if enabled).
Standard	Report digits as they are detected.
Long	Report long digits as they are detected. A long digit is 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	Report both types of digits. Because a long digit consists of one or more normal digits, a single long duration key press generates one `standard` event and one `long` event. A double keypress creates two `standard` events and one `long` event.The notification events are sent in a SIP INFO method to the last contact address of record for the session.

The maskdigits values are:

Value	Description
Yes	Disables Clear Text Logging. DTMF data that is normally written into the IP Media Server log files is masked (replaced with asterisks) or discarded.
No (default)	Enables Clear Text Logging.

Example 13. Keypress Response Attributes and Values

```
<?xml version="1.0"?>
<MediaServerControl version=”1.0”>
   <notification>
      <keypress digit="[0-9]|[A-D]|#|*”
         length="standard|long"
         method=”standard|long|double"
         interdigittime="{mS}">
         <status command=”idle|play|collect|record"
            duration={seconds}/>
      </keypress>
   </notification>
</MediaServerControl>
```

The keypress response (Example 13) contains the following attributes (and the <status> element):

Attribute	Description
digit	Digit that was detected. Possible values are: `0–9, A–D, #,*`
length	Digit length. Values: standard, long.
method	Method used to collect digits: Values: **Value** / **Description** standard — Normal digits long — Long duration digits (more than 1 second) double — Double sequence digits
interdigittime	Time in milliseconds (ms) since the end of the previous digit to the start of this digit (interdigittime). If there is no previous digit, the interdigittime is 0.

The <status> element has the following attributes:

Attribute	Description
command	Command the IP Media Server was executing at the time the digit was entered. Values: idle, play, collect, record.
duration	How long the IP Media Server has been processing the command (in seconds).

Status information can be used to determine if a digit occurred during a play or a record, or was part of a collect.

Playing Audio to a Participant Leg

The <play/> and <playcollect/> requests deliver prompts and optionally collect DTMF input on participant legs. These functions normally occur after the user has made a feature request through some DTMF input, as described in the previous section.

To reduce the end-to-end signaling and setup time, the conference interface allows these IVR operations to occur while the user is joined to the conference URI. To turn on this feature, use the mixmode attribute of the <configure_leg> request.

When set to parked, the mixmode attribute isolates the user's audio input and output so IVR operations can occur. After the IVR operation is completed, the user's mixmode is returned to its previous setting with an additional <configure_leg> request.

Detecting and Reporting Busy Call Progress Tones in MSCML

The IP Media Server can detect busy call progress tones and notify the application if they occur. This capability is exposed through MSCML's generic event subscription and notification model. The feature can be used, for example to augment SIP signaling mechanisms when determining if a call has been connected. For example, calls that terminate on a PBX might appear to be connected from a SIP perspective, that is, a 200 final response was generated, but the end device might not have been picked up.

As in the case of other event types, subscriptions and notifications for busy events are transported in SIP INFO messages within an INVITE created dialog. The SIP dialog provides the context and scope for the specific call to which the event subscription pertains.

Busy events are supported by the IVR (ivr) and advanced (enhanced) conferencing (conf) services. To subscribe to busy events, as shown in the following code sample, send the indicated MSCML payload within the appropriate SIP dialog.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_leg>
         <subscribe>
            <events>
               <signal type="busy" report="yes|no"/>
            </events>
         </subscribe>
      </configure_leg>
   </request>
</MediaServerControl>
```

For performance purposes, the IP Media Server executes the busy tone detection algorithm only when the application has subscribed to this event. The application server subscribes to busy events only when they are needed and unsubscribes when they are not. Because this particular signal is useful only during the call setup phase, expect subscriptions to be short, for example, less than 30 seconds.

Busy notifications are sent in the same manner as all other MSCML events. However, the difference between busy events and MSCML events is the MSCML event child element, <signal/>. The IP Media Server sends a busy event notification <notification/> message after detecting the start of the busy tone.

Note: Only North American busy tones are currently supported.

Example 14 shows the busy event subscription, response and event notification process.

Example 14. Busy Event Subscription, Response, and Notification Process

Request

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

Response

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <response request="configure_leg" code="200"
      text=OK"/>
</MediaServerControl>
```

Notification

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

Un-subscribe to busy events and response request

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_leg>
         <subscribe>
            <events>
               <signal type="busy" report="no"/>
            </events>
         </subscribe>
      </configure_leg>
   </request>
</MediaServerControl>
```

Response

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <response request="configure_leg" code="200"
      text="OK"/>
</MediaServerControl>
```

For details concerning the elements, attributes, and values associated with busy event subscription and reporting, see "MSCML Conferencing Reference" (page 137). This reference is an alphabetical listing in tabular format.

Simultaneous Play and Record

You can record a conference while a media stream is being played to it. To do this, use the configure_leg method attribute type to create an 'internal' leg. That leg can record the conference while another leg is playing to the conference, or vice versa.

Creating an Internal Conference Leg

To create the internal leg on which to record the conference, send a SIP INVITE request using the MSCML element configure_leg with the attribute `type=internal`. When the internal leg is created, you can use it to record the conference.

Recording a Conference

To record a conference using the internal leg, send an INFO message to the internal leg with the MSCML command playrecord, for example:

```
<?xml version='1.0'?>
<MediaServerControl version='1.0'>
   <request>
      <playrecord
         recurl='file:////var/snowshore/rec/temp.ulaw'
            recencoding='ulaw' initsilence="infinite"
            endsilence="infinite/>
   </request>
</MediaServerControl>
```

The <playrecord> MSCML command that initiates the recording must be accompanied by specific command line arguments in order for the recording to take place properly. The 'initsilence="infinite"' and 'endsilence="infinite"' attributes must accompany the <playrecord> request, so that the recording does not begin or end based on the characteristics of the audio stream. The 'duration="infinite"' attribute may accompany the <playrecord> request, so that the recording does not end before the end of the conference.

The recording ends when either the MSCML <stop> request is received or the conference is removed from the IP Media Server, whichever occurs first.

MSCML Conferencing Requests

This section describes the MSCML request parent and child elements and attributes that support the conferencing service (conf=).

For a complete formal definition of the MSCML grammar, see "MSCML Schema" (page 295).

Conferencing Request Elements and Attributes

This section describes the MSCML conferencing request elements and their attributes.

configure_conference

The <configure_conference> element can appear in the INVITE or the INFO message.

In INVITE Message

The MSCML in the initial INVITE that creates the conference sets the reserveconfmedia and reserved talkers attributes.

Note: The reserveconfmedia attribute cannot be changed. Sending the reserveconfmedia in subsequent <configure_conference> requests to the same conference ID generates a 4xx client error response.

A <configure_conference> request must always be sent to establish an enhanced conference even if the <configure_conference> request does not have any additional attributes.

Note: The reservedtalkers attribute is currently ignored by the IP Media Server, so the conference size is not limited by the value of reservedtalkers.

Table 33 describes the attributes for an <configure_conference> element in an INVITE message.

Table 33. Configure_Conference Attributes in INVITE Message

Attribute	Values	Default	Description
reservedtalkers	2 -available IP Media Server capacity		Specifies the number of legs allocated when the conference is created.

Table 33. Configure_Conference Attributes in INVITE Message (continued)

Attribute	Values	Default	Description
reserveconfmedia	(yes | no)	Yes	Enables the application to record the conference and to play to the entire conference. A Yes setting allocates one of the available talker ports as the play leg and sets the mixmode for that leg to preferred. These settings ensure that conference announcements are mixed into the output as the loudest audio.

Table 34 describes the subscribe element.

Table 34. Subscribe Element in INVITE with Configure_Conference

Element	Description
subscribe	Used when subscribing and unsubscribing to active talker events.

In INFO Message

A <configure_conference> element sent in a SIP INFO request to an existing conference can modify the conference size and the subscribe element for the conference. See Modifying an Advanced Conference on page 112 for more information.

Table 35 describes the attributes for an <configure_conference> element in an INFO message.

Table 35. Configure_Conference Attributes in INFO Message

Attribute	Values	Description
reservedtalkers	2- available IP Media Server capacity	Specifies the number of legs allocated when the conference is created.

Table 36 describes the subscribe element.

Table 36. Subscribe Element in INFO with Configure_Conference

Element	Description
subscribe	Used when subscribing and unsubscribing to active talker events.

configure_leg

To set the properties for individual conference legs and modify asynchronous DTMF event subscriptions, use the configure_leg request element. This element can be sent in the following ways:

- 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.

Table 37 lists and describes the attributes for the configure_leg element.

Table 37. Configure_Leg Attributes

Attribute	Value	Default	Description
dtmfclamp	{yes | no}	Yes	Removes the audio when a DTMF digit is detected.
mixmode	{full | private | preferred | listen | parked | mute_video | mute_audio | lecture | lecture_video | lecture_audio}	Full	Specifies the mode for mixing audio from this leg.
type	{talker | listener | internal}	talker	Identifies the participant for this conference leg as a talker.
toneclamp	{yes | no}	Yes	Removes (clamps) audio when the IP Media Server detects non-voice signals.
gain now	delta=db level =db		

Table 38 describes the subscribe element.

Table 38. Subscribe Element for Configure_Leg

Element	Description
subscribe	Used when subscribing and unsubscribing to asynchronous DTMF events.

Coached Conferencing

The IP Media Server enables applications to create personalized mixes for participants through the MSCML <configure_team/> element. A common use of this feature is to support coaching of one participant by another.

The coaching scenario includes three participants:

- Coach (supervisor) who coaches the agent
- Agent who interacts with the client
- Client (customer) who receives advice from the agent.

A coached conference is a 3-way conference call where the coach monitors the call between the company's agent and the external client. The coach hears both the agent and the client speaking; the coach can speak to the agent, but the client cannot hear the coach.

Overview

To create an application that supports the coaching scenario where the client cannot hear the coach, one would need to identify the relationships among the participants.

You create personalized mixes by manipulating two MSCML objects:

- The list of team members (<teammate/> elements) set using <configure_team/>
- The mixmode attribute set through <configure_leg/>

The IP 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 application to specify other participants who can be heard in addition to the normal mixed output. For example, in the coaching scenario, the coach and the agent are configured as part of the team. As a result, the agent can hear the coach as well as the conference mix.

Team relationships are implicitly symmetric. If the application sets the coach as a team member of the agent, then the agent is automatically set as a team member for the coach.

One can use the id attribute set through <configure_leg/> to identify the various participants. Each participant must have a unique ID to use personalized mixing.

By itself, the team list only defines those participants that can be heard. The mixmode attribute of each team member determines whether their audio input is actually included in the personalized mix. If the teammate's mixmode is set to private, then it is included; if the mixmode is set to any other value, it is not. Returning to the coaching scenario, the coach's mixmode must be set to private to enable the coach's input to be heard only by the agent (student).

MSCML Elements and Attributes of Coached Conferencing

Coached conferencing requires the <configure_conference> element and depends on two other major MSCML elements:

- <configure_leg> (parent element) with its new value for the mixmode attribute (mixmode=private)
- <configure_team> (child element of configure_leg)

For information about these MSCML elements, see "configure_conference" (page 126) and "configure_leg" (page 128).

configure_team

The configure_team element allows the user to make the participants members of a team within a specific conference.

The configure_team element is a child of the configure_leg parent element.

The configure_team can be sent in either a SIP INVITE message or in an INFO message depending on your application needs.

The configure_team request can be used in the following situations:

- In an INVITE message to add a participant whose properties differ from the properties established for the conference as a whole.
- In an INFO message to change the properties for an existing leg.

Table 39 describes the attributes for the configure_team element.

Table 39. Configure_Team Attributes

Attribute	Value	Default	Description
action	add|delete|query|set	query	Allows the user to modify the team list by using one of the following values: • add—adds a teammate. • delete—deletes a teammate. • query—returns the teammate list. • 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 39. Configure_Team Attributes (continued)

Attribute	Value	Default	Description
id	alphanumeric characters; must be less than 64	unique ID (within the conference)	Identifies each participant uniquely in a coached conference. **Note:** If the MSCML configure_team specifies an invalid ID, the IP Media Server ignores the invalid ID and turns the action into a query.

Configuring a Coached Conference

To configure coached conferencing:

1 Join each leg to the conference, being certain to include a unique ID in the <configure_leg/> request.

Note: The leg ID needs to be unique only within the scope of the conference it belongs to.

2 Configure the teammate list and mixmode of each participant as required.

Table 40 summarizes how MSCML elements and attributes are used to create a coached conferencing configuration and the expected results.

Table 40. MSCML Elements and Attributes for Creating Coached Conferencing

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

It is often possible to combine both actions (step 1 and 2) in a single MSCML request.

The following sections describe how to create coached conferencing in a step-by-step approach.

Creating the Conference

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

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>
```

The IP Media Server responds as follows:

```
<?xml version="1.0"?>
<MediaServerControl version=?1.0">
   <response request="configure_leg" code="200"
        text="OK"/>
</MediaServerControl>
```

After this step, the conference looks like this:

Participant	ID	Team Members	Mixmode	Hears
Supervisor	supervisor	None	Private	Silence

Note: You cannot configure the teammate list for the coach yet because there are no other participants in the conference. A participant must be joined to the conference before it can be added as a teammate for another leg.

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>
```

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

The IP Media Server responds as follows:

```
<?xml version=”1.0”?>
<MediaServerControl version=?1.0”>
   <response request=”configure_leg” code=”200”
        text=”OK”>
      <team numteam=”1”>
         <teammate id=”supervisor”/>
      </team>
   </response>
</MediaServerControl>
```

After this step, the conference looks like this:

Participant	ID	Team Members	Mixmode	Hears
Supervisor	supervisor	Agent	Private	Agent
Agent	agent	Supervisor	Full	Supervisor

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. No further configuration is required 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>
```

Note: Strictly speaking, it is not a requirement that the customer leg be given a unique ID, because it will not be specified as a team member. However, when using coached conferencing, it can be beneficial for each leg to be assigned a unique ID in the initial INVITE request. Assigning a unique ID eliminates the need to set it later by sending a SIP INFO if personalized mixing for the customer leg is desired.

After the previous MSCML request is sent, the IP Media Server responds as follows:

```
<?xml version=”1.0”?>
<MediaServerControl version=?1.0”>
   <response request=”configure_leg” code=”200”
        text=”OK”/>
</MediaServerControl>
```

After this last step, the conference configuration is complete and should look like:

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

Supervisor Query for Number of Team Members

The IP Media Server responds to a query from the supervisor using numteam to indicate how many members are on the team. Example 15 shows that there are two members on the team, including the requester.

Example 15. Shows Two Members on the Team

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <response request="configure_leg" code="200"
        text="OK">
      <team id="super" numteam="1">
         <teammate id="agent"/>
      </team>
   </response>
</MediaServerControl>
```

Exiting the Conference

If the agent hangs up, the application removes the agent’s leg from the supervisor’s team list and from the entire conference. This process works exactly like the BYE method—no message is sent to other members of the conference.

For details concerning the BYE method and how a conference is torn down, see “Ending an Advanced Conference” (page 113).

Note: The IP Media Server does not keep track of old team lists or names of old team lists. Former team members must re-join using the MSCML configure_team element as initially stated.

For sample code and call flows associated with coach conferencing, see "Coached Conferencing" (page 202).

Using SIP INFO

Coached conferencing can be achieved by sending MSCML <configure_team/> elements in SIP INVITE requests. However, if the desired call flow is different or if the initial settings require changes, use SIP INFO to carry the desired MSCML payload.

MSCML Conferencing Reference

This section describes each of the MSCML conferencing elements and attributes.

MSCML Elements

This section describes the MSCML conferencing elements (listed in alphabetical order).

activetalkers

Subscribes to activetalker events.

Parent: events

Note: When subscription events are configured for reporting (report=), an updated report is issued only when the activetalkers change, regardless of the interval. If the interval is set to 60 seconds, and the same three participants talk for 119 seconds, there is no new report until the second 60-second interval.

---

configure_team

Configures the team list for a conference participant to enable personalized mixes.

Note: The personalized mix is not activated unless the participant's mixmode is set to private.

Parent: configure_leg

events

Contains a list of events whose subscription state is to be modified.

Parent: subscribe

Child Elements: activetalkers, signal, keypress

keypress

Subscribes to asynchronous DTMF events.

Parent: events

notification

Sent by the IP Media Server to notify the client application when an event is detected.

Parent: None

Child Elements: signal, keypress, conference

signal

Monitors outbound calls that can terminate on a PBX. In this particular case and from a signaling perspective, the call can be connected even though the target might not have been reached. The IP Media Server detects an in-band busy signal and reports this event to the application, so that false connects are eliminated.

Child Elements: notification

Values: busy (North American busy tone)

subscribe

Parent element required for busy event subscription, response, and event notification. To unsubscribe to busy events and response requests, set the signal attribute report to no.

Parents: configure_conference, configure_leg

Child Elements: events

teammate

Sets up and identifies the team list <teammate id=" "> of team participant(s) in coached conferencing. Used with the ID attribute.

Parents: configure_team, configure_leg

MSCML Attributes

This section describes the MSCML conferencing attributes (listed in alphabetical order).

action

Operation to be performed on the team list.

Attribute of: configure_team

Values

Value	Description
add	Adds a teammate.
delete	Deletes a teammate.
query (default)	Returns the teammate list.
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.

dtmfclamp

Whether the audio from the leg is removed (clamped) from the conference mix when a DTMF digit is detected.

In a conference leg with DTMF clamping set to no, detected 2833 events are re-generated (mixed) to all other conference participants. To receive a 2833 event payload, you must first set up a channel for 2833 mode.

Transmitting 2833 events through conferences is implemented with the concept of a single 2833 talker; only one leg can send 2833 at a time. The leg that first detects 2833 and is not clamped becomes the 2833 talker and has that 2833 event transmitted to completion. Once that event completes, another leg can become the 2833 talker. This scheme prevents mixing of simultaneous 2833 events from different sources thereby confusing the recipients. However, it allows for any leg to send 2833 in the conference, but not simultaneously.

Attribute of: configure_leg

Values:

Value	Description
yes (default)	Audio from leg is removed when a DTMF digit is detected. The default should be modified by applications only on rare occasions.
no	Detection of DTMF does not clamp the audio from this leg.

id

Alphanumeric characters that uniquely identify the three participants in coached conferencing. Number of characters cannot exceed 64.

Attribute of: configure_team

Example: `id=supervisor1`

mixmode

Mode for mixing audio from and to this leg.

Attribute of: configure_leg

Values:

Value	Description
full (default)	Audio is mixed, but not as the loudest leg.
preferred	The input audio from a leg with mixmode set to preferred is assigned a higher weight than legs set to full when determining who is included in the mix. With reasonable input levels from conference participants, this setting causes the preferred leg to be part of the mix. This setting overrides the default selection of mix inputs based strictly on loudness.
parked	Complete isolation from conference input and output. Used to isolate user audio input and output so IVR operations can occur. After IVR operation is complete, returns the user's mixmode to previous setting with an additional <configure_leg> request.
private	The leg's audio input is sent only to its teammates rather than to the full conference. The participant still hears the regular conference audio.
mute_video	Suppresses only the video feed of the participant and allows the audio to be fed into the conference mixer. If the leg with mute_video variation is the loudest talker then no video is sent to the other participants.
mute_audio	Suppresses only the audio from the participant and allows the video to be received from a participant.
lecture	Makes that leg the lecturer for the conference. All non-lecture participants have the audio and video muted.
lecture_video	Makes that leg the video lecturer for the conference. All non-lecture participants have the video muted once any participant becomes the lecturer_video.
lecture_audio	Makes that leg the audio lecturer for the conference. All non-lecture participants have the audio muted once any participant becomes lecture_audio.

repeat

Provides the client application with a way of repeating tone and prompts for a given number of times with the ability to limit the duration to a given length of time, if desired. The user can set the duration for an infinite length of time, or the repeating can be terminated by an interrupt.

Attribute of: prompt

Values: 0 - infinite (Default: 1.)

report

Enables and disables notification for events. Report is an attribute of activetalkers, signal, and keypress.

Attribute of: activetalkers, signal

Values:

Value	Description
yes	Enables reporting of the associated event.
no (default)	Disables reporting of the associated event.

Attribute of: keypress

Values:

Value	Description
normal	Reports normal digits.
long	Reports long digits.
both	Reports both manual and long digits.
none (default)	Disables asynchronous DTMF reporting.

reserveconfmedia

Allocates a control leg resource for playing or recording audio to or from the entire conference.

Attribute of: configure_conference

Values:

Value	Description
yes (default)	Must be set to yes for the application to play to or record the conference. The yes value consumes one of the available talker ports (as set by reservedtalkers) for the conference and sets the mixmode for the virtual conference leg to preferred, which ensures that conference announcements are mixed into the output as the loudest audio.
no	Does not play to or record the conference.

reservedtalkers

How many talker legs are allocated when the conference is created.

Audio from talker legs is processed by the mixing algorithm to produce the conference output.

This attribute can be carried in the initial SIP INVITE that creates the conference or in mid-call INFO request to resize an existing conference.

Note: The `reservedtalkers` attribute is currently ignored by the IP Media Server, so the conference size is not limited by the value of `reservedtalkers`.

Attribute of: configure_conference

Values are numbers in the range of 2 to available media sever capacity.

toneclamp

Removes (clamps) audio when non-voice signals are present in the media stream.

Attribute of: configure_leg

Values:

Value	Description
yes (default)	Audio is clamped when tones are detected.
no	Audio is not clamped.

type

The type of participant for this conference leg.

Attribute of: configure_leg

Values:

Value	Description
talker (default)	The audio input from the participant is processed by a mixing algorithm to create the conference output.
listener	Audio input from the participant is not included in the conference mix
internal	A conference leg used for playing to or recording from a conference.

5 - IVR with MSCML

This chapter explains the IP Media Server's implementation for Interactive Voice Response (IVR) using the ivr service indicator.

The IP Media Server offers two interfaces for those choosing to develop IVR/DTMF applications: IVR with MSCML and VoiceXML. For further information concerning VoiceXML, see "VoiceXML 2.0 and Dialog Service" (page 169), and "VoiceXML Version 1.0 and Dialog Service" (page 247).

This chapter covers the following major sections:

IVR Service

The IVR service (ivr) supports basic Interactive Voice Response functions, such as:

- Playing announcements
- Collecting DTMF digits
- Recording audio

These functions are based on MSCML elements that are added to the message body of a SIP request.

Example 16. IVR Service Indicator with MSCML Elements

```
To: sip:ivr@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407027 INFO
Content-Length: 237
Content-Type: application/mediaservercontrol+xml
Via: SIP/2.0/UDP 192.168.1.126:5060
<?xml version='1.0'?>
<MediaServerControl version='1.0'>
   <request>
      <playcollect barge='yes' cleardigits='yes'
           maxdigits='4'>
         <prompt>
            <audio url='http://192.168.1.126:8013/app/
               audio/askauthcode.raw'/>
         </prompt>
      </playcollect>
   </request>
 </MediaServerControl>

Response to INFO request from the IP Media Server:
 SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060
From: sip:appserver@192.168.1.126;tag=oi32zw
To: sip:ivr@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407027 INFO
```

The request payload (block or stream of data/information) for IVR can be located in either the initial SIP INVITE or in INFO requests.

Your application must use the INFO method for mid-call requests.

The application server must support sequenced INFOs per RFC 2543bis-03 "SIP: Session Initiation Protocol".

The IP Media Server notifies the application that the command has completed by sending a response message containing final status information and data such as collected DTMF digits. See "IVR Response Elements and Attributes" (page 166) for details on IVR responses.

IVR requests are not queued. If the application receives a request while another is in progress, the first operation is stopped and the new request is carried out. The IP Media Server generates a <response/> message for the first request and returns any data collected up to that point. If an application stops a request in progress, but does not initiate another operation, the IP Media Server issues a <stop> request and generates a <response/> message.

The IP Media Server treats a SIP re-INVITE with Hold media (c=0.0.0.0) as an implicit <stop/> request, immediately terminating the running <play/>, <playcollect/> or <playrecord/> request and sending a <response/> to the running request indicating reason=stopped. See IP Media Server "IP Media Server Behavior When Hold Media is Presented" (page 83)for additional information on Hold media.

The IVR service indicator is `ivr`.

```
INVITE sip:ivr@MS_IP
```

There are five MSCML directives for IVR functions:

- <play/>
- <playcollect/>
- <playrecord/>
- <stop/>
- <managecontent/>

The MSCML reference for these directives begins on page 118. See "MSCML Schema" (page 295) for the complete MSCML grammar.

See "Sample Code and Call Flows" (page 173) for code samples and call flow diagrams.

Playing Announcements

To play an announcement without interruption and with no digit collection, use the <play/> request. For example, use this request to announce the name of a new participant to the entire conference.

To immediately halt the current request and trap any content retrieval errors, use the Stop on Error feature. For further details, see "Handling of Content Retrieval Errors" (page 152)

Elements and Attributes

In the body of the request, the <prompt/> element specifies the announcement to play.

The attribute id (optional) is an application-defined request identifier that correlates the asynchronous response with its original request and echoes back to the application in the IP Media Server's response.

The promptur1, promptencoding, and offset attributes have been deprecated. Applications should use the <prompt/> element instead.

For further details concerning IVR attributes, see "IVR Attributes" (page 156).

Responses

When the announcement has finished playing, the Dialogic® IP Media Server sends a <response> payload to the application in a SIP INFO message.

The response contains the following with some examples noted in parenthesis:

- Return code (200)
- Return text (ok)
- Reason (EOF)
- Actual duration of the prompt
- ID, if one was provided

For more information about responses to <play>, see "IVR Response Elements and Attributes" (page 166).

Collecting DTMF Digits

To collect DTMF digits, use the <playcollect/> request.

This request has multiple attributes, all of which are optional. For a detailed description of the attributes, see "MSCML IVR Reference" (page 154).

Prompting

The presence or absence of the <prompt/> element controls whether to initiate an announcement or only collect digits. The following is an example of how the <prompt/> element works.

```
<prompt baseurl="file:///opt/snowshore/prompts/conf/">
   <audio url="please_enter.wav"/>
   <variable type="silence" value="1"/>
   <audio url="your.wav"/>
   <variable type="silence" value="1"/>
   <audio url="pin_number.wav"/>
</prompt>
```

For additional information about the prompt element, see "prompt" (page 125).

Digit Buffering

The ivr service automatically detects and buffers DTMF digits. When the IP Media Server receives a <playcollect/> request, it begins to examine the DTMF digits buffer.

To determine whether immediate action is required, the IP Media Server compares any previously buffered digits for a match with the following attributes:

- returnkey
- escapekey
- maxdigits

This examination provides the type-ahead behavior for menu traversal and other types of IVR interactions.

The application can override type-ahead behavior by setting the cleardigits attribute to yes, removing all previously-buffered digits. As a result, the application only considers user input that occurs after the request.

Caution: If cleardigits is set to no, previously-buffered digits may result in the prompt being interrupted (barged into) immediately. As a result, prompt play does not begin and digit collection starts immediately.

The default for barge is yes. If the barge attribute is set to no, cleardigits is implicitly understood to have a value of yes. This value provides that the application clears any DTMF input occurring before the current collection after the request completes.

Star and Pound Keys

The application can set two special keys to a single DTMF digit to invoke special processing when detected.

The escapekey (defaults to *) indicates that the user intends to terminate the current operation without saving any input collected up to that point. Detection of the specified escapekey digit terminates the request immediately and generates a response.

The returnkey (defaults to #) indicates that the user has completed the input and wants to return all collected digits to the application. If the IP Media Server detects a returnkey, the IP Media Server immediately terminates collection and returns the collected digits to the application in the <response> message.

The IP Media Server allows you to trap any content retrieval errors and immediately halt the current request by using the stop_on_error attribute. For further details, see "Handling of Content Retrieval Errors" (page 152).

Timing Attributes

Several attributes control how long the IP Media Server waits for digits in the input sequence. All timer values are expressed in milliseconds. The timers include the following:

- firstdigittimer controls how long the IP Media Server waits for the initial DTMF input.
- interdigittimer controls how long the IP Media Server waits between DTMF inputs.
- extradigittimer controls how long the IP Media Server waits for additional user input after the specified number of digits (maxdigits) have been collected.

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

If the IP Media Server detects the returnkey pattern during the extradigit interval, the collected digits are returned to the application and as a result, the returnkey is removed from the digit buffer.

If this were not the case, and considering the above example, the application would receive xxx and leave the terminating # in the digit buffer to be processed by the next <playcollect/> request. This could result in unintentionally terminating the prompt that immediately follows.

Note: The extradigittimer has no effect unless returnkey has been defined.

Responses

When the <playcollect/> has finished playing, the IP Media Server sends a <response> payload to the application in a SIP INFO message.

The <response> consists of the following, with some examples noted in parenthesis:

- Return code (200)
- Return text (OK)
- Reason (timeout)
- Actual duration of the prompt (playduration)
- Collected digits
- ID, if one was provided (application-defined)

For further details, see "IVR Response Elements and Attributes" (page 166).

Recording Audio

To record an audio request, use the <playrecord/> request.

The <playrecord/> request directs the IP Media Server to capture a real-time audio stream (Real-Time Transport Protocol) and deliver the resulting content to a URL specified by the controlling application. You can only specify file scheme URLs (NFS transport).

Playrecord Attributes

This request directive contains multiple attributes. The recurl attribute is required, because it identifies the URL for the recorded audio. All other attributes are optional

The presence or absence of the <prompt> element controls whether or not a prompt plays before recording begins.

Playrecord Process

When the IP Media Server requests a prompt, <playrecord/> has two stages.

The first is essentially a <playcollect/> operation. The application can set the prompt phase to be interrupted by DTMF input and can also specify an escapekey that will terminate the <playcollect/> request before the recording phase begins. The escapekey applies only to <playcollect/> and has no effect once recording starts.

When the escapekey is pressed, the IP Media Server generates a response message and the operation ends immediately. If any other keys are pressed and if the prompt has been set as interruptible (barge=yes), play stops immediately and the recording phase begins.

Any digits collected in the prompt phase, with the exception of those specified by the recstopmask attribute, are buffered and returned in the response.

If the request proceeds to the recording phase, the application discards any digits from the collection phase from the buffer to eliminate unintended termination of the recording.

The application compares digits detected during the recording phase to the digits specified in the recstopmask to determine if the digits should terminate recording.

Digits not present in the recstopmask are ignored and passed into the recording. If the recording is terminated because of a DTMF input, the collected digits are returned to the application in the <response>.

Once recording begins, the application writes the audio to the specified recurl URL, no matter what DTMF events are detected. The application must examine the DTMF input returned in the <response> message to determine whether the audio file should be saved or deleted and then re-recorded.

You have the option of trapping any content retrieval errors and immediately halting the current request by using the stop_on_error attribute. See "Handling of Content Retrieval Errors" (page 152) for more information.

Timing Attributes

The initsilence and endsilence timing attributes control how long the IP Media Server waits for the start of speech to begin the recording and the absence of speech to end the recording.

Table 41 describes these timing attributes.

Table 41. Timing Attributes

Attribute	Definition
initsilence	How long to wait for initial speech input before terminating (canceling) the recording. This attribute can be a positive integer value in milliseconds, or can be set to -1, which directs the IP Media Server to wait indefinitely. The default is 3000 ms (3 seconds).
endsilence	How long the IP Media Server waits after speech has ended to stop the recording. This parameter takes a positive integer value in milliseconds, or can be set to -1. With a value of -1, the recording continues indefinitely after speech has ended, but may terminate due to a DTMF keypress or because the maximum desired duration has been reached. The default maximum duration is 4000 ms (4 seconds). If the endsilence timer expires, the Dialogic® IP Media Server trims the end of the recorded audio by an amount equal to the endsilence parameter.

Additional Attributes

Table 42 describes additional attributes associated with <playrecord>.

Table 42. Additional Attributes of PlayRecord

Attribute	Definition
mode	Whether the recording overwrites or appends.
recencoding	Specifies the content codec to use.
duration	Specifies the time in ms for the entire recording.
beep	Specifies whether a beep signifies the start of recording.

For details on all IVR attributes, see "IVR Attributes" (page 156).

Responses

When the IP Media Server has finished recording, it generates a <response> message and sends it to the application in a SIP INFO message.

The response contains the following with some examples noted in parenthesis:

- Return code (200)
- Return text (ok)
- Reason (endsilence)
- Actual duration of the prompt (playduration)
- Collected digits
- ID, if one was provided (application-defined)

For further details, see "IVR Response Elements and Attributes" (page 166).

Handling of Content Retrieval Errors

Applications can control the IP Media Server's behavior when a content retrieval error occurs and capture detailed information regarding what happened. This feature is controlled through the stop_on_error attribute, which pertains to the <prompt/> tag. The <prompt /> tag appears in <play/>, <playcollect/>, and <playrecord/> requests. This attribute takes the following values:

Value	Description
yes	The IP Media Server stops the request if a fetch error occurs.
no (default)	Original MSCML behavior; fetch errors do not stop the request. For compatibility reasons, this is the default.

Details of the issue are returned in the <error_info/> element in the MSCML response. This element has three attributes: code, text and context. The code and text attributes are similar to their counterparts in the MSCML response; however, these attributes contain the error code and text received from the content server. The context attribute holds the URL that the IP Media Server was attempting to retrieve when the error occurred.

Stopping an IVR Request in Progress

The application issues a <stop/> request to stop a request in progress and not initiate another operation. This request generates a <response/> message from the IP Media Server.

ID Attribute

The id attribute is application-defined. This optional attribute is the only attribute related to the <stop> request.

This application-defined request ID correlates the asynchronous response with its original request and echoes back to the application through the IP Media Server's response.

Response

The response carries the following:

- Return code (200)
- Return text (OK)
- ID, if one was provided (application-defined)

MSCML IVR Reference

This section lists the MSCML elements and attributes that support the IVR service (ivr).

For the complete MSCML grammar, see "MSCML Schema" (page 295).

IVR Elements

Table 43 summarizes MSCML elements that support IVR.

Table 43. MSCML Elements for IVR

Element	Attributes	Response Attributes	Description
managecontent	src, dest, action, httpmethod, name, mimetype, fetchtimeout	N/A	Moves recorded content from the IP Media Server to remote locations using the HTTP protocol. This is a store and forward model, which requires the local temporary recording to be completed before it can be sent to the Web server (i.e., <playrecord/>). There is no provision for streaming the recording to the Web server while it is being recorded.
play	delay, duration, id, locale, offset, prompt, promptencoding, repeat, stop_on_error	code, id, playduration, reason, text	Plays an announcement that cannot be interrupted by DTMF input and does not require digit collection. Specifies the announcement to be played by creating a <prompt> element in the body of the request.
playcollect	barge, cleardigits, delay, duration, escapekey, extradigittimer, firstdigittimer, id, interdigittimer, locale, maskdigits, maxdigits, offset, prompt, promptencoding, repeat, returnkey	code, digits, id, playduration, reason, text	Collects user input or DTMF digits when an announcement might be interrupted. The announcement is optional, and if not set, the result is digit collection only.

Table 43. MSCML Elements for IVR (continued)

Element	Attributes	Response Attributes	Description
playrecord	barge, beep, cleardigits, delay, duration, endsilence, escapekey, id, initsilence, locale, mode, offset, prompt, promptencoding, recencoding, recstopmask, recurl, repeat	code, digits, id, playduration, reason, reclength, text	Request used when recording audio to set the properties the IP Media Server uses to capture RTP and deliver it to a URL specified by the controlling application. The application can specify an optional prompt to play prior to the recording.
stop	id	code, id, text	Stops a request in progress and does not initiate another operation.

IVR Prompt Block

The prompt block in the body of the play, playcollect, or playrecord request contains one or more references to physical audio files, provisioned sequences, or attributes that are played in the order in which they appear.

Attributes: baseurl, locale, stop_on_error, repeat, duration, offset, delay

The following examples (17–19) illustrate several uses of the prompt block.

Example 17. Prompt Block

```
<prompt baseurl="file:///opt/snowshore/
     prompts/conf/">
   <audio url="please_enter.wav"/>
   <variable type="silence" value="1"/>
   <audio url="your.wav"/>
   <variable type="silence" value="1"/>
   <audio url="pin_number.wav"/>
</prompt>
```

Using the repeat, duration, delay and offset attributes, the application can repeat prompts for a given number of times with the ability to limit the duration to a given length, if desired. The user can set the duration attribute for an infinite length of time or the application can terminate repetition using an interrupt <stop/>. The following examples illustrate these attributes.

Example 18. Prompt Block with Repeat for Single File

```
<request>
<play id=212>
   <prompt repeat="infinite" duration="infinite"
      offset=”1” delay=”10”>
```

```
      <audio url="file:////opt/snowshore/prompts/
         generic/10.ulaw"/>
   </prompt>
</play>
</request>
```

Example 19. Prompt Block with Repeat for Several Files

```
<request>
<play id=21212>
   <prompt baseurl="file:////opt/snowshore/prompts/
        generic/" locale="en_US" repeat="infinite"
        duration="infinite" offset="0" delay="2000">
      <audio url="num_changed.wav"/>
      <audio url="new_number.wav"/>
      <variable type="dig" subtype="ndn"
         value="9725551212"/>
      <audio url="make_note.wav"/>
   </prompt>
</play>
</request>
```

Prompt Elements

The VoiceVML prompt element takes two elements to specify the file to be played: <audio> and <variable>.

<audio>

Each audio element in a <prompt/> block refers to a physical audio file or sequence to be played. Audio files are played in the order in which they are listed in the block.

Attributes: encoding, url, src

<variable>

A symbol or string of symbols whose value changes.

Attributes: subtype, type, value

IVR Attributes

Attributes are listed in alphabetical order. All attributes are optional unless otherwise noted.

barge

Whether a prompt gets interrupted when any DTMF digit is detected.

Attribute for: playcollect, playrecord

Values:

Value	Description
yes (default)	DTMF digit detection interrupts the prompt. **Note:** Any digit barges in. There is no barge mask.
no	DTMF digit does not interrupt prompt. Any digits that arrive during the prompt are discarded. The digits are not saved for processing during the collection portion of playcollect.

baseurl

Base URL to be placed in front of the url attribute of any <audio/> tags that do not match a supported scheme (`file:////, http://`). This is a notational convenience to reduce the size of the MSCML payload.

Attribute for: prompt

beep

Whether a beep is played to signify the start of a recording.

Attribute for: playrecord

Values:

Value	Description
yes (default)	The IP Media Server plays a tone to signify the start of a recording. The beep is an 880 Hz tone 250ms in duration.
no	No tone is played at the start of a recording.

cleardigits

Whether the application removes previously collected digits from the buffer before the command starts. For type-ahead, set cleardigits to no.

Attribute for: playcollect, playrecord

Values:

Value	Description
yes	Digits are flushed.
no (default)	Digits are not flushed.

delay

Time between plays for the entire sequence (in milliseconds).

Attribute for: prompt

Values range from 0 – infinite. (Default: 0).

duration

Number of milliseconds that the entire recording can span. Recordings continue until an ending condition (such as a digit or ending silence) is detected or until the recording media is full.

Attribute for: playrecord

Values: 0 - infinite (Default: infinite.) A value of -1 means no limit is imposed.

encoding

Content encoding of physical audio files that are not in self-describing .wav or .au formats.

Attribute for: audio

Values:

Value	Description
µlaw	G.711 µlaw encoding.
alaw	G.711 alaw encoding.

There is no default.

endsilence

Number of milliseconds of trailing silence that can elapse before terminating the recording. The IP Media Server reduces the recording by an amount equal to the endsilence value if endsilence is set.

Attribute for: playrecord

Values: a number in milliseconds. (Default: 4000 ms.) To disable the setting, use a value of -1, which indicates that no endsilence timer should be applied.

escapekey

A digit that is used to terminate the current operation.

When this digit is pressed, the command terminates and the digit buffer is cleared. No digits are returned to the application, and the reason escapekey is returned with the <response/> event.

If detected in the play phase, the prompt stops immediately, and the command completes without entering the record phase.

Attribute for: playcollect, playrecord

Values:

Default is * (star key). If no escapekey is desired or if you want to collect * as a digit, you can set escapekey to be empty using two consecutive double quotes, for example, escapekey"".

extradigittimer

Amount of time allowed for the user to enter an additional digit after the desired number of digits has been collected. This attribute is typically used when collecting variable-length inputs with a returnkey parameter defined. If this timer elapses, the command completes with the reason=timer.

Attribute for: playcollect

Values: A number in milliseconds.(Default: 1000 ms.) A value of `-1` directs the IP Media Server to wait indefinitely for additional input.

Note: If the extra digit detected is the returnkey, then all collected digits are returned.

---

firstdigittimer

Amount of time allowed for the user to enter the first DTMF digit. If the timer elapses, the command completes with the reason nodigits.

Attribute for: playcollect

Values: A number in milliseconds. (Default: 5000 ms.) A value of -1 directs the IP Media Server to wait indefinitely for the first input. Using this setting for conferences ensures that user input is not ignored.

id

An application-defined request identifier that correlates the asynchronous response with its original request. The provided ID is returned back to the application in the IP Media Server's response.

Optional.

Attribute for: play, playcollect, playrecord, stop

interdigittimer

Amount of time allowed for the user to enter additional digits before the desired number of digits have been collected. If this timer elapses, the command completes with the reason timer.

Attribute for: playcollect

Values: A number in milliseconds. (Default: 2000 ms.) A value of -1 directs the IP Media Server to wait indefinitely for additional input.

initsilence

Number of milliseconds of initial silence that can elapse before the recording is terminated. If speech is detected in this interval, recording proceeds.

Attribute for: playrecord

Values: A number in milliseconds. (Default: 3000ms.) A value of -1 indicates that no initial silence period is desired or alternately that the IP Media Server should wait forever for the start of speech. The value of -1 effectively disables the setting.

locale

Language and country variant to be used when resolving variables contained in the <prompt> block. 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 elements are concatenated with a single underscore (%x5F) character.

Attribute for: prompt

Example: `locale=en_US`

maskdigits

Manages clear text logging.

Attribute for: playcollect, event report, report keypress

Values:

Value	Description
Yes	Sensitive data is not be logged in clear text. DTMF data that is normally written to the IP Media Server log files is masked or discarded. Masked DTMF data appears as a '-' character in the log files. Discarded DTMF is indicated by the text string <?Sensitive Data?> in the log files.
No (default)	DTMF data is recorded and written to the IP Media Server log files.

maxdigits

Maximum number of digits to be collected and returned by the IP Media Server.

Attribute for: playcollect

Values: 1–62. (Default: 1.)

mode

Whether the recording overwrites or appends to the supplied URL.

Note: This attribute only affects file:/// scheme URLs.

Attribute for: playrecord

Values:

Value	Description
overwrite (default)	Recording overwrites the contents of the URL.
append	Recording is appended to the contents of the URL.

offset

Position in a prompt or sequence where play begins. The value of this attribute is expressed in milliseconds with 0 indicating the start of the prompt or sequence.

Offset can be either outside or inside the prompt block. The offset outside the prompt block supports existing code and refers to the offset in a single set. If the offset is included inside the prompt block, it works the same way, but only on the first set of repeats.

The IP Media Server supports both positive (skip forward) and negative (skip back) values. Offsets apply to sequences as well as files. If the specified offset cannot be reached in the current segment, the next one is processed until the desired offset is reached.

Attribute for: prompt, play, playcollect, playrecord

Values: 0 - infinite (Default: 0.)

promptencoding

Encoding of physical audio files that are not in self-describing .wav or .au formats.

Note: This attribute only affects file:/// scheme URLs.

When using a physical audio file, the encoding a-law or µ-law must be specified in the file header, in the extension, or by using this attribute.

Attribute for: play, playcollect, playrecord

Values:

Value	Description
ulaw	G.711 µ-law encoding.
alaw	G.711 a-law encoding.
msgsm	MSGSM

If not specified, the promptencoding is determined from either the file extension, or if the first four bytes of the file indicate the presence of a RIFF or .snd header.

recstopmask

DTMF digits that terminate the recording. Input digits not contained in the mask are ignored and passed into the recording. The terminating DTMF digit is returned to the application in the <response/> event.

Attribute for: playrecord

Values: 0 1 2 3 4 5 6 7 8 9 * #

recurl

Target for the recording. Only NFS (file:////) is supported for recording. Filenames ending in .au or .wav produce files written in those formats. Required.

Attribute for: playrecord

recencoding

Encoding used for the recording.

Attribute for: playrecord

Values:

Value	Description
ulaw (default)	G.711 μ-law encoding.
alaw	G.711 a-law encoding.
ms_gsm	MSGSM

repeat

Provides the client application with a way of repeating tones and prompts for a given number of times with the ability to limit the duration to a given length of time, if desired. The user can set the duration for an infinite length of time or the repeating can be terminated by an interrupt <stop/>.

Attribute for: prompt

Values: 0 - infinite (Default: 1.)

Note: The value infinite means the prompt should be repeated indefinitely.

report

Enables or disables notification for events. Report is an attribute for both signal and keypress.

Attribute for: signal

Values:

Value	Description
yes	Enables reporting of the associated event.
no (default)	Disables reporting of the associated event.

Attribute for: keypress

Values:

Value	Description
normal	Reports normal digits.
long	Reports long digits.
both	Reports both manual and long digits.

Value	Description
none (default)	Disables asynchronous DTMF reporting.

returnkey

Digit that signifies the end of the collection process.

When this digit is pressed, the command terminates. All collected digits are sent to the application, and the reason returnkey is returned in the <response> event with the collected digits.

Attribute for: playcollect

Values: 0 1 2 3 4 5 6 7 8 9 * # A, B, C, D

The default is # (the pound key).

If the returnkey is not required or if you want to collect # as a digit, set the returnkey to be empty by using two consecutive double quotes for example, returnkey="".

stop_on_error

Controls IP Media Server handling and reporting of errors encountered when retrieving remote content.

Attribute for: prompt

Values:

Value	Description
yes	The IP Media Server stops the request if a fetch error occurs.
no (default)	Original MSCML behavior; fetch errors do not stop the request. The default is no for compatibility reasons.

subtype

Minor or subtype values describing how to render a variable as audio.

Attribute for: variable

Values:

Value	Spoken as...
mdy 20021015	“October Fifteenth Two Thousand Two”
ymd 20021015	“Two Thousand Two October Fifteen”

Value	Spoken as...
ndn	North American dialing phone number phrasing (NPA-NXX-XXXX), with appropriate pauses
dmy	20021015 is spoken as “Fifteen October Two Thousand Two”.
t12 1700	“Five p.m.”
t24 1700	“Seventeen hundred hours”
gen	generic digits (for example: “one”, “five”, “zero”)
crd	“one”
ord	“first”

For detailed descriptions of subtype values, see “Variable Types and Subtypes” (page 94).

type

Type is an attribute for signal and variable.

Attribute for: signal (required)

Identifies the type of signal to be reported.

Values: busy.

Attribute for: variable (required)

Identifies the type of variable to be rendered as audio.

Values: date**,** digit, duration, money, month, number, silenc, string, time, weekday.

For detailed descriptions of subtype values, see “Variable Types and Subtypes” (page 94).

url

URL of the audio to be played. Can resolve to a physical file or a provisioned sequence.

Attribute for: audio (required)

value

String value for conversion to audio according to its type and subtype for example, 12/25/02.

Attribute for: variable (required)

IVR Response Elements and Attributes

The IP Media Server acknowledges receipt of an application request by sending a response of either 200 OK or 415 BAD MEDIA TYPE. (The IP Media Server sends the 415 BAD MEDIA TYPE response when the SIP request contains a content type other than application/sdp or application/mediaservercontrol+xml).

The <response> message is transported in a SIP INFO request.

If there is an error in the request or the request cannot be completed, the <response> message is sent shortly after receiving the request. If the request is able to proceed, the <response> contains final status information.

Response Elements

The following elements are used in response messages.

error_info

Contains details of remote content retrieval errors, if enabled by the stop_on_error attribute of <prompt/>.

Response for: play, playcollect, playrecord

Values: Contains code, text, and context attributes that contain specific error information.

id

If an application-defined ID was specified in the request, that ID is returned to the application in the response.

Response for: play, playcollect, playrecord, stop

Response Attributes

code

Status of command.

Response for: play, playcollect, playrecord, stop

Values:

Value	Description
200	Command completed.
400	For playrecord: command not accepted due to error. The text attribute describes the cause of the error.
501	For playrecord: error because the URL type specified was not supported.

digits

Collected digits, if any.

Response for: playcollect, playrecord

playduration

Elapsed play time (in milliseconds) of a prompt or sequence before the request completed due to user intervention or end of sequence.

Response for: play, playcollect, playrecord, stop

reason

Cause for ending the play, playcollect, or playrecord command.

Response for: play

Values:

Value	Description
EOF	The play was completed when the end of file was reached.
stopped	A stop command, another command, or a SIP re-INVITE with hold media stopped the play.

Response for: playcollect

Values:

Value	Description
match	A digit was detected.
timeout	No digit was received in time.
returnkey	The return key terminated the operation.
escapekey	The escape key terminated the operation.
interrupted	Another request was received before the current one could finish.
stopped	A stop command, another command, of a SIP re-INVITE with hold media stopped the play.

Response for: playrecord

Values:

Value	Description
digit	A digit was detected.
endsilence	Trailing silence was detected.
init_silence	No voice (initial silence) was detected.
max_duration	Time for recording was complete.
stopped	A stop command, another command, or a SIP re-INVITE with hold media stopped the play.
escapekey	The escape key was entered either in the play mode or in the record mode. The application terminated the recording.
error	The operation failed.

reclength

Length of the recording in bytes.

Response for: playrecord

text

Whether the command succeeded.

Response for: play, playcollect, playrecord, stop

Error opening file for <playrecord> indicates requested record destination URL file could not be opened.

URL type is not implemented for <playrecord>. Indicates that the requested record destination URL file type is not supported.

6 - VoiceXML 2.0 and Dialog Service

This chapter provides information about VoiceXML 2.0 support.

The IP Media Server Release 2.2 supports both VoiceXML Version 2.0 and Voice XML 1.0. The default browser is VoiceXML 2.0. VoiceXML 2.0 provides an advance over VoiceXML 1.0, not so much in regards to functionality, but in terms of interoperability and clarity.

Note: Scripts that you have used with VoiceXML 1.0 may have to be updated to work VoiceXML 2.0.

- For reference information about VoiceXML Version 2.0, see the *VoiceXML Version 2.0 Reference* under the directory of that name on your documentation CD (open the file `VXML_2_0_Reference.html`).
- For reference information about VoiceXML Version 1.0, see "VoiceXML Version 1.0 and Dialog Service" (page 247).

Overview of VoiceXML 2.0

This section describes the notable enhancements in VoiceXML 2.0 relative to VoiceXML 1.0.

<grammar> VoiceXML 2.0 can be used for creating audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed initiative conversations. VoiceXML 2.0 supports the XML format of the grammar specification for speech and DTMF grammars. The <grammar> element also contains a mode attribute with values to indicate whether it is a voice grammar or a DTMF grammar. The <dtmf> element of VoiceXML 1.0 is replaced in VoiceXML 2.0 with a <grammar> element with its mode attribute set to dtmf. VoiceXML 2.0 platform supports this XML format for DTMF grammars. By using a single mandatory XML grammar format in VoiceXML 2.0, application developers can choose to write portable speech and DTMF grammars.

MRCP VoiceXML 2.0 supports Media Resource Control Protocol (MRCP) version 1.0. MRCP controls media service resources like speech synthesizers, recognizers, signal generators, signal detectors, fax servers, etc., over a network. This protocol is designed to work with streaming protocols like RTSP (Real Time Streaming Protocol) or SIP (Session Initiation Protocol) that help establish control connections to external media streaming devices and media delivery mechanisms like RTP (Real Time Protocol). MRCP integrates speech recognition and text to speech engines from Scansoft, Nuance, and IBM. Speech recognition software applications that listen to words spoken over a telephone and recognize these words can pass the recognized words as text to a Text-To-Speech (TTS) software application that synthesizes speech from application text for playback over a telephone.

**application .lastresult$** VoiceXML 2.0 defines a new variable, *application.lastresult$*, that provides information about the last recognition in the application. VoiceXML 1.0 does not provide a mechanism for inspecting the recognition result arising from grammars in the <link> element. If user input in an active dialog (typically a <field> inside a <form>) matched a grammar in a <link>, there is no way to evaluate its recognition result before executing its action. In VoiceXML 2.0, when user input matches a grammar in <field>, the result can be evaluated using the <field> element; for example, the developer can check the confidence by inspecting the field confidence shadow variable.

log The <log> element is new in VoiceXML 2.0. Developers typically use it during the debugging process to generate a debug message.

bargeintype VoiceXML 2.0 provides the developer with more control over the type of bargein performed by the platform with the new bargeintype attribute in the <prompt> element. This attribute has a number of values to determine how aggressively bargein is performed.

<choice>

VoiceXML 1.0 uses the text content of <choice> elements in <menu> to generate a grammar specifying sub-phrases. For example,

```
<menu>
 <prompt>
    Welcome home. Say one of: <enumerate/>
 </prompt>
 <choice ... > Sports news </choice>
 <choice ... > Weather news </choice>
 <choice ... > Stargazer astrophysics news </choice>
</menu>
```

The last <choice> would be matched if the user said phrases such as "Stargazer", "Stargazer news", "astrophysics news", and so forth. The exact grammar generation mechanism is language- and platform-dependent.

Although there are some use cases for this mechanism, there is also a strong use case for introducing a strict form of grammar generation. In VoiceXML 2.0 a <choice> is matched only if the user says exactly the content. If alternative phrases are required, these can be specified in multiple <choice>s. This option provides the developer with more control over what is recognized, making application behavior more consistent across platforms developed by different vendors.

accept

To provide this control, an accept attribute on the <menu> element was introduced in VoiceXML 2.0: with the value exact (the default) the element defines the exact phrase to be recognized, while the value approximate indicates the earlier "approximate" matching. The attribute is also defined on the <choice> element so specific <choice> elements can override the general <menu> strategy.

<throw>
<catch>

VoiceXML 2.0 features enhanced <throw> and <catch> to provide additional information. <throw> now has attributes to allow developers to specify additional information besides the event name; the message attribute is used to statically specify the additional information, whereas messageexpr dynamically specifies the information. In VoiceXML 1.0, one could not specify a handler for a general event type and then process specific event types in different ways. In VoiceXML 2.0, <catch> handlers have two new anonymous variables:

- _event Full name of the event that was thrown.
- _message Value of the message string from the corresponding <throw>.

<audio>

The <audio> element has been enhanced with an expr attribute. In addition to providing the capability to dynamically set the audio to be played back, this enhancement also allows <audio> element to be silently ignored if the expr attribute evaluates to ECMAScript undefined. Application developers can choose to use this feature to specify a list of <audio> elements in their document, where each <audio> element is only activated if expr has a defined value.

xml:lang In VoiceXML 2.0, the vxml lang attribute has been replaced with xml:lang to bring VoiceXML into alignment with other W3C XML languages. The application developer can specify the language for both spoken input and output by assigning this attribute a language value defined in [RFC1766].

<subdialog> The <subdialog> element provides a mechanism for decomposing complex sequences of dialogs to better structure them or to create reusable components. In VoiceXML 2.0, a subdialog context is independent of its calling dialog, but the subdialog context follows normal scoping rules for grammars, events, and variables.

Root and Leaf Documents

VoiceXML 1.0 somewhat lacked clarity in definition of, and transitions between, root and leaf documents. VoiceXML 2.0 explicitly defines these transitions in terms of <choice>, <goto>, <link>, <subdialog>, and <submit> elements and explains whether the application root context is preserved or initialized.

Conformance

VoiceXML 2.0 clarifies conformance both in terms of VoiceXML documents and in terms of VoiceXML processors. This aligns VoiceXML with other W3C specifications and the definitions are generally aligned with those in the Speech Grammar and Speech Synthesis specifications.

A conforming VoiceXML 2.0 document requires that it is a well-formed XML document, and that it provides a namespace declaration on the <vxml> element.

7 - Sample Code and Call Flows

This chapter provides examples and call flow diagrams for a variety of scenarios.

Sections in this chapter are:

Announcements

Play an Announcement as Early Media

The numbers in the left-hand column reference the Call Flow diagram in Figure 6 (page 176).

Example 20. Announcement as Early Media

1	INVITE sip:annc@192.168.12.155:5060;early=yes ;play= file:////opt/ snowshore/prompts/generic/10.ulaw SIP/2.0 Via: SIP/2.0/UDP 192.168.1.150:6100 To: <sip:annc@192.168.12.155:5060> From: <sip:test0@192.168.12.153:5060> Call-ID: 27125@192.168.1.150 Contact: sip:192.168.1.150:6100 CSeq: 2 INVITE Content-Type: application/sdp Supported: timer Supported: 100rel Session-Expires: 60 Content-Length: 153 v=0 o=SnowShoreUaV1 14250 3757 IN IP4 192.168.1.150 s=SnowShore Sdp t=0 0 m=audio 6000/1 RTP/AVP 0 c=IN IP4 192.168.12.154 a=sendrecv a=ptime:20
2	SIP/2.0 100 Trying Contact: sip:192.168.12.155:5060 Via: SIP/2.0/UDP 192.168.1.150:6100 To: <sip:annc@192.168.12.155:5060>;tag=100 5580780 From: <sip:test0@192.168.12.153:5060> Call-ID: 27125@192.168.1.150 CSeq: 2 INVITE Content-Length: 0

Example 20. Announcement as Early Media (continued)

```
3       SIP/2.0 183 Session Progress
        Contact: sip:192.168.12.155:5060
        Via: SIP/2.0/UDP 192.168.1.150:6100
        To:
           <sip:annc@192.168.12.155:5060>;tag=100
           5580780
        From: <sip:test0@192.168.12.153:5060>
        Call-ID: 27125@192.168.1.150
        CSeq: 2 INVITE
        Content-Type: application/sdp
        Require: 100rel
        RSeq: 22752
        Content-Length: 153

        v=0
        o=SnowShoreUaV1 27736 410 IN IP4
           192.168.12.155
        s=SnowShore Sdp
        t=0 0
        m=audio 4362/1 RTP/AVP 0
        c=IN IP4 192.168.12.156
        a=sendrecv
        a=ptime:20
4       SIP/2.0 200 OK
        Via: Sip/2.0/UDP 192.168.1.150:6100
        To:
           <sip:annc@192.168.12.155:5060>;tag=100
           5580780
        From: <sip:test0@192.168.12.153:5060>
        Call-ID: 27125@192.168.1.150
        CSeq: 3 PRACK
        Content-Length: 0
5       SIP/2.0 487 Request Terminated
        Contact: sip:192.168.12.155:5060
        Via: SIP/2.0/UDP 192.168.12.150:6100
        To:
           sip:annc@192.168.12.155:5060>;tag=1005
           580780
        From: sip:4444@192.168.12.153:5060
        Call-ID: 27125@192.168.1.150
        CSeq: 2 INVITE
        Content-Length: 0
```

Example 20. Announcement as Early Media (continued)

```
6          ACK sip:annc@192.168.12.155:5060 SIP/2.0
           Via: SIP/2.0/UDP 192.168.12.150:6100
           To:
              sip:annc@192.168.12.155:5060>;tag=1005
              580780
           From: sip:4444@192.168.12.153:5060
           Call-ID: 27125@192.168.1.150
           CSeq: 2 ACK
           Content-Length: 0
```

Call Flow for an Early Media Announcement

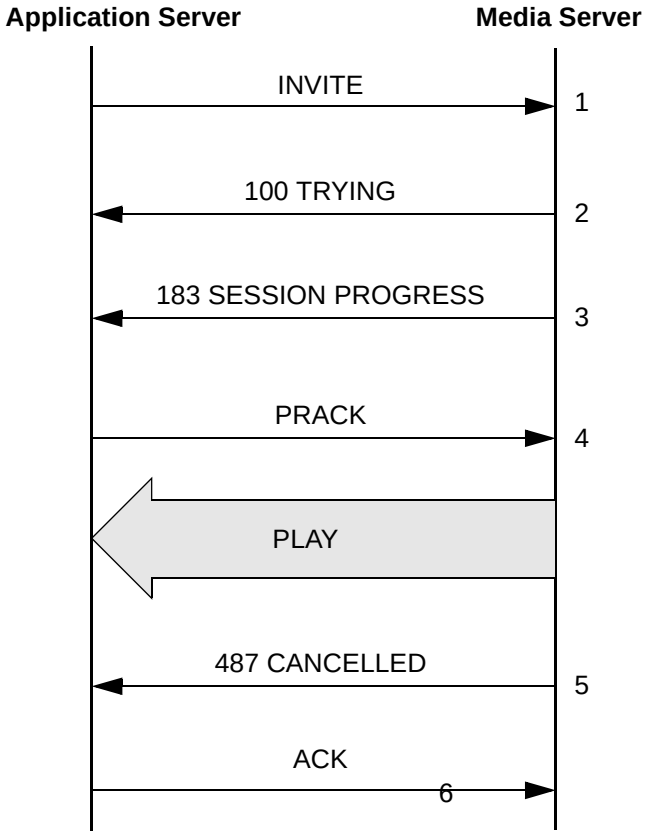

Figure 6. Call Flow: Announcement, Early Media

Playing an Announcement as Normal Media

The numbers in the left-hand column reference the diagram Figure 7 (page 179).

Example 21. Playing an Announcement as Normal Media

```
1          sip:annc@192.168.12.155:5060;play=file:///
              /opt/snowshore/
              prompts/generic/10.ulaw;early=no
              SIP/2.0
           Via: SIP/2.0/UDP 192.168.1.150:6100
           To: <sip:annc@192.168.12.155:5060>
           From: <sip:test0@192.168.12.153:5060>
           Call-ID: 27103@192.168.1.150
           Contact: sip:192.168.1.150:6100
           CSeq: 2 INVITE
           Content-Type: application/sdp
           Supported: timer
           Supported: 100rel
           Session-Expires: 60
           Content-Length: 154

           v=0
           o=SnowShoreUaV1 11845 20808 IN IP4
              192.168.1.150
           s=SnowShore Sdp
           t=0 0
           m=audio 6000/1 RTP/AVP 0
           c=IN IP4 192.168.12.154
           a=sendrecv
           a=ptime:20

2          SIP/2.0 100 Trying
           Contact: sip:192.168.12.155:5060
           Via: SIP/2.0/UDP 192.168.1.150:6100
           To:
              <sip:annc@192.168.12.155:5060>;tag=1005
              580654
           From: <sip:test0@192.168.12.153:5060>
           Call-ID: 27103@192.168.1.150
           CSeq: 2 INVITE
           Content-Length: 0
```

Example 21. Playing an Announcement as Normal Media (continued)

```
3          SIP/2.0 200 OK
           Contact: sip:192.168.12.155:5060
           Via: SIP/2.0/UDP 192.168.1.150:6100
           To:
              <sip:annc@192.168.12.155:5060>;tag=1005
              580654
           From: <sip:test0@192.168.12.153:5060>
           Call-ID: 27103@192.168.1.150
           CSeq: 2 INVITE
           Content-Type: application/sdp
           Session-Expires: 60
           Require: timer
           Content-Length: 152

           v=0
           o=SnowShoreUaV1 5987 313 IN IP4
              192.168.12.155
           s=SnowShore Sdp
           t=0 0
           m=audio 4360/1 RTP/AVP 0
           c=IN IP4 192.168.12.156
           a=sendrecv
           a=ptime:20

4          ACK sip:annc@192.168.12.155:5060 SIP/2.0
           Via: SIP/2.0/UDP 192.168.1.150:6100
           To:
              <sip:annc@192.168.12.155:5060>;tag=1005
              580654
           From: <sip:test0@192.168.12.153:5060>
           Call-ID: 27103@192.168.1.150
           CSeq: 2 ACK
           Content-Length: 0

5          BYE sip:annc@192.168.12.150:5060 SIP/2.0
           Via: SIP/2.0/UDP 192.168.1.150:6100
           To:
              <sip:test0@192.168.12.153:5060>;tag=100
              5580654
           From: <sip:annc@192.168.12.155:5060>
           Call-ID: 27103@192.168.1.150
           CSeq: 3 BYE
           Supported: timer
           Supported: 100rel
           Content-Length: 0
```

Example 21. Playing an Announcement as Normal Media (continued)

```
6          SIP/2.0 200 OK
           Via: SIP/2.0/UDP 192.168.1.150:6100
           To:
              <sip:test0@192.168.12.153:5060>;tag=100
              5580654
           From: <sip:annc@192.168.12.155:5060>
           Call-ID: 27103@192.168.1.150
           CSeq: 3 BYE
           Content-Length: 0
```

Call Flow for a Normal Media Announcement

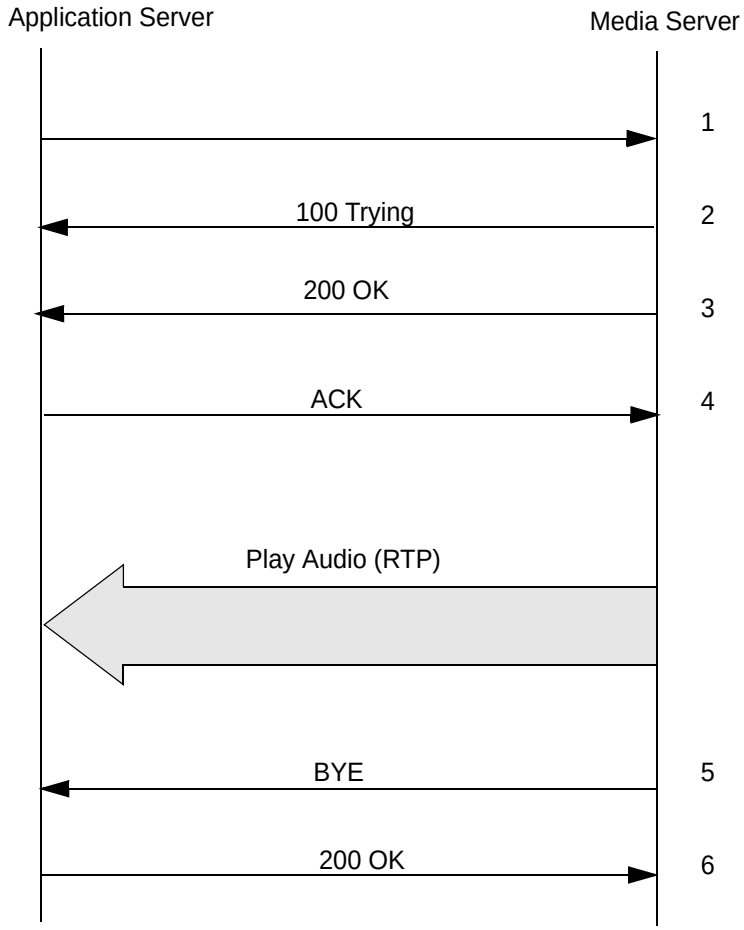

Figure 7. Call Flow: Announcement, Normal Media

Stopping Media—Hold

A SIP INVITE request places the remote party on hold.

Example 22. Stopping Media Using Hold

Note: Hold is represented with 0.0.0.0 on the line beginning with c=Re-INVITE.

[SIP header is not shown.]

```
v=0
o=anonymous 0 0 IN IP4 10.128.41.9
s=anonymous0
t=0 0
m=audio 5010 RTP/AVP 0 101
c=IN IP4 0.0.0.0
a=rtpmap:0 pcmu/8000
a=sendrecv
a=ptime:20
a=rtpmap:101 telephone-event/8000/1
m=video 5012 RTP/AVP 34
c=IN IP4 0.0.0.0
a=rtpmap:34 h263/90000
a=sendrecv
```

A successful response (200 OK) to the hold INVITE request confirms the on-hold state. There is nothing unique in the SDP.

```
200 OK
v=0
o=SnowShoreUaV1 846930886 1681692777 IN IP4
   10.128.41.223
s=SnowShore Sdp

t=0 0
m=audio 6000 RTP/AVP 0 101
c=IN IP4 10.128.41.223
a=sendrecv
a=ptime:20
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000/1
m=video 6002 RTP/AVP 34
c=IN IP4 10.128.41.223
a=sendrecv
a=ptime:20
a=rtpmap:34 H263/90000
```

re-INVITE to release the hold

[SIP header not shown]

```
v=0
o=anonymous 0 0 IN IP4 10.128.41.9
s=anonymous0
t=0 0
m=audio 5010 RTP/AVP 0 101
c=IN IP4 10.128.41.9
a=rtpmap:0 pcmu/8000
a=sendrecv
a=ptime:20
a=rtpmap:101 telephone-event/8000/1
m=video 5012 RTP/AVP 34
c=IN IP4 10.128.41.9
a=rtpmap:34 h263/90000
a=sendrecv
```

200 OK sent by sipd to release the HOLD.

[SIP header not shown]

```
v=0
o=SnowShoreUaV1 846930886 1681692777 IN IP4
   10.128.41.223
s=SnowShore Sdp
t=0 0
m=audio 0 RTP/AVP 0 101
c=IN IP4
a=sendrecv
a=ptime:20
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000/1
m=video 5012 RTP/AVP 34
c=IN IP4 10.128.41.9
a=sendrecv
a=ptime:20
a=rtpmap:34 H263/90000
```

Call Flow for Stopping Media—Hold

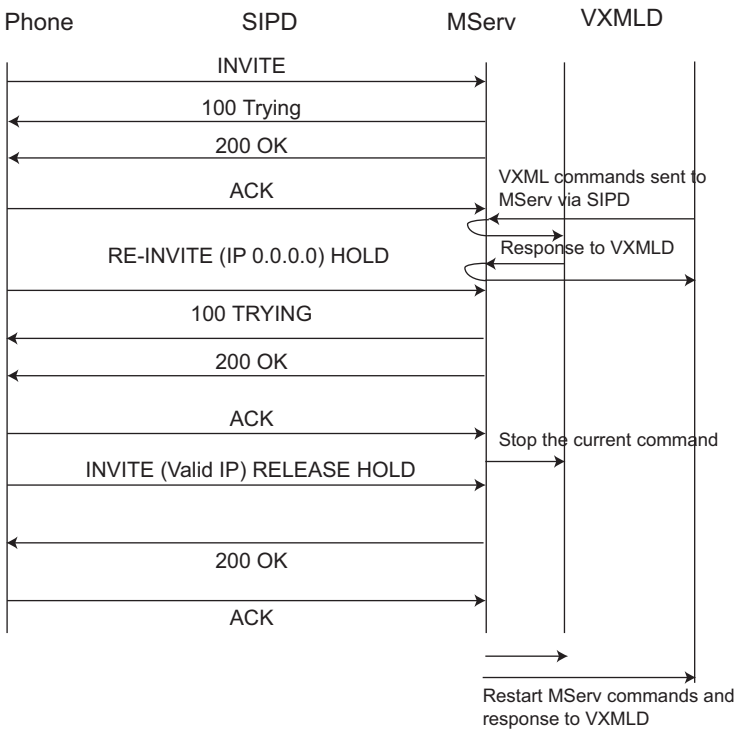

Figure 8. Call Flow: Stopping Media Using Hold

Repeating the Audio

Example 23. Repeating Audio Indefinitely

```
<request>
<play id=21212>
<prompt baseurl="file:////opt/snowshore/prompts/
generic/" locale="en_US" repeat="infinite"
duration="infinite" offset="0" delay="2000">
   <audio url="num_changed.wav"/>
   <audio url="new_number.wav"/>
   <variable type="dig" subtype="ndn"
value="9725551212"/>
   <audio url="make_note.wav"/>
</prompt>
</play>
</request>
```

Conferences

Creating a Simple Conference

Example 24. Simple Conference

```
INVITE sip:conf=1234@192.168.12.153 SIP/2.0
From: sip:threepcc@192.168.1.126;tag=1as8ut
To: sip:conf=1234@192.168.12.153
Call-ID: 1031579120515@192.168.1.126
CSeq: 1327320033 INVITE
Content-Length: 201
Content-Type: application/sdp
Contact: <sip:192.168.1.126:5060;
   transport=udp>
Via: SIP/2.0/UDP 192.168.1.126:5060

v=0
o=Pingtel 5 5 IN IP4 192.168.12.109
s=phone-call
c=IN IP4 192.168.12.109
t=0 0
m=audio 8770 RTP/AVP 0 96 8
a=rtpmap:0 pcmu/8000/1
a=rtpmap:96 telephone-event/8000/1
a=rtpmap:8 pcma/8000/1

SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060

To: sip:conf=1234@192.168.12.153;tag=
   1031665657
From: sip:threepcc@192.168.1.126;tag=1as8ut
Call-ID: 1031579120515@192.168.1.126
CSeq: 1327320033 INVITE
Contact: sip:192.168.12.153:5060
Content-Type:application/sdp
Session-Expires:120
Content-Length: 153
a=ptime:20

v=0
o=SnowShoreUaV1 22263 30720 IN IP4 192.168.12.153
s=SnowShore Sdp
t=0 0
m=audio 4202 RTP/AVP 0
c=IN IP4 192.168.12.154
a=sendrecv
```

Call Flow for a Simple Conference (Normal Media)

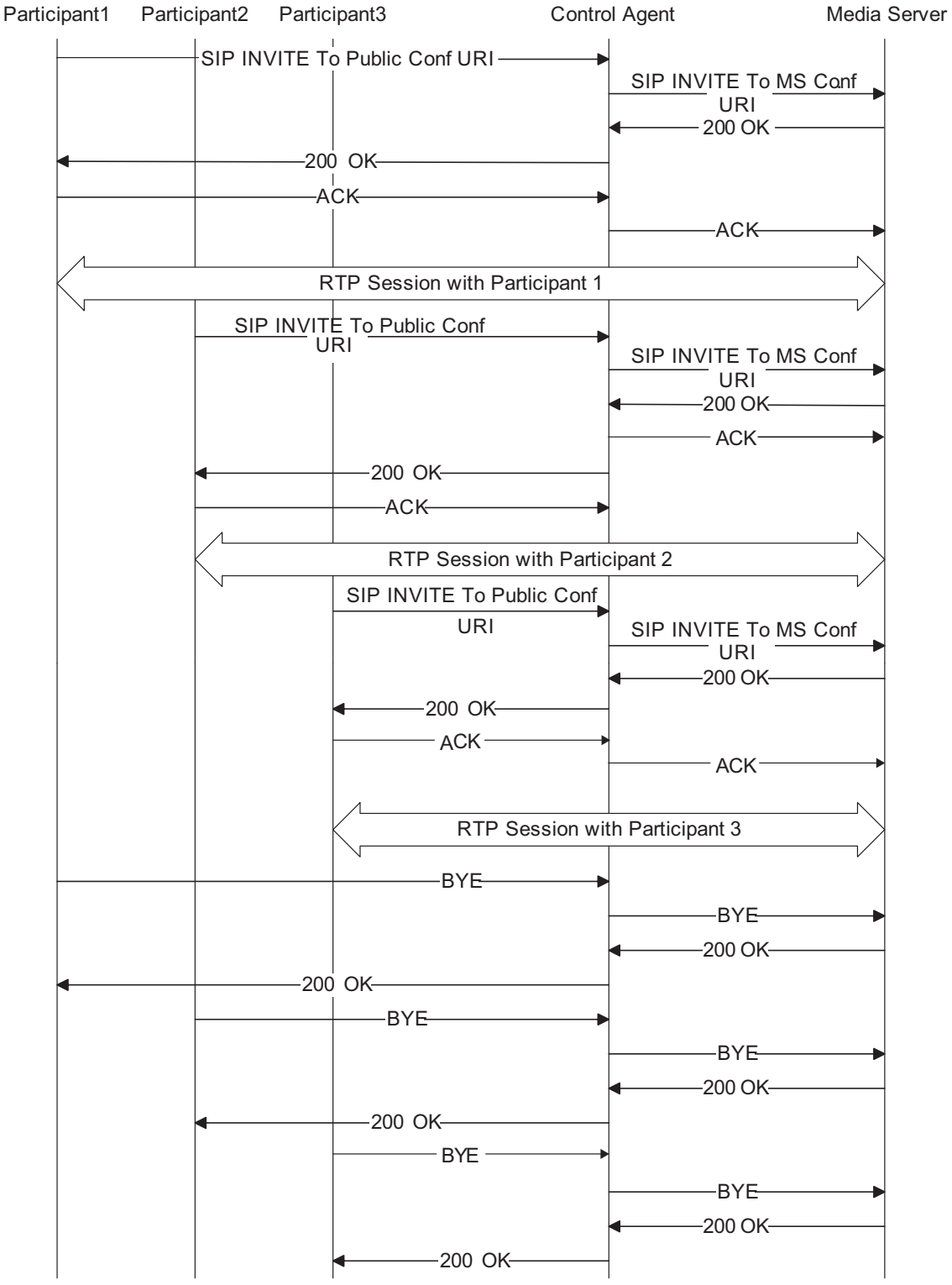

Figure 9. Call Flow, Simple Conference

Creating an Advanced Conference

Example 25. Advanced Conference

```
INVITE sip:conf=1234@192.168.12.153 SIP/2.0
From: sip:mscmltest@192.168.1.126;tag=
   7hl229
To: sip:conf=1234@192.168.12.153
Call-ID: 1031579500288@192.168.1.126
CSeq: 1597127738 INVITE
Content-Length: 504
Content-Type: multipart/mixed;boundary=snow-bound
Contact: <sip:192.168.1.126:5060;transport=
   udp>
Via: SIP/2.0/UDP 192.168.1.126:5060

--snow-bound
Content-type:application/mediaservercontrol+xml

<?xml version='1.0'?>
<MediaServerControl version='1.0'>
<request>
<configure_conference reserveconfmedia='yes'
   reservedtalkers='10' />
</request>
</MediaServerControl>
--snow-bound
Content-type:application/sdp

v=0
o=Pingtel 5 5 IN IP4 192.168.12.109
s=phone-call
c=IN IP4 0.0.0.0
t=0 0

--snow-bound--
```

Note: In compliance with RFC1341 (MIME), there is a blank line following the boundary marker and a CRLF separating each line in the multipart/mixed MSCML body.

Expected Response

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060
To: sip:conf=1234@192.168.12.153;tag=
   1031666037
From: sip:mscmltest@192.168.1.126;tag=
   7hl229
Call-ID: 1031579500288@192.168.1.126
CSeq: 1597127738 INVITE
```

```
Contact: sip:192.168.12.153:5060
Content-Type:multipart/mixed;boundary=
   snow-boundary-1
Session-Expires:120
Content-Length: 444

--snow-boundary-1
Content-Type: application/mediaservercontrol+xml

<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="configure_conference"
   code="200" text="OK"/>
</MediaServerControl>

--snow-boundary-1
Content-Type: application/sdp

v=0
o=SnowShoreUaV1 15485 15497 IN IP4 192.168.12.153
s=SnowShore Sdp
t=0 0
c=IN IP4 0.0.0.0

--snow-boundary-1--
```

Call Flow to Set up an Advanced Conference

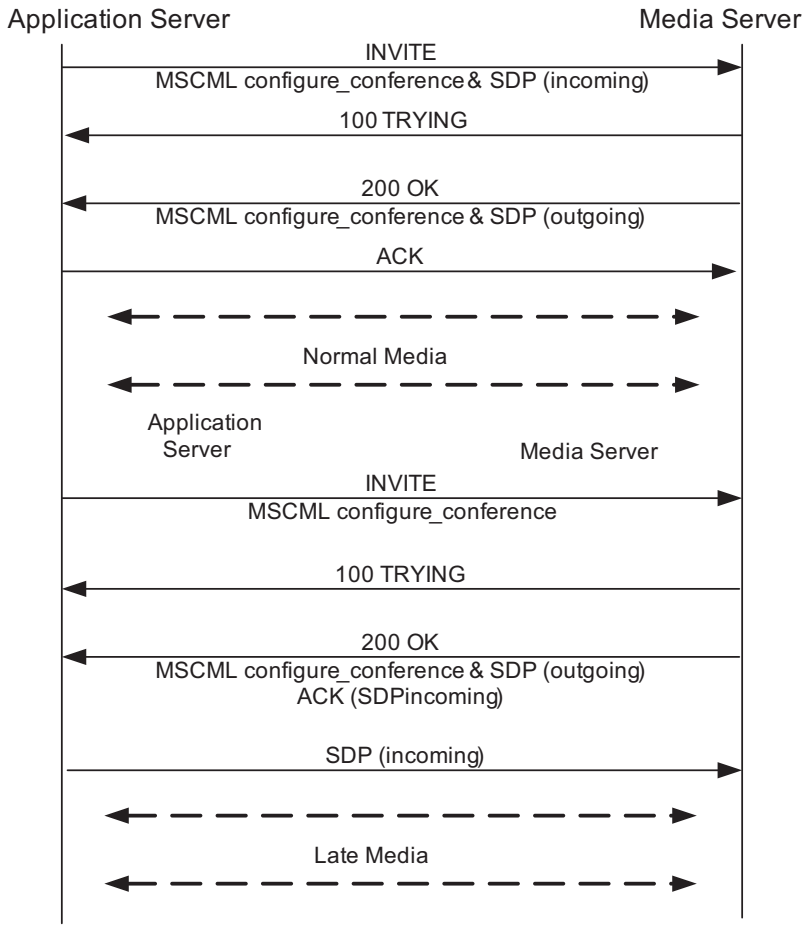

Figure 10. Call Flow: Advanced Conference

Modifying Conference Using Subscribe

Example 26. Modifying a Conference Using Subscribe

```
INFO sip:conf=1234@192.168.12.153 SIP/2.0
From: sip:mscmltest@192.168.1.126;tag=
   7hl229
To: sip:conf=1234@192.168.12.153
Call-ID: 1031579500288@192.168.1.126
CSeq: 1597127738 INFO
Content-Length: 504
Content-Type: application/mediaservercontrol+xml
Contact: <sip:192.168.1.126:5060;transport=
   udp>
Via: SIP/2.0/UDP 192.168.1.126:5060

<?xml version='1.0'?>
<MediaServerControl version="1.0">
   <request>
      <configure_conference>
         <subscribe>
            <events>
               <activetalkers report="yes"
            </events>
         </subscribe>
   </configure_conference>
   </request>
   </MediaServerControl>
```

Providing Communication for Participant in an Advanced Conference

Note: This example uses defaults.

Example 27. Adding Participant to Advanced Conference

```
INVITE sip:conf=1234@192.168.12.153 SIP/2.0
From: sip:threepcc@192.168.1.126;tag=jtopyd
To: sip:conf=1234@192.168.12.153
Call-ID: 1031580002825@192.168.1.126
CSeq: 2076031605 INVITE
Content-Length: 201
Content-Type: application/sdp
Contact: <sip:192.168.1.126:5060;transport=
   udp>
Via: SIP/2.0/UDP 192.168.1.126:5060

v=0
o=Pingtel 5 5 IN IP4 192.168.12.109
s=phone-call
c=IN IP4 192.168.12.109
t=0 0
m=audio 8770 RTP/AVP 0 96 8
a=rtpmap:0 pcmu/8000/1
a=rtpmap:96 telephone-event/8000/1
a=rtpmap:8 pcma/8000/1
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060
To: sip:conf=1234@192.168.12.153;tag=
   1
   031666539
From: sip:threepcc@192.168.1.126;tag=
   jtopyd
Call-ID: 1031580002825@192.168.1.126
CSeq: 2076031605 INVITE
Contact: sip:192.168.12.153:5060
Content-Type:application/sdp
Session-Expires:120
Content-Length: 152

v=0
o=SnowShoreUaV1 7030 13061 IN IP4 192.168.12.153
s=SnowShore Sdp
t=0 0
m=audio 4208 RTP/AVP 0
c=IN IP4 192.168.12.154
a=sendrecv
a=ptime:20
```

Joining a Participant Using Special Attributes

Example 28. Joining Participant Using Special Attributes

```
INVITE sip:conf=1234@9.9.3.1 SIP/2.0
From: sip:confbridge@10.10.10.150;tag=4qjtic
To: sip:conf=1234@9.9.3.1
Call-ID: 1031256911282@10.10.10.150
CSeq: 1887335379 INVITE
Content-Length: 481
Content-Type: multipart/mixed;
   boundary=confbridge-bound
Contact: <sip:10.10.10.150:5060;
   transport=udp>
Via: SIP/2.0/UDP 10.10.10.150:5060

--confbridge-bound
content-type:application/sdp

v=0
o=Pingtel 5 5 IN IP4 10.10.10.16
s=phone-call
c=IN IP4 10.10.10.16
t=0 0
m=audio 8766 RTP/AVP 0 96 8
a=rtpmap:0 pcmu/8000/1
a=rtpmap:96 telephone-event/8000/1
a=rtpmap:8 pcma/8000/1
--confbridge-bound
content-type:application/mediaservercontrol+xml

<?xml version='1.0'?>
<MediaServerControl version='1.0'>
<request>
<configure_leg mixmode='private' />
</request>
</MediaServerControl>
--confbridge-bound--
```

Suspending Communications within a Conference

Example 29. Suspending Communications during a Conference (Continued)

```
INFO sip:192.168.12.153:5060 SIP/2.0
From: sip:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407038 INFO
Content-Length: 142

Content-Type: application/mediaservercontrol+xml
Via: SIP/2.0/UDP 192.168.1.126:5060

<?xml version='1.0'?>
<MediaServerControl version='1.0'>
<request>
<configure_leg mixmode='mute' />
</request>
</MediaServerControl>

SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060
To: sip:conf=1234@192.168.12.153;tag=
   1031666918
From: sip:confbridge@192.168.1.126;tag=
   oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407038 INFO
Content-Length: 0
```

Response to Mute a Conference Participant

Example 30. Response to Muting a Participant

```
INFO sip:confbridge@192.168.1.126:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.153:5060

To: sip:confbridge@192.168.1.126;tag=oi32zw
From: sip:conf=1234@192.168.12.153;tag=
   1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 15398 INFO
Content-Type:application/mediaservercontrol+xml
Supported:timer
Supported:100rel
Content-Length: 137
<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="configure_leg" code="200"
   text="OK" />
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.12.153:5060
From: sip:conf=1234@192.168.12.153;tag=
   1031666918
To: sip:confbridge@192.168.1.126;tag=oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 15398 INFO
Content-Length: 0
```

Playing Audio to Conference Participant

Changing Mixmode to Parked

Example 31. Changing Participant Mixmode to Parked

```
INFO sip:192.168.12.153:5060 SIP/2.0
From: sip:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407032 INFO
Content-Length: 144
Content-Type: application/mediaservercontrol+xml
Via: SIP/2.0/UDP 192.168.1.126:5060
<?xml version='1.0'?>
<MediaServerControl version='1.0'>
<request>
<configure_leg mixmode='parked' />
</request>
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060
To: sip:conf=1234@192.168.12.153;tag=
   1031666918

From: sip:confbridge@192.168.1.126;tag=
   oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407032 INFO
Content-Length: 0
```

Response to Parked

```
INFO sip:confbridge@192.168.1.126:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.153:5060
To: sip:confbridge@192.168.1.126;tag=oi32zw
From: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 15392 INFO
Content-Type:application/mediaservercontrol+xml
Supported:timer
Supported:100rel
Content-Length: 137
<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="configure_leg" code="200"
   text="OK" />
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.12.153:5060
From: sip:conf=1234@192.168.12.153;tag=1031666918
To: sip:confbridge@192.168.1.126;tag=oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 15392 INFO
Content-Length: 0
```

Playing the Audio

Example 32. Playing Audio

```
INFO sip:192.168.12.153:5060 SIP/2.0
From: sip:confbridge@192.168.1.126;tag=
   oi32zw
To: sip:conf=1234@192.168.12.153;tag=
   1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407033 INFO
Content-Length: 194
Content-Type: application/mediaservercontrol+xml
Via: SIP/2.0/UDP 192.168.1.126:5060
<?xml version='1.0'?>
<MediaServerControl version='1.0'>
<request>
<play prompt='http://192.168.1.126:8013/
   confbridge/audio/help.raw' promptencoding='ulaw' />
</request>
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060
To: sip:conf=1234@192.168.12.153;tag=
   1031666918
```

```
From: sip:confbridge@192.168.1.126;tag=
   oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407033 INFO
Content-Length: 0
```

Response to Message to Play Audio

Example 33. Response to Message to Play Audio

```
INFO sip:confbridge@192.168.1.126:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.153:5060
To: sip:confbridge@192.168.1.126;tag=oi32zw
From: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 15393 INFO
Content-Type:application/mediaservercontrol+xml
Supported:timer
Supported:100rel
Content-Length: 160
<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="play" code="200" text="OK"
   reason="EOF" playduration="8890" />
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.12.153:5060
From: sip:conf=1234@192.168.12.153;tag=1031666918
To: sip:confbridge@192.168.1.126;tag=oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 15393 INFO
Content-Length: 0
```

Changing Participant Mixmode Back to Full

Example 34. Changing Mixmode Back to Full

```
INFO sip:192.168.12.153:5060 SIP/2.0
From: sip:confbridge@192.168.1.126;tag=
   oi32zw
To: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407034 INFO
Content-Length: 142
Content-Type: application/mediaservercontrol+xml
Via: SIP/2.0/UDP 192.168.1.126:5060
<?xml version='1.0'?>
<MediaServerControl version='1.0'>
<request>
```

```
<configure_leg mixmode='full' />
</request>
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060

To: sip:conf=1234@192.168.12.153;tag=
   1031666918
From: sip:confbridge@192.168.1.126;tag=
   oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407034 INFO
Content-Length: 0
```

Response to Mixmode Change

Example 35. Response to Mixmode Change

```
INFO sip:confbridge@192.168.1.126:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.153:5060
To: sip:confbridge@192.168.1.126;tag=oi32zw
From: sip:conf=1234@192.168.12.153;tag=
   1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 15394 INFO
Content-Type:application/mediaservercontrol+xml
Supported:timer
Supported:100rel
Content-Length: 137
<?xml version="1.0"?>
<MediaServerControl version="1.0">
<response request="configure_leg" code="200" text="OK"
   />
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.12.153:5060
From: sip:conf=1234@192.168.12.153;tag=
   1031666918
To: sip:confbridge@192.168.1.126;tag=oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 15394 INFO
Content-Length: 0
```

Playcollect and Playrecord in a Conference

Changing Participant Mixmode to Parked

Example 36. Changing Mixmode to Parked Mode

```
INFO sip:192.168.12.153:5060 SIP/2.0
From: sip:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407032 INFO
Content-Length: 144
Content-Type: application/mediaservercontrol+xml
Via: SIP/2.0/UDP 192.168.1.126:5060
<?xml version='1.0'?>
<MediaServerControl version='1.0'>
<request>
<configure_leg mixmode='parked' />
</request>
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060
To: sip:conf=1234@192.168.12.153;tag=
   1031666918
From: sip:confbridge@192.168.1.126;tag=
   oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407032 INFO
Content-Length: 0
```

Response to Parked

Example 37. Response to Parked

```
INFO sip:confbridge@192.168.1.126:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.153:5060
To: sip:confbridge@192.168.1.126;tag=oi32zw
From: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 15392 INFO
Content-Type:application/mediaservercontrol+xml
Supported:timer
Supported:100rel
Content-Length: 137
<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="configure_leg" code="200"
   text="OK" />
</MediaServerControl>
```

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.12.153:5060
From: sip:conf=1234@192.168.12.153;tag=1031666918
To: sip:confbridge@192.168.1.126;tag=oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 15392 INFO
Content-Length: 0
```

PlayCollect

```
INFO sip:192.168.12.153:5060 SIP/2.0
From: sip:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407027 INFO
Content-Length: 237
Content-Type: application/mediaservercontrol+xml
Via: SIP/2.0/UDP 192.168.1.126:5060
<?xml version='1.0'?>
<MediaServerControl version='1.0'>
<request>
<playcollect
   prompt='http://192.168.1.126:8013/confbridge/audio/
askauthcode.raw' barge='yes' cleardigits='yes'
   maxdigits='4' />
</request>
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060
From: sip:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407027 INFO
Content-Length: 0
```

Response to PlayCollec

```
INFO sip:confbridge@192.168.1.126:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.153:5060
To: sip:confbridge@192.168.1.126;tag=oi32zw
From: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 15387 INFO
Content-Type:application/mediaservercontrol+xml
Supported:timer
Supported:100rel
Content-Length: 183

<?xml version="1.0"?>
<MediaServerControl version="1.0">
```

```
        <response request="playcollect" code="200"
   text="OK" reason="timeout" digits="85"
   playduration="2260"/>
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.12.153:5060
From: sip:conf=1234@192.168.12.153;tag=1031666918
To: sip:confbridge@192.168.1.126;tag=oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 15387 INFO
Content-Length: 0
```

Sending PlayRecord

```
INFO sip:192.168.12.153:5060 SIP/2.0
From: sip:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407028 INFO
Content-Length: 300
Content-Type: application/mediaservercontrol+xml
Via: SIP/2.0/UDP 192.168.1.126:5060
<?xml version='1.0'?>
<MediaServerControl version='1.0'>
<request>
<playrecord
   prompt='http://192.168.1.126:8013/confbridge/audio/
   tellyourname.raw' promptencoding='ulaw' barge='yes'
   cleardigits='yes'
         recurl='file:////var/snowshore/rec/temp.ulaw'
   recencoding='ulaw' />
</request>
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060
From: sip:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407028 INFO
Content-Length: 0
```

Response to PlayRecord

```
INFO sip:confbridge@192.168.1.126:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.153:5060
To: sip:confbridge@192.168.1.126;tag=oi32zw
From: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 15388 INFO
Content-Type:application/mediaservercontrol+xml
Supported:timer
Supported:100rel
```

```
Content-Length: 197
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <response request="playrecord" code="200" text="OK"
      reason="digit" digits="#" reclength="14320"
      playduration="4820" />
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.12.153:5060
From: sip:conf=1234@192.168.12.153;tag=1031666918
To: sip:confbridge@192.168.1.126;tag=oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 15388 INFO
Content-Length: 0
```

Changing Participant Back to Full

```
INFO sip:192.168.12.153:5060 SIP/2.0
From: sip:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407030 INFO
Content-Length: 142
Content-Type: application/mediaservercontrol+xml
Via: SIP/2.0/UDP 192.168.1.126:5060
<?xml version='1.0'?>
<MediaServerControl version='1.0'>
<request>
<configure_leg mixmode='full' />
</request>
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.126:5060
From: sip:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407030 INFO
Content-Length: 0
```

Response to Full

```
INFO sip:confbridge@192.168.1.126:5060 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.153:5060
To: sip:confbridge@192.168.1.126;tag=oi32zw
From: sip:conf=1234@192.168.12.153;tag=1031666918
Call-ID: 1031580381757@192.168.1.126
CSeq: 15390 INFO
Content-Type:application/mediaservercontrol+xml
Supported:timer
Supported:100rel
Content-Length: 137
<?xml version="1.0"?>
```

```
<MediaServerControl version="1.0">
   <response request="configure_leg" code="200"
   text="OK" />
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.12.153:5060
From: sip:conf=1234@192.168.12.153;tag=1031666918
To: sip:confbridge@192.168.1.126;tag=oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 15390 INFO
Content-Length: 0
```

Coached Conferencing

Example 38. Coached Conferencing

1 Invitation for control leg of enhanced conference

```
INVITE sip:conf=1234@192.168.12.150:5060 SIP/2.0
From: sip:control@192.168.12.158:5558;tag=1
To: sip:conf=1234@192.168.12.150:5060
Subject: Call from Control
Cseq: 1 INVITE
Call-ID: 1119358373
Contact: <sip:control@192.168.12.158:5558>
Content-Type: multipart/mixed;boundary=salem
Via: SIP/2.0/UDP 192.168.12.158:5558

--salem
Content-Type: application/sdp

v=0
o=mserv 1 1119358373 IN IP4 192.168.12.151
s=Riptide_Media_Server
c=IN IP4 0.0.0.0
t=0 0
m=audio 0 RTP/AVP 0

--salem
Content-Type: application/mediaservercontrol+xml

<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_conference reservedtalkers="20"
        reserveconfmedia="no">
      </configure_conference>
   </request>
</MediaServerControl>

--salem--
```

First response from IP Media Server:

```
SIP/2.0 100 Trying
Contact: sip:192.168.12.150:5060
Server: SnowShoreMediaServer/1.4.0-050614A
Via: SIP/2.0/UDP 192.168.12.158:5558
To: sip:conf=1234@192.168.12.150:5060;tag=1119358026
From: sip:control@192.168.12.158:5558;tag=1
Call-ID: 1119358373
CSeq: 1 INVITE
Content-Length: 0
```

Second response from IP Media Server:

```
SIP/2.0 200 Ok
Server: SnowShoreMediaServer/1.4.0-050614A
Via: SIP/2.0/UDP 192.168.12.158:5558
To: sip:conf=1234@192.168.12.150:5060;tag=1119358026
From: sip:control@192.168.12.158:5558;tag=1
Call-ID: 1119358373
CSeq: 1 INVITE
Contact: sip:192.168.12.150:5060
Content-Type: multipart/mixed;boundary=snow-boundary-1
Session-Expires: 6000
Content-Length: 466

--snow-boundary-1
Content-Type: application/mediaservercontrol+xml

<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="configure_conference"
   code="200" text="OK"/>
</MediaServerControl>
--snow-boundary-1
Content-Type: application/sdp

v=0
o=SnowShoreUaV1 1102520059 2044897763 IN IP4
   192.168.12.150
s=SnowShore Sdp
t=0 0
m=audio 0 RTP/AVP 0
c=IN IP4 0.0.0.0
a=sendrecv
a=ptime:20
a=rtpmap:0 PCMU/8000

--snow-boundary-1--
```

Customer Leg Invitation

```
INVITE sip:conf=1234@192.168.12.150:5060 SIP/2.0
From: sip:control@192.168.12.158:5560;tag=1
To: sip:conf=1234@192.168.12.150:5060
Subject: Call from Control
Cseq: 1 INVITE
Call-ID: 1119358621
Contact: <sip:control@192.168.12.158:5560>
Content-Type: multipart/mixed;boundary=salem
Via: SIP/2.0/UDP 192.168.12.158:5560

--salem
Content-Type: application/sdp
v=0
o=mserv 1 1119358621 IN IP4 192.168.12.151
s=Riptide_Media_Server
c=IN IP4 192.168.12.155
t=0 0
m=audio 8766 RTP/AVP 0

--salem
Content-Type: application/mediaservercontrol+xml

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

--salem--
```

IP Media Server response to client application:

```
SIP/2.0 200 Ok
Server: SnowShoreMediaServer/1.4.0-050614A
Via: SIP/2.0/UDP 192.168.12.158:5560
To: sip:conf=1234@192.168.12.150:5060;
   tag=1119358274
From: sip:control@192.168.12.158:5560;tag=1
Call-ID: 1119358621
CSeq: 1 INVITE
Contact: sip:192.168.12.150:5060
Content-Type: multipart/mixed;boundary=
   snow-boundary-1
Session-Expires: 6000
Content-Length: 469

--snow-boundary-1
Content-Type: application/mediaservercontrol+xml
```

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="configure_leg" code="200"
   text="OK"/>
</MediaServerControl>
--snow-boundary-1
Content-Type: application/sdp

v=0
o=SnowShoreUaV1 1365180540 1540383426 IN IP4
   192.168.12.150
s=SnowShore Sdp
t=0 0
m=audio 6006 RTP/AVP 0
c=IN IP4 192.168.12.150
a=sendrecv
a=ptime:20
a=rtpmap:0 PCMU/8000
--snow-boundary-1--
-----------------------------------------
```

2 Agent Invite

```
INVITE sip:conf=1234@192.168.12.150:5060 SIP/2.0
From: sip:control@192.168.12.158:5562;tag=1
To: sip:conf=1234@192.168.12.150:5060
Subject: Call from Control
Cseq: 1 INVITE
Call-ID: 1119358826
Contact: <sip:control@192.168.12.158:5562>
Content-Type: multipart/mixed;boundary=salem
Via: SIP/2.0/UDP 192.168.12.158:5562

--salem
Content-Type: application/sdp

v=0
o=mserv 1 1119358826 IN IP4 192.168.12.151
s=Riptide_Media_Server
c=IN IP4 192.168.12.156
t=0 0
m=audio 8766 RTP/AVP 0

--salem
Content-Type: application/mediaservercontrol+xml
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_leg id="agent"/>
   </request>
</MediaServerControl>
```

```
--salem--
```

Response to agent:

```
SIP/2.0 200 Ok
Server: SnowShoreMediaServer/1.4.0-050614A
Via: SIP/2.0/UDP 192.168.12.158:5562
To: sip:conf=1234@192.168.12.150:5060;
   tag=1119358479
From: sip:control@192.168.12.158:5562;tag=1
Call-ID: 1119358826
CSeq: 1 INVITE
Contact: sip:192.168.12.150:5060
Content-Type: multipart/mixed;boundary=snow-boundary-1
Session-Expires: 6000
Content-Length: 467
--snow-boundary-1
Content-Type: application/mediaservercontrol+xml

<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="configure_leg" code="200"
   text="OK"/>
</MediaServerControl>
--snow-boundary-1
Content-Type: application/sdp

v=0
o=SnowShoreUaV1 1303455736 35005211 IN IP4
   192.168.12.150
s=SnowShore Sdp
t=0 0
m=audio 6008 RTP/AVP 0
c=IN IP4 192.168.12.150
a=sendrecv
a=ptime:20
a=rtpmap:0 PCMU/8000

--snow-boundary-1--
----------------------------------------
```

3 Supervisor- invited with ID and put into private mode

```
INVITE sip:conf=1234@192.168.12.150:5060 SIP/2.0
From: sip:control@192.168.12.158:5564;tag=1
To: sip:conf=1234@192.168.12.150:5060
Subject: Call from Control
Cseq: 1 INVITE
Call-ID: 1119359020
Contact: <sip:control@192.168.12.158:5564>
Content-Type: multipart/mixed;boundary=salem
Via: SIP/2.0/UDP 192.168.12.158:5564

--salem
```

```
Content-Type: application/sdp

v=0
o=mserv 1 1119359020 IN IP4 192.168.12.151
s=Riptide_Media_Server
c=IN IP4 192.168.12.157
t=0 0
m=audio 32126 RTP/AVP 0

--salem
Content-Type: application/mediaservercontrol+xml

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

Response from Media Server:

```
SIP/2.0 200 Ok
Server: SnowShoreMediaServer/1.4.0-050614A
Via: SIP/2.0/UDP 192.168.12.158:5564
To: sip:conf=1234@192.168.12.150:5060;
   tag=1119358673
From: sip:control@192.168.12.158:5564;tag=1
Call-ID: 1119359020
CSeq: 1 INVITE
Contact: sip:192.168.12.150:5060
Content-Type: multipart/mixed;boundary=snow-boundary-1
Session-Expires: 6000
Content-Length: 468

--snow-boundary-1
Content-Type: application/mediaservercontrol+xml


<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="configure_leg" code="200"
   text="OK"/>
</MediaServerControl>
--snow-boundary-1
Content-Type: application/sdp
v=0
o=SnowShoreUaV1 294702567 1726956429
   IN IP4 192.168.12.150
s=SnowShore Sdp
t=0 0
m=audio 6010 RTP/AVP 0
c=IN IP4 192.168.12.150
a=sendrecv
```

```
a=ptime:20
a=rtpmap:0 PCMU/8000

--snow-boundary-1--
----------------------------------------
```

4 Supervisor makes teammate of agent and can start coach conference with INFO message

```
INFO sip:192.168.12.150:5060 SIP/2.0
From: sip:control@192.168.12.158:5564;tag=1
To: sip:conf=1234@192.168.12.150:5060;
   tag=1119358673
Call-ID: 1119359020
Cseq: 2 INFO
Via: SIP/2.0/UDP 192.168.12.158:5564
Content-Type: application/mediaservercontrol+xml
Content-Length: 190
<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_team action="add">
         <teammate id="agent" />
      </configure_team>
   </request>
</MediaServerControl>
```

Response from IP Media Server:

```
INFO sip:control@192.168.12.158:5564 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.150:5060
To: sip:control@192.168.12.158:5564;tag=1
From: sip:conf=1234@192.168.12.150:5060;
   tag=1119358673
Call-ID: 1119359020
User-Agent: SnowShoreMediaServer/1.4.0-050614A
CSeq: 2 INFO
Content-Type: application/mediaservercontrol+xml
Supported: timer
Supported: 100rel
Content-Length: 231

<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <response request="configure_team" code="200" text="OK">
      <team id="super" numteam="1">
         <teammate id="agent">
         </teammate>
      </team>
   </response>
</MediaServerControl>
----------------------------------------
```

5 Supervisor goes directly into coached conferencing in one step using the INVITE

```
INVITE sip:conf=1234@192.168.12.150:5060 SIP/2.0
From: sip:control@192.168.12.158:5564;tag=1
To: sip:conf=1234@192.168.12.150:5060
Subject: Call from Control
Cseq: 1 INVITE
Call-ID: 1119359610
Contact: <sip:control@192.168.12.158:5564>
Content-Type: multipart/mixed;boundary=salem
Via: SIP/2.0/UDP 192.168.12.158:5564
--salem
Content-Type: application/sdp

v=0
o=mserv 1 1119359610 IN IP4 192.168.12.151
s=Riptide_Media_Server
c=IN IP4 192.168.12.157
t=0 0
m=audio 32128 RTP/AVP 0
--salem
Content-Type: application/mediaservercontrol+xml

<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_leg id="super" mixmode="private">
         <configure_team action="set">
            <teammate id="agent"/>
         </configure_team>
      </configure_leg>
   </request>
</MediaServerControl>

--salem--
```

Response from IP Media Server:

```
SIP/2.0 200 Ok
Server: SnowShoreMediaServer/1.4.0-050614A
Via: SIP/2.0/UDP 192.168.12.158:5564
To: sip:conf=1234@192.168.12.150:5060;
   tag=1119359262
From: sip:control@192.168.12.158:5564;tag=1
Call-ID: 1119359610
CSeq: 1 INVITE
Contact: sip:192.168.12.150:5060

Content-Type: multipart/mixed;boundary=snow-boundary-1
Session-Expires: 6000
Content-Length: 549
```

```
--snow-boundary-1
Content-Type: application/mediaservercontrol+xml

<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <response request="configure_leg" code="200" text="OK"/>
      <team id="super" numteam="1">
         <teammate id="agent">
         </teammate>
      </team>
</MediaServerControl>
--snow-boundary-1
Content-Type: application/sdp

v=0
o=SnowShoreUaV1 861021530 278722862 IN IP4
   192.168.12.150
s=SnowShore Sdp
t=0 0
m=audio 6012 RTP/AVP 0
c=IN IP4 192.168.12.150
a=sendrecv
a=ptime:20
a=rtpmap:0 PCMU/8000
--snow-boundary-1
-----------------------------------------
```

6 A query on team list from the supervisor's leg

```
INFO sip:192.168.12.150:5060 SIP/2.0
From: sip:control@192.168.12.158:5564;
   tag=1
To: sip:conf=1234@192.168.12.150:5060;
   tag=1119359262
Call-ID: 1119359610
Cseq: 2 INFO
Via: SIP/2.0/UDP 192.168.12.158:5564
Content-Type: application/mediaservercontrol+xml
Content-Length: 167

<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_team action="query">
      </configure_team>
   </request>
</MediaServerControl>
```

Response from IP Media Server:

```
INFO sip:control@192.168.12.158:5564 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.150:5060
```

```
To: sip:control@192.168.12.158:5564;tag=1
From: sip:conf=1234@192.168.12.150:5060;
   tag=1119359262
Call-ID: 1119359610
User-Agent: SnowShoreMediaServer/
   1.4.0-050614A
CSeq: 2 INFO
Content-Type: application/mediaservercontrol+xml
Supported: timer
Supported: 100rel
Content-Length: 231

<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <response request="configure_team" code="200"
      text="OK">
      <team id="super" numteam="1">
         <teammate id="agent">
         </teammate>
      </team>
   </response>
</MediaServerControl>

----------------------------------------
```

7 Supervisor clears entire team list.

```
INFO sip:192.168.12.150:5060 SIP/2.0
From: sip:control@192.168.12.158:5564;tag=1
To: sip:conf=1234@192.168.12.150:5060;
   tag=1119359262
Call-ID: 1119359610
Cseq: 3 INFO
Via: SIP/2.0/UDP 192.168.12.158:5564
Content-Type: application/mediaservercontrol+xml
Content-Length: 147

<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_team action="set"/>
   </request>
</MediaServerControl>
```

Response from the IP Media Server:

```
INFO sip:control@192.168.12.158:5564 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.150:5060
To: sip:control@192.168.12.158:5564;tag=1

From: sip:conf=1234@192.168.12.150:5060;tag=1119359262
Call-ID: 1119359610
User-Agent: SnowShoreMediaServer/1.4.0-050614A
CSeq: 3 INFO
```

```
      Content-Type: application/mediaservercontrol+xml
      Supported: timer
      Supported: 100rel
      Content-Length: 191

      <?xml version="1.0"?>
      <MediaServerControl version="1.0">
         <response request="configure_team" code="200"
            text="OK">
            <team id="super" numteam="0">
            </team>
         </response>
      </MediaServerControl>
---------------------------------------
```

8 Agent queries who is on the team

```
      INFO sip:192.168.12.150:5060 SIP/2.0
      From: sip:control@192.168.12.158:5562;
         tag=1
      To: sip:conf=1234@192.168.12.150:5060;
         tag=1119358479
      Call-ID: 1119358826
      Cseq: 2 INFO
      Via: SIP/2.0/UDP 192.168.12.158:5562
      Content-Type: application/mediaservercontrol+xml
      Content-Length: 167

      <?xml version="1.0"?>
      <MediaServerControl version="1.0">
         <request>
            <configure_team action="query">
            </configure_team>
         </request>
      </MediaServerControl>
```

Response from IP Media Server:

```
      INFO sip:control@192.168.12.158:5562 SIP/2.0
      Via: SIP/2.0/UDP 192.168.12.150:5060
      To: sip:control@192.168.12.158:5562;tag=1
      From: sip:conf=1234@192.168.12.150:5060;
         tag=1119358479
      Call-ID: 1119358826
      User-Agent: SnowShoreMediaServer/1.4.0-050614A
      CSeq: 2 INFO
      Content-Type: application/mediaservercontrol+xml
      Supported: timer
      Supported: 100rel
      Content-Length: 191
      <?xml version="1.0"?>
      <MediaServerControl version="1.0">
         <response request="configure_team" code="200"
            text="OK">
```

```
      <team id="agent" numteam="0">
      </team>
   </response>
</MediaServerControl>

----------------------------------------
```

9 Supervisor is added to the conference: mixmode=full

```
INFO sip:192.168.12.150:5060 SIP/2.0
From: sip:control@192.168.12.158:5564;tag=1
To: sip:conf=1234@192.168.12.150:5060;
   tag=1119359262
Call-ID: 1119359610
Cseq: 4 INFO
Via: SIP/2.0/UDP 192.168.12.158:5564
Content-Type: application/mediaservercontrol+xml
Content-Length: 148

<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <request>
      <configure_leg mixmode="full"/>
   </request>
</MediaServerControl>
```

Response from IP Media Server:

```
INFO sip:control@192.168.12.158:5564 SIP/2.0
Via: SIP/2.0/UDP 192.168.12.150:5060
To: sip:control@192.168.12.158:5564;tag=1
From: sip:conf=1234@192.168.12.150:5060;
   tag=1119359262
Call-ID: 1119359610
User-Agent: SnowShoreMediaServer/1.4.0-050614A
CSeq: 4 INFO
Content-Type: application/mediaservercontrol+xml
Supported: timer
Supported: 100rel
Content-Length: 137

<?xml version="1.0"?>
<MediaServerControl version="1.0">
   <response request="configure_leg" code="200"
   text="OK" />
</MediaServerControl>
```

IVR with MSCML

Playing a Simple Announcement

Play Payload

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <play id="332985001"
   prompt="file:////opt/snowshore/prompts/generic/
   10.ulaw"/>
  </request>
</MediaServerControl>
```

Expected Response

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="play" id="332985001"
   code="200" text="OK" reason="EOF"/>
</MediaServerControl>
```

Playing a Sequenced Announcement

Play Payload

Example 39. Playing Sequenced Announcement—Payload

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
<request>
<playcollect id=21212 maxdigits=5 firstdigittimer=250
   interdigittimer=100 extradigittimer=250>
  <prompt baseurl="file:////opt/snowshore/
   |prompts/generic/" locale="en_US">
   <audio url="num_changed.wav"/>
   <delay duration="10"/>
   <audio url="new_number.wav"/>
   <variable type="dig" subtype="ndn"
   value="9725551212"/>
   <audio url="make_note.wav"/>
  </prompt>
</playcollect>
</request>
</MediaServerControl>
```

Stopping a Play Command

Request 1 Payload

Example 40. Request 1 Payload

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
      <play id="332985011"
   prompt="file:////opt/snowshore/
   prompts/generic/
   10.ulaw"/>
  </request>
</MediaServerControl>
```

Request 2 Payload

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <stop id="332985012" />
  </request>
</MediaServerControl>
```

Expected Response to Request 1 Payload

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="play" id="332985011"
   code="200" playduration=”10000”  text="OK"
   reason="stopped"/>
</MediaServerControl>
```

Expected Response to Request 2 Payload

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
        <response request="stop" id="332985012"
   code="200" text="OK"/>
</MediaServerControl>
```

PlayCollect

Payload

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <playcollect id="332986004"
   prompt="file:////opt/snowshore/prompts/generic/
   10.ulaw" maxdigits="1" barge="yes"/>
  </request>
</MediaServerControl>
```

Expected Response

The play is barged into on the digit press. The operation matches the digit 2 and the INFO response indicates the match.

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
     <response request="playcollect" id="332986004"
   code="200" text="OK" reason="match"
   playduration=”10000” digits="2"/>
</MediaServerControl>
```

Playing a Recording

Payload

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <playrecord id="332987015"
   prompt="file:////opt/snowshore/prompts/generic/en_U
   S/10.ulaw"
   recurl="file:////var/snowshore/rec/temp01.ulaw"
   recencoding="ulaw"/>
  </request>
</MediaServerControl>
```

Expected Response

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <response request="playrecord" id="332987015"
   code="200" text="OK" reason="end_silence"
   reclength="1234" playduration=”10000” digits=""/>
</MediaServerControl>
```

Stopping a Recording

Request 1 Payload

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <request>
    <playrecord id="332987022"
   prompt="file:////opt/snowshore/prompts/generic/
   10.ulaw" initsilence="30000"
   recurl=file:////tmp/7001.ulaw"/>
  </request>
</MediaServerControl>
```

Request 2 Payload

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

Expected Response Request 1 Payload

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <response request="playrecord" id="332987022"
   code="200" text="OK" reason="stopped" reclength="0"
   playduration=”10000” digits=""/>
</MediaServerControl>
```

Expected Response Request 2 Payload

```
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <response request="stop" code="200" text="OK"/>
</MediaServerControl>
```

Asynchronous DTMF

Subscribing to Standard Digit Events

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

Subscribing to Long Digit Events

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

Subscribing to Both Standard and Long Digit Events

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

Turning Off Digit Event Reporting

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

Example Responses

The following shows the response generated by detection of a standard "4" DTMF digit.

Note: This is the first digit detected so 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>
```

The following shows the response 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>
```

Example 41 shows the responses that are generated by the detection of two standard pound (#) events which also meet the double sequence criteria for long digits. In this scenario, three responses are generated; two for each of the standard pound events and one indicating that these responses comprise a long digit.

Multiple responses are generated because the subscription contained <keypress report="both"/>, as shown in Example 41.

Example 41. Asynchronous DTMF Response

```
1. <?xml version="1.0"?>
   <MediaServerControl version="1.0">
      <notification>
         <keypress digit="#" length="standard"
            method="standard" interdigittime="0">
            <status command="idle" duration="5"/>
         </keypress>
      </notification>
   </MediaServerControl>

2. <?xml version="1.0"?>
<MediaServerControl version="1.0">
     <notification>
        <keypress digit="#" length="standard"
   method="standard"
       interdigittime="1000">
         <status command="idle" duration="6"/>
   </keypress>
     </notification>
   </MediaServerControl>
3. <?xml version="1.0"?>

<MediaServerControl version="1.0">
     <notification>
         <keypress digit="#" length="long"
   method="double"
       interdigittime="1000">
<status command="idle" duration="6"/>
       </keypress>
     </notification>
   </MediaServerControl>
```

If asynchronous DTMF is used in a conferencing scenario, make sure to maintain the subscription across multiple <configure_leg/> requests.

Call Flow for IVR with MSCML

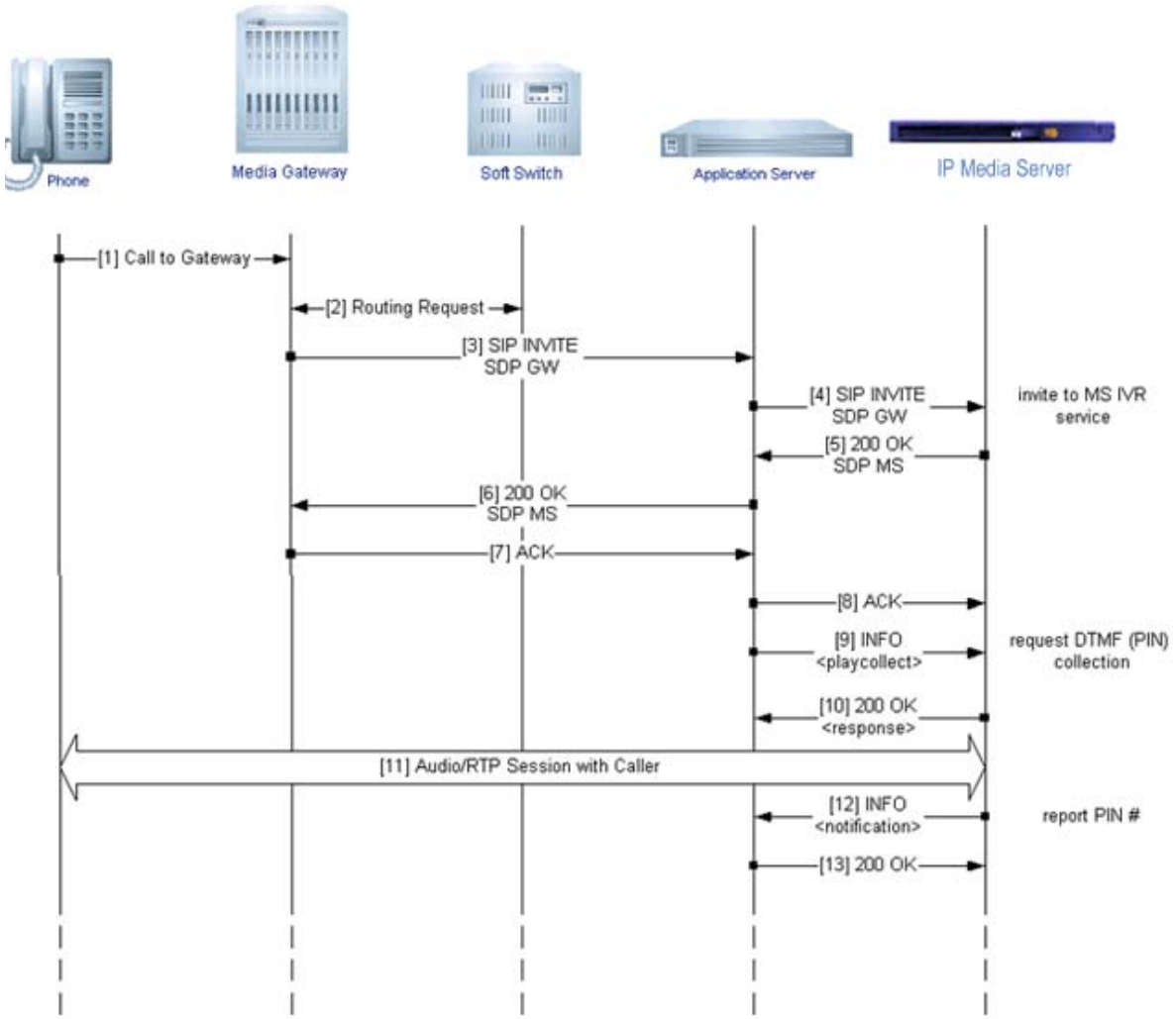

Figure 11. Call Flow: IVR with MSCML

Call Flow for PIN Collection, IVR with MCSCML

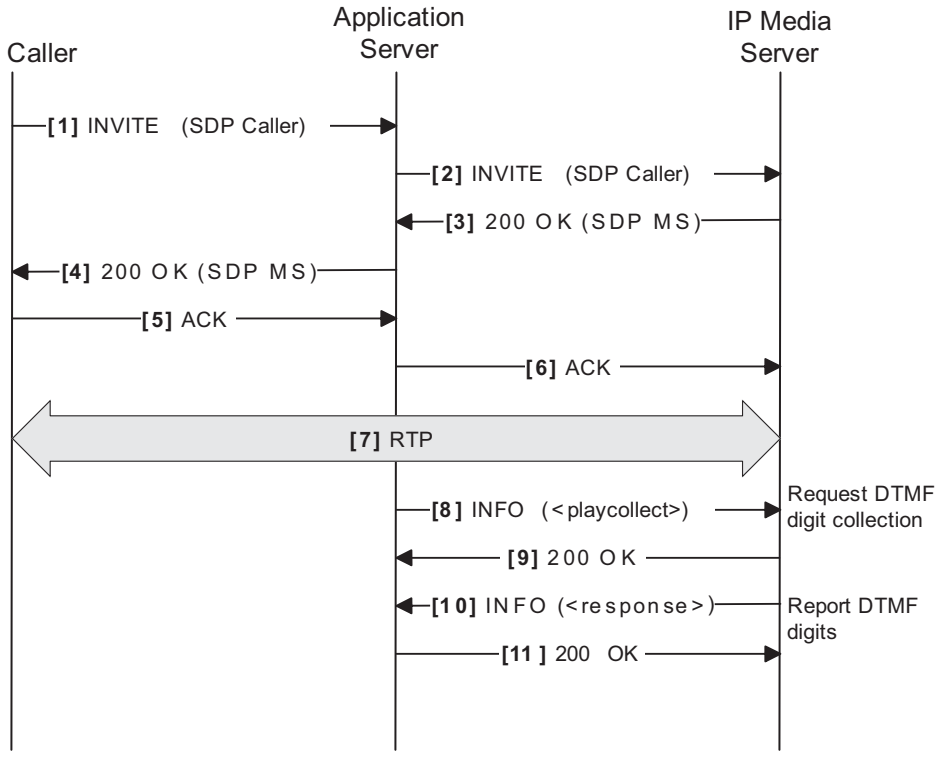

Figure 12. Call Flow: PIN Collection, IVR with MSCML

Explanation of Call Flow

The numbers in the following table correspond to those on the call flow diagram in Figure 12.

[1]	SIP INVITE from caller to application server Method: SIP INVITE Body: Caller's SDP
[2]	SIP INVITE from application server to IP Media Server Method: SIP INVITE Body: Caller's SDP
[3]	Response to SIP INVITE [2] Method: SIP INVITE Body: Dialogic® IP Media Server's SDP
[4]	Response to SIP INVITE [1] Method: SIP INVITE Body: Dialogic® IP Media Server's SDP
[5]	Acknowledgement of final response to INVITE [1] Method: SIP ACK Body: None
[6]	Acknowledgement of final response to INVITE [2] Method: SIP ACK Body: None

[7]	Bi-directional SDP established between caller and IP Media Server
[8]	Application server directs Dialogic® IP Media Server to play a prompt and collect returned digits Method: SIP INFO Body: <?xml version="1.0"?> <MediaServerControl version="1.0"> <request> <playcollect prompt="file:////opt/snowshore/prompts/conf/enterpin.wav" maxdigits="4" cleardigits="yes" returnkey="#" escapekey="*"> </playcollect> </request> </MediaServerControl
[9]	Response from Dialogic® IP Media Server to <playcollect> request Method: SIP INFO Body: Empty
[10]	Return collected DTMF digits Method: SIP INFO Body: <?xml version="1.0"?> <MediaServerControl version="1.0"> <response request="playcollect" code="200" text="OK" reason="match" digits="1234" playduration=”5230”/> </MediaServerControl>
[11]	Response from application server to INFO response Method: SIP INFO Body: Empty

Call Flow for Recording a Message, IVR with MSCML

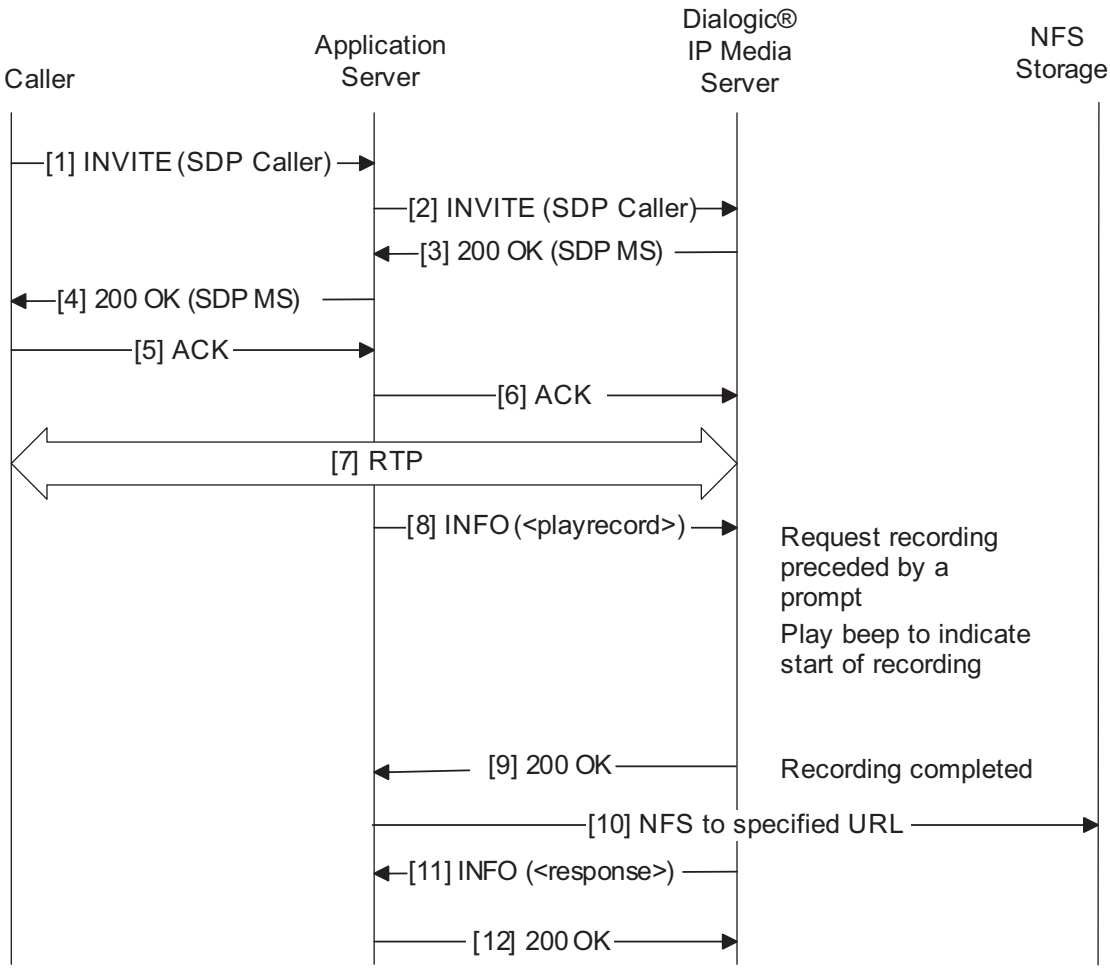

Figure 13. Call Flow: Recording, IVR with MSCML

Explanation of Call Flow

The numbers in the following table correspond to those on the call flow diagram in Figure 13.

[1]	SIP INVITE from caller to application server Method: SIP INVITE Body: Caller's SDP
[2]	SIP INVITE from application server to IP Media Server Method: SIP INVITE Body: Caller's SDP
[3]	Response to SIP INVITE [2] Method: SIP INVITE Body: Dialogic® IP Media Server's SDP
[4]	Response to SIP INVITE [1] Method: SIP INVITE Body: Dialogic® IP Media Server's SDP
[5]	Acknowledgement of final response to INVITE [1] Method: SIP ACK Body: None
[6]	Acknowledgement of final response to INVITE [2] Method: SIP ACK Body: None
[7]	Bi-directional SDP established between caller and IP Media Server
[8]	Application server directs IP Media Server to play a prompt and then record the caller's message. Method: SIP INFO Body: <?xml version="1.0"?> <MediaServerControl version="1.0"> request> <playrecord prompt="file:////opt/snowshore/prompts/conf/recmessage.wav" escapekey="*" recurl="file:////storage/msg123.ulaw" initsilence="1000" endsilence="4000" beep="yes" recstopmask="01234356789*#"/> </request> </MediaServerControl>
[9]	Response from Dialogic® IP Media Server to <playrecord> request Method: SIP INFO Body: Empty Response from Dialogic® IP Media Server to <playrecord> request

[10]	Recording is written to the external storage location specified in the url parameter of the <record> request.
[11]	Response from IP Media Server to application server when recording is complete Method: SIP INFO Body: <MediaServerControl version="1.0"> <response request="playrecord" code="200" text="OK" reason="endsilence" digits="" playduration=”4738” reclength=”24000”/> </MediaServerControl>
[12]	Response from application server to INFO response Method: SIP INFO Body: Empty

VoiceXML

Playing an Announcement

The code sample below is for a simple announcement, such as "The number you have reached...". This could be a dynamically generated script, which would fill in the values for the variables "was" and "is".

Example 42. Playing an Announcement, VoiceXML

```
<?xml version="1.0" ?>
<vxml version="1.0" application="default.xml">
<form id="changed">
<block>
<var name="was" expr="9785551212"/>
<var name="is" expr="9785554141"/>
<prompt>
   <audio src="audio/numreached.ulaw"/>
   <value class="phone" mode="recorded" expr="was"/>
   <break/>
   <audio src="audio/beenchanged.ulaw"/>
   <audio src="audio/newnumberis.ulaw"/>

<value class="phone" mode="recorded" expr="is"/>

<audio src="audio/pleasenote.ulaw"/>
</prompt>
</block>
</form>
</vxml>
```

PIN Collection

This script collects a PIN number, and submits it to a web server for verification.

Example 43. PIN Collection

```
<?xml version="1.0" ?>

<vxml version="1.0" application="default.xml">

<form id="get_pin">

<field name="pin" type="digits?length=4">
   <prompt count="1">
```

```
<audio src="audio/enterpin.ulaw"> Please enter your PIN
   number.</audio>
   </prompt>
   <prompt count="2">
      <audio src="audio/pin.ulaw"> PIN? </audio>

</prompt>
   <noinput count="1"> <reprompt/> </noinput>
   <noinput count="2"> <exit/> </noinput>
</field>

</field>
</form>
</vxml>
<field name="confirm" type="boolean">
   <prompt>
      <audio src="audio/yourpinis.ulaw">
   Your pin is </audio>
      <value expr="pin"/>
      <audio src="audio/correct.ulaw">
         If this correct press 1.  Otherwise, press 2
      </audio>
   </prompt>
   <nomatch count="1"> <reprompt/> </nomatch>
<nomatch count="3"> <exit/> </nomatch>
   <noinput count="1"> <reprompt/> </noinput>
   <noinput count="3"> <exit/> </noinput>
   <filled>
      <if cond="confirm">
         <submit namelist="pin" next="pin.cgi"/>
   <else/>
         <clear namelist="pin confirm"/>
      </if>
   </filled>
```

Call Flow for VoiceXML

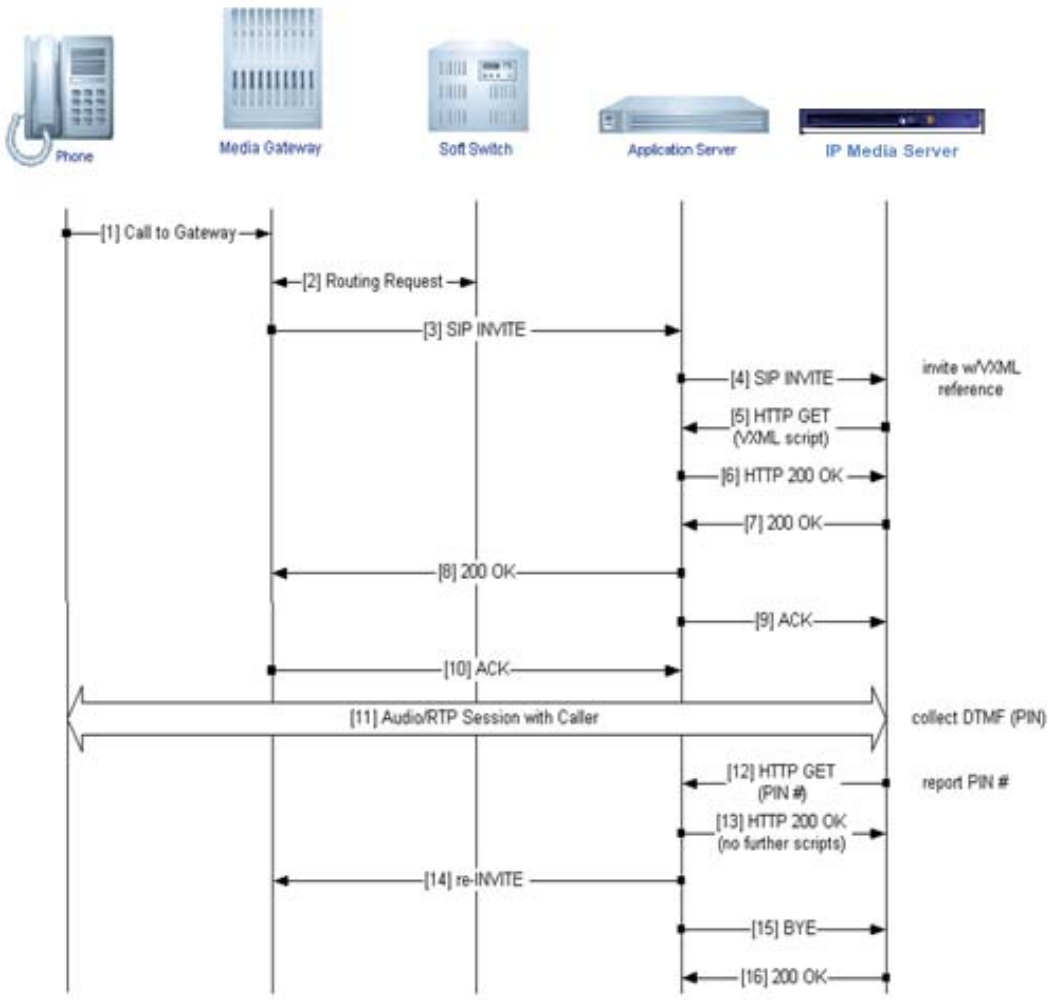

Figure 14. Call Flow: VoiceXML

Transferring a Call

This script depicts a bridged transfer, which returns the caller to the original session with the interpreter when completed. One MSCML session (in addition to a VoiceXML resource) is required for each simultaneous transfer session. MSCML and VXML session configuration is performed at the CLI logical-vhost level.

Example 44. Transferring a Call, VoiceXML

```
<?xml version="1.0"?>

<!-- 10.10.10.111 is the SnowShore MPS         -->
<!-- 12345679@10.10.10.253 is a valid SIP destination    -->
<!-- This script would first play “star”, followed by “One” -->
<!-- Then attempts to do the transfer. If the transfer is  -->
<!-- success, when the call is ended, you’ll hear the reason-->
<!-- number (see below in the script) followed by the number-->
<!-- seconds the call is connected betweeen the caller and -->
<!-- the callee.                        -->
<vxml version=”1.0”>
<form name=”genericTransfer”>
<block>
<audio src=”http://10.10.10.111/snowshore/prompts/gmvoices/star.neu” />
   </block>


<transfer name=”makethecall” dest=”sip:12345679@10.10.10.253” connecttimeout=”30s”
  bridge=”true”>
      <prompt>
      <audio src=”http://10.10.10.111/snowshore/prompts/gmvoices/1.ris” />
      </prompt>
      <filled>
      <if cond=”makethecall == ‘busy’”>
      <prompt>

<audiosrc=”http://10.10.10.111/snowshore/prompts/gmvoices/2.ris” />
      </prompt>
      <elseif cond=”makethecall == ‘noanswer’”/>
      <prompt>

<audio src=”http://10.10.10.111/snowshore/prompts/gmvoices/3.ris” />
      </prompt>
      <elseif cond=”makethecall == ‘near_end_disconnect’”/>
      <prompt>
      <audio src=”http://10.10.10.111/snowshore/prompts/gmvoices/4.ris” />
      </prompt>
      <elseif cond=”makethecall == ‘far_end_disconnect’”/>
      <prompt>


<audio src=”http://10.10.10.111/snowshore/prompts/gmvoices/5.ris” />
      </prompt>
      <elseif cond=”makethecall == ‘network_disconnect’”/>
<prompt>
```

```
      <audio src=”http://10.10.10.111/snowshore/prompts/gmvoices/6.ris” />
      </prompt>

<elseif cond=”makethecall == ‘network_busy’”/>
      <prompt>
<audio src=”http://10.10.10.111/snowshore/prompts/gmvoices/7.ris” />
      </prompt>
      <elseif />
      <prompt>
      <value class=”number” expr=”makethecall$.duration” />
      <audio src=”http://10.10.10.111/snowshore/prompts/gmvoices/seconds.neu” />
      </prompt>
      </if>
      </filled>
      </transfer>
      <block>
      <audio src=”http://10.10.10.111/snowshore/prompts/gmvoices/star.neu” />
</block>
</form>
</vxml>
```

VoiceXML Script—Using VCR Controls

Example 45. Using VCR Controls

```
<link event="vcr">
        <dtmf>
               7 {vcr_rew} | 8 {vcr_pause} | 9 {vcr_fwd}
        </dtmf>
</link>

<form id="form1">
<!- Remove the default termchar, as we don't want to wait for it -->
      <property name="termchar" value=""/>
        <property name="com.snowshore.vcr_skip" value="1s"/>   <!-- default 5s -->
      <property name="com.snowshore.vcr_pause" value="10s"/> <!-- default 10s -->
<!- VCR controls work on this prompt only ->
<block>
       <prompt vcr="true">
<audio
src="file:////net/192.168.12.12/opt/snowshore/prompts/gmvoices/
  en_US/susan/msg055"/>

<!-- The message audio -->audio
src="file:////net/192.168.12.12/opt/snowshore/prompts/generic/
  1MinuteMusic.raw"/>
</prompt></prompt>
</block>
<!-- This is the menu "To repeat, press 1, to save, press 2, to erase ... -->
<!-- VCR controls don't affect this prompt, but ARE active, i.e. 7 can    -->
<!-- backup out of this menu back into the message      -->
<field name="field1" type="digits?length=1">
       <prompt>
audio
```

```
src="file:////net/192.168.12.12/opt/snowshore/prompts/generic/
  en_US/try_again.ulaw"/>
         </prompt>
filled>

       <!-- Fetch next thing to do -->
<submit
next="msgfetch?sid=12345&amp;msgid=42&amp;digit=$field1"/>
         </filled>

<catch event="vcr">
            <!- VCR events must loop back to the top in order to be effective->
            <clear/>
            <reprompt/>
            </catch>
</field>
</form>
```

A - Audio Library

This appendix describes a preconfigured sound library (audio files) which consists of phrases and messages, numbers, time and money, quantities, and miscellaneous words.

This appendix covers the following major sections:

Sound Library

The Dialogic® IP Media Server comes with a library of stock audio segments, professionally recorded in a female voice. These files are stored in the following location:

```
/opt/snowshore/prompts/
```

Within this directory are two subdirectories:

- `/generic/` contains some files recorded by Dialogic that can be useful for testing.
- `/gmvoices/`) holds a directory of audio files in US English (`/en_US/susan/`) and a directory of audio files in Chinese Mandarin (`/zh_CN/sara/`).

These audio segments provide a consistent set of sounds that have been recorded to support a wide variety of pre-programmed phrases, spoken numbers, dates, and other sequenced recordings.

These files are in a protected area and cannot be modified. If you have application-specific audio files, Consider placing these files an NFS server on your network rather than locally on the IP Media Server.

With the exception of the generic audio files and audio files for letters of the alphabet, the other files listed in the following pages exist in both US English and Chinese Mandarin versions.

The tables on the following pages explain the categories of phrases provided with the prompt library. Phrases are categorized as follows:

Phrases and Messages

Phrases and message are located in:

```
opt/snowshore/prompts/gmvoices/en_US/susan/
```

and in

```
/opt/snowshore/prompts/gmvoices/zh_CN/sara/
```

These files are standard phrase segments that can be combined into sequences.

Table 44. Prompt Library: Standard Phrases

Filename	Spoken as
msg001	one second of silence
msg002	600Hz beep tone
msg003	Goodbye.
msg004	Hello.
msg005	Good morning.
msg006	Good afternoon.
msg007	Good evening.
msg008	If this is correct...
msg009	To transfer to another extension...
msg010	To return to the main menu...
msg011	To return to the previous menu...
msg012	To delete this message...
msg013	To delete all messages...
msg014	To exit the system...
msg015	To end this call...
msg016	To cancel delivery of this message...
msg017	To send your message now...
msg018	Please hold while your call is being transferred.
msg019	Please leave your message after the tone.
msg020	Please begin recording after the tone.
msg021	Thank you for calling.
msg022	Thank you for calling, goodbye.
msg023	Thank you.
msg024	Please enter your mailbox number.
msg025	Please enter your passcode.
msg026	I'm sorry, that passcode is invalid. Please reenter your passcode.
msg027	I'm sorry, that mailbox number is invalid. Please reenter your mailbox number.

Table 44. Prompt Library: Standard Phrases (continued)

Filename	Spoken as
msg028	I'm sorry, that is an invalid entry. Please try again.
msg029	You have no messages.
msg030	You have no more messages.
msg031	You have...
msg032	...new message.
msg033	...new messages.
msg034	...saved message.
msg035	...saved messages.
msg036	Your mailbox is currently full.
msg037	Message deleted.
msg038	Message saved.
msg039	Please hold.
msg040	Please hold for assistance.
msg041	I'm sorry...
msg042	You entered...
msg043	End of messages.
msg044	We cannot identify that entry. Please try again.
msg045	Please hold while I transfer your call to the operator.
msg046	We're sorry you are having difficulty. Please try your call again later.
msg047	If this is correct, press 1. If not, press 2.
msg048	Please enter your passcode now.
msg049	If you're calling from a touchtone phone, press 1 now. Otherwise, please stay on the line to speak with an operator.
msg050	Please enter your business phone number beginning with the area code now.
msg051	You have selected...
msg052	You have reached...
msg053	To make another selection...
msg054	To try again...

Table 44. Prompt Library: Standard Phrases (continued)

Filename	Spoken as
msg055	Extension...
msg056	The telephone number you entered is...
msg057	The fax number you entered is...
msg058	Please enter the account number.
msg059	Please enter your personal identification number.
msg060	Your personal identification number is...
msg061	Thank you, please hold.
msg062	Please record your message at the tone. When you are finished you may hang up or...
msg063	...for more options.
msg064	Please record your greeting at the tone.
msg065	To save this message, press...
msg066	To delete this message, press...
msg067	To listen to the next message, press...
msg068	To replay this message, press...
msg069	The passcode you entered is...
msg070	Your message to...
msg071	...was delivered.
msg072	...could not be delivered.
msg073	I'm sorry, we cannot connect your call at this time. Please try again later.
msg074	Mailbox number...
msg075	...is full.

Numbers

These files are spoken as cardinal numbers. (Ordinals are listed with dates.)

Some numbers have multiple files for neutral, rising, and falling inflection, as indicated by their file extensions.

All these files are located in:

```
opt/snowshore/prompts/gmvoices/en_US/susan/
```

and in

```
/opt/snowshore/prompts/gmvoices/zh_CN/sara/
```

Table 45. Prompt Library: Cardinal Numbers

Filename	Numbers	Extension	Spoken As
0.neu ... through 9.neu 0.ris ... through 9.ris 0.dwn ...through 9.dwn	0 through 9	.neu .ris .dwn	Zero through Nine with neutral, rising, or falling intonation
10.num, 11.num, 12.num... through 100.num	10 through 100	.num	Ten, Eleven, Twelve,... One hundred'
100.neu ... through 1000.neu 100.ris ... through 1000.ris 100.dwn ...through 1000.dwn	100 through 1000 in increments of 100	.neu .ris .dwn	One hundred through One thousand with neutral, rising, or falling intonation.
thousand.num million.num billion.num trillion.num	thousand, million, billion, and trillion	.num	Thousand, Million, Billion, Trillion

Dates and Ordinal Numbers

The following files consists of dates and ordinal numbers.

Table 46. Prompt Library: Dates and Ordinal Numbers

Filename	Days/Months	Spoken As
mon.dat tue.dat wed.dat thu.dat fri.dat sat.dat sun.dat	Monday through Sunday	Day of the week, Monday through Sunday

Table 46. Prompt Library: Dates and Ordinal Numbers (continued)

Filename	Days/Months	Spoken As
1st.dat, 2nd.dat, 3rd.dat... through 31st.dat	1st, 2nd, 3rd... through 31st	Spoken as ordinal number First, Second, Fourth, Thirty-first
jan.dat feb.dat mar.dat apr.dat may.dat jun.dat jul.dat aug.dat sep.dat oct.dat nov.dat dec.dat	January through December	Name of month January, February, etc.
Filename	**Years**	**Spoken As**
1975.dat ... through 2015.dat	1975 through 2015	Spoken as year. Nineteen seventy five, ‘Two thousand and fifteen

Letters

There are two audio files for each letter of the alphabet: one with rising intonation, for example aup.ltr and one with falling intonation, for example adn.ltr.

Having both types of files allows you to select audio files by context. For example, the letters C and N have different sounds in the acronyms for NBC and CNN.

These audio files are located in:

```
opt/snowshore/prompts/gmvoices/en_US/susan/
```

and in

```
/opt/snowshore/prompts/gmvoices/zh_CN/sara/
```

Filename	Spoken With	Example
aup through zup.ltr	Rising intonation	aup.ltr; bup.ltr ... through zup.ltr
adn through zdn.ltr	Falling intonation	adn.ltr; bdn.ltr ... through zdn.ltr

Time and Money

You can combine these audio files with audio files for numbers to form sequences for times of day and money values. These audio files are located in:

```
opt/snowshore/prompts/gmvoices/en_US/susan/
```

and in

```
/opt/snowshore/prompts/gmvoices/zh_CN/sara/
```

Table 47. Prompt Library: Time and Money Phrases

Filename	Spoken as Time Zones:
est.tim	Eastern Standard Time
cst.tim	Central Standard Time
mst.tim	Mountain Standard Time
pst.tim	Pacific Standard Time
am.tim	AM
at.tim	at (time)
mid.tim	Midnight
noon.tim	Noon
oclock.tim	o’clock
pm.tim	PM
and.mon	and (money)
cent.mon	Cent
cents.mon	Cents
dollar.mon	Dollar
dollars.mon	Dollars

Press Keys

Most of these files have two values: one with falling intonation (*.dwn) and one with neutral intonation (*.neu).

Press key audio files are located in:

```
opt/snowshore/prompts/gmvoices/en_US/susan/
```

and in

```
/opt/snowshore/prompts/gmvoices/zh_CN/sara/
```

Table 48. Prompt Library: Press Key Phrases

Filename	Spoken as Press Key Phrases:
press	Press
prs0.dwn / prs0.neu	Press 0
prs1.dwn / prs1.neu	Press 1
prs2.dwn / prs2.neu	Press 2
prs3.dwn / prs3.neu	Press 3
prs4.dwn / prs4.neu	Press 4
prs5.dwn / prs5.neu	Press 5
prs6.dwn / prs6.neu	Press 6
prs7.dwn / prs7.neu	Press 7
prs8.dwn / prs8.neu	Press 8
prs9.dwn / prs9.neu	Press 9
asterisk.dn /asterisk.neu	Asterisk
pound.dwn / pound.neu	Pound
star.dwn / star.neu	Star

Quantities

Quantity audio files are located in:

```
opt/snowshore/prompts/gmvoices/en_US/susan/
```

and in

```
/opt/snowshore/prompts/gmvoices/zh_CN/sara/
```

Table 49. Prompt Library: Quantities

Filename	Spoken as	
year.neu	year	neutral
years.neu	years	neutral
month.neu	month	neutral
months.neu	months	neutral
day.neu	day	neutral
days.neu	days	neutral
hour.neu	hour	neutral
hours.neu	hours	neutral
minute.neu	minute	neutral
minutes.neu	minutes	neutral
second.neu	second	neutral
seconds.neu	seconds	neutral
dollar.neu	dollar	neutral
dollars.neu	dollars	neutral
cent.neu	cent	neutral
cents.neu	cents	neutral
minus.dwn	minus	neutral
year.dwn	year	final intonation
years.dwn	years	final intonation
month.dwn	month	final intonation
months.dwn	months	final intonation
day.dwn	day	final intonation
days.dwn	days	final intonation

Table 49. Prompt Library: Quantities (continued)

Filename	Spoken as	
hour.dwn	hour	final intonation
hours.dwn	hours	final intonation
minute.dwn	minute	final intonation
minutes.dwn	minutes	final intonation
second.dwn	second	final intonation
seconds.dwn	seconds	final intonation
dollar.dwn	dollar	final intonation
dollars.dwn	dollars	final intonation
cent.dwn	cent	final intonation
cents.dwn	cents	final intonation
minus.dwn	minus	final intonation

Miscellaneous Words

Miscellaneous words are located in:

```
opt/snowshore/prompts/gmvoices/en_US/susan/
```

and in

```
/opt/snowshore/prompts/gmvo1ices/zh_CN/sara/
```

Note: Many of these files do not have filename extensions.

Table 50. Prompt Library: Miscellaneous Words

Filename	Spoken as
act	Activate
acted	Activated
dash	Dash
deact	Deactivate
deacted	Deactivated
for	For
no	No

Table 50. Prompt Library: Miscellaneous Words (continued)

Filename	Spoken as
none	None
o.dwn o.neu o.ris	O
off.dwn off.neu	Off
on.dwn on.neu	On
or	Or
percent	Percent
point	Point
sent	Sent

Generic Audio Files

Generic audio files are located in:

```
/opt/snowshore/prompts/generic/en_US
```

These files were internally recorded by Dialogic and can be useful for testing purposes.

Table 51. Generic Prompt Phrases

Filename	Spoken as
ac_changed.ulaw ac_changed.wav	“The area code for the number you are dialing has changed to...”
circuit_busy.ulaw circuit_busy.wav	“We’re sorry. All circuits are busy now.”
contact_provider.ulaw contact_provider.wav	“Please contact your service provider.”
dial_again.ulaw dial_again.wav	“If you’d like to make a call, please hang up and dial again.”
dial_operator.ulaw dial_operator.wav	“If you need help, please hang up and dial your operator.”
disconnected.ulaw disconnected.wav	“...has been disconnected.”
make_note.ulaw make_note.wav	“Please make a note of it.”
new_number.ulaw new_number.wav	“The new number is...”
no_permission.ulaw no_permission.wav	“You no longer have permission to utilize the system.”
num_changed.ulaw num_changed.wav	“The number you are calling has changed.”
num_dialed.ulaw num_dialed.wav	“The number you have dialed...”
num_invalid.ulaw num_invalid.wav	“...is not a valid number.”

Table 51. Generic Prompt Phrases (continued)

Filename	Spoken as (continued)
please_check.ulaw please_check.wav	“Please check the number and dial again.”
service_outage.ulaw service_outage.wav	“Please contact your service provider for information relating to this service outage.”
try_again.ulaw try_again.wav	“Please hang up and try your call again later.”

B - VoiceXML Version 1.0 and Dialog Service

This appendix explains the basics of VoiceXML (VXML) Version 1.0 and lists the supported VoiceXML elements and attributes, and explains ECMAScript language functionality.

Note: For information about VoiceXML 2.0, see the *VoiceXML 2.0 Reference*.

The dialog service is one of two interfaces that the IP Media Server offers for developing IVR/DTMF applications. The other service is ivr through which SIP requests are enhanced by MSCML message bodies for play, playcollect, and playrecord. For details on IVR, see Chapter 5, "IVR with MSCML".

This appendix includes the following sections:

- VoiceXML Properties
- ECMAScript Functionality
- Support for VoiceXML Extended Session Variables

About VoiceXML

VoiceXML is a W3C standard scripting language for playing audio prompts and for collecting DTMF input.

The interpreter executes VoiceXML dialogs on an RTP stream (see next section). Each dialog represents an announcement, menu, or other IVR script. The dialogs finish when they have posted information to a Web server or returned a namelist back to the command that invoked the browser. If additional dialogs are needed, then the application runs another script.

While the media server supports full ECMA (European Computer Manufacturer's Association) Scripting abilities, Dialogic recommends that you perform that type of logic on the application server rather than in the VoiceXML script. This method reduces the processing and memory required by the Dialogic® IP Media Server and allows the Dialogic® IP Media Server to handle a large volume of simultaneous sessions.

VoiceXML Interpreter

The IP Media Server includes interpreters for VoiceXML 1.0 and VoiceXML 2.0.

Note: For information about VoiceXML 2.0, see the *VoiceXML 2.0 Reference*.

If the IP Media Server receives a SIP INVITE request directed to the dialog service, a VoiceXML session begins. The VoiceXML session initially gets (fetches) and executes the VoiceXML script that is specified in the SIP Request-URI parameter, voicexml.

Once specified, a VoiceXML script URI remains in effect until it completes execution or the session is stopped.

Dialog Service Indicator and Request URI

Through the dialog service (`dialog`), the Dialogic® IP Media Server executes VoiceXML documents to offer IVR scripting with DTMF input and recorded audio output.

The application references the initial VoiceXML script using the Request-URI parameter voicexml. The following SIP Request-URI directs the IP Media Server to retrieve and execute script1.vxml from the server app1.carrier.com.

```
INVITE sip:dialog@MS_IP;voicexml=http://
   app1.carrier.com/path/
   script1.vxml
```

It is common for the HTTP URI referencing the VoiceXML script to be a query that contains its own parameters so the script can be dynamically generated. For example, the following HTTP URI provides a subscriber ID so the VoiceXML script can be appropriately personalized.

```
INVITE sip:dialog@MS_IP;voicexml=http://
   app1.carrier.com/cgi/bin/
   genvxml.pl?subscriberid=34590087
```

If a query HTTP URI is used, be careful to replace characters that are reserved in SIP with their hexadecimal equivalents preceded by a percent (%) character. This substitution is called escaping. In particular, the question mark (?) and equals sign (=) must be escaped to conform with SIP standards. The following example shows the escaped form of the previous example.

```
INVITE sip:dialog@MS_IP;voicexml=http://
   app1.carrier.com/cgi/bin/
   genvxml.pl%3Fsubscriberid%3D34590087
```

The SIP URI must be escaped, as described in "Syntax and Escaping" (page 80).

VoiceXML Launcher

If a SIP INVITE to the Dialogic® IP Media Server is directed to the dialog service but does not specify a file (voicexml=), then the Dialogic® IP Media Server launches the dialog service and runs a default script configured by the system administrator using the Web UI MEDIA SERVER > VOICEXML menu.

VoiceXML Concepts

Syntax

VoiceXML (VXML) is an eXtensible Markup Language for the creation of IVR and Automated Speech Recognition (ASR) applications. Based on XML tag/attribute format, its syntax involves enclosing instructions (items) within a tag structure. For example:

```
<element_nameattribute_name="attribute_value">
…contained items….
</element_name>
```

Generic XML concepts remain unchanged in VoiceXML. For example, any character data must be escaped as per the XML specification.

Scope

As for all XML, VoiceXML observes a hierarchical structure and applies the construct of scope to define the range within the source where a variable, event handler, or other element is applicable.

Resource Fetching

VoiceXML defines several attributes to qualify properties relevant to the caching and fetching of documents and other resources.

Note: The Dialogic® IP Media Server does not currently support these attributes.

VoiceXML Application and Its Documents

A VoiceXML application is a set of VoiceXML scripts that share the same root document. The application consists of one or more text files called documents. Document files are identified by a .vxml extension and are retrieved using HTTP or (Network File System) NFS.

If a VoiceXML application includes multiple documents, one of these can be the application root document. The application shares the root document among all other documents. Sharing root documents occurs when the script author explicitly sets the root document to be the same in multiple VoiceXML scripts. The IP Media Server's VoiceXML browser supports the VoiceXML standard.

Whenever a user interacts with a document in an application, the corresponding root document is also loaded. The root document is unloaded when a user transitions to a document not in the application.

While the root document is loaded, the application root document's variables are available to other documents as application variables. Grammars defined in the root document also remain active for the duration of the application.

Figure 15 shows how a root document is shared in a VoiceXML application.

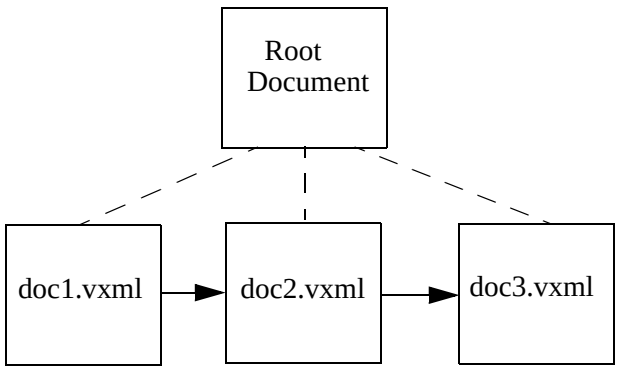

Figure 15. Sharing of Root Document in VoiceXML Application

Dialogs

A VoiceXML document contains dialogs or conversations between a prompting VoiceXML interpreter and a responding caller.

Conversational building blocks are as follows:

- Prompts (in the Dialogic® IP Media Server context for playing audio)
- The grammar that specifies the range of acceptable user responses to a prompt
- Transitions (to the URI of the next prompt/response exchange)

Conversational constructs are as follows:

- Forms that collect an input or event, process it, and select the next form to visit.
- Menus for organization and navigation (multiple-choice options, with a transition for each).
- Links that execute a transaction or throw an event.

Grammar and Scripting

The Dialogic® IP Media Server supports DTMF grammars and also supports ECMA tags that involve the scripting capabilities of the VoiceXML interpreter. See "ECMAScript Functionality" (page 289).

However, executing complex ECMAScript on the IP Media Server can affect its ability to perform real-time media processing. To conserve processing capacity and memory, Dialogic recommends limiting the use of ECMAscripting on the IP Media Server.

Session Variables

Voice XML provides a facility, called session variables, to pass signaling information to Voice XML scripts. The IP Media Server supports the standard Voice XML session variables as well as extensions which capture additional information from the SIP INVITE. See "Support for VoiceXML Extended Session Variables" (page 292) for further details.

File Storage and Retrieval

You can access VoiceXML scripts through the network using NFS and HTTP protocols. Audio and video content can be retrieved and stored using HTTP and NFS as well.

Note: The VoiceXML files can be located anywhere as long as the VoiceXML interpreter can access them.

Media Content Recovery Extension

The Media Content Recovery mechanism has been extended such that there can be increased reliability of recorded content delivery from Voice XML applications.

"Next-generation" application architectures such as SIP and VoiceXML are highly distributed in nature and rely on multiple components and network communication protocols to function. In the case of VoiceXML applications, the link between the VoiceXML browser (the IP Media Server) and the application server, which provides the scripts and content, is critical.

Now that SIP and VoiceXML applications have been successfully deployed, carriers expect reliability and behavior similar to the legacy applications that are being replaced. In most situations, it is acceptable for a subscriber to call back and establish a new VoiceXML session if a failure occurs. The notable exception is when the caller has successfully recorded a message and expects for it to be sent. If the failure occurs before the message can be transmitted, there is no way to inform the user of the issue and there is nothing the user can do about it. The Media Content Recovery extension provides a solution to the specific issue of reliable content delivery.

Recording is implemented in VoiceXML through the <record> element. When a <record> element is processed, the IP Media Server creates a temporary file with a locally unique identifier name.

The IP Media Server VoiceXML 1.0 browser maintains a list of the temporary files created during each call and deletes them when the VoiceXML script terminates. The browser also deletes any temporary files upon startup. When the Media Content Recovery mechanism is used, temporary recordings that are left over after a failure will be processed. At startup, a recovery daemon detects temporary files that have been tagged for recovery.

The Media Content Recovery extension utilizes the <data> element to enable the application to associate recovery data with specific recorded content. The browser supports recovery data in the form of a completely specified HTTP URI. It is the VoiceXML application's responsibility to make sure the URI is correct and contains the information needed to deliver the content to the ultimate recipient.

If the Media Content Recovery extensions are not used, the IP Media Server processes the recording in accordance with standard VoiceXML. If the VoiceXML Media Content Recovery extensions are present in the VoiceXML script, then the recovery feature is used. A VoiceXML script that provides the user with a confirmation probably should not include the VoiceXML extensions due to the possibility that the recording could be recovered and sent without the user providing confirmation.

VoiceXML Elements Reference

This section describes the VoiceXML elements (tags) supported by the Dialogic® IP Media Server. For a more detailed description of VoiceXML attributes, see "VoiceXML Attributes Reference" (page 276).

<assign>

Assigns a value to a variable.

The value of this tag must be empty.

Parents: <block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>

Attributes:

Attribute	Description
name	Name of the variable being assigned to a value. **Required.**
expr	ECMAScript expression that, when evaluated, is assigned as the new value of the variable. It must be a valid ECMA value. **Required.**

<audio>

Plays pre-recorded audio and accesses the audio from the URI, which is specified by the src attribute.

The audio encoding is determined from the file header when .wav or .au formats are used. Otherwise, the audio encoding is implied by the file extension according to the following table.

Extension	Encoding
ulaw	G.711µ-law
alaw	G.711 A law
msgsm	Microsoft® GSM

If the encoding cannot be determined by the extension, it is assumed to be G.711 µ-law.

The value of this tag can be empty or can contain one or more of the child elements listed below.

Parents: <audio>, <block>, <catch>, <choice>, <error>, <field>, <filled>, <help>, <if>, <initial>, <menu>, <noinput>, <nomatch>, <prompt>, <record>, <subdialog>

Child Elements: <audio> <break> <sayas> <value>

Attributes:

Attribute	Description
src	String literal containing the URI for the audio. **Required.**

<block>

Defines a container for non-interactive executable content such as welcome prompts.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parent: <form>

Child Elements: <assign>, <audio>, <clear>, <disconnect>, <exit>, <goto>, <if>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>

Attributes:

Attribute	Description
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.
expr	Initial value of the form-item variable; the default is ECMAScript undefined. If initialized to a value, then the form item is not visited unless the form-item variable is cleared.
name	Name of the form-item variable used to track whether this block can be executed. Defaults to an inaccessible internal variable.

<break>

Adds a pause to the audio content.

The value of this tag must be empty.

Parents: <audio>, <choice>, <prompt>

Attributes:

Attribute	Description
size	A relative pause duration, with possible values of none (0 ms), small (50 ms), medium (200 ms) or large (500 ms). The default is medium.
msecs	Text. The number of milliseconds to pause.

<catch>

Defines an event handler within the current scope.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parents: <field>, <form>, <initial>, <menu>, <record>, <subdialog>, <vxml>

Child Elements: <audio> <value>

Attributes:

Attribute	Description
event	Event or events to catch. **Required.**
count	The occurrence of the event. (Default: 1.) The count allows you to handle different occurrences of the same event differently. Each form item and <menu> maintains a counter for each event that occurs while it is being visited; these counters are reset each time the <menu> or form item's <form> is re-entered.
cond	Boolean value of TRUE or FALSE. (Default: TRUE). When FALSE, the tag is ignored or skipped; the form is not visited.

<choice>

Defines a menu item by specifying one choice in the menu. It can specify a DTMF grammar fragment or the URI to go to when the choice is selected. When the choice is made, the tag either transitions to a new dialog or throws an event.

The Dialogic® IP Media Server handles DTMF grammar fragments only. The contents must be <audio> and not text.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parent: <menu>

Child Elements: <audio>, <sayas>, <value>

Attributes:

Attribute	Description
dtmf	DTMF sequence for the choice.
event	An event to be thrown instead of going to ext.
expr	Expression to evaluate instead of going to next.
next	URI of next dialog or document.

To indicate the action to take, set one of the following: next, expr, or event.

<clear>

Resets one or more form items to its initial state, including setting the form item variable to ECMAScript undefined and re-initializing the prompt counter and the event counters for the form item.

The value of this tag must be empty.

Parents: <block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>

Attributes:

Attribute	Description
namelist	List of items to clear. (When not specified, the application clears all form items in the current form.) **Required.**

<data>

Enables Media Content Recovery Extension. The Media Content Recovery feature is triggered by a <data> element with either of the following set:

- The src property set to builtin:persist.
  or
- The srcexpr property evaluates to the ECMA string builtin:persist.

An error.semantic event is generated if the <data> element is encountered with src or srcexpr set to anything but builtin:persist.

An error.semantic is generated if the <data> tag has any of the following attributes set with the builtin:persist data submission:

- method, enctype, fetchaudio, fetchint, fetchtimeout, maxage, maxstale

A valid <data> tag causes the recovery file to be created. If the recovery file cannot be written, an error.semantic event is generated.

The <data> tag is controlled entirely by its namelist attribute.

The namelist attribute must have one and only one variable name that is the same as the <record> element to be protected by the Media Content Recovery feature. The <record> input item must be in the same <form> element. If it is not, an error.semantic event is generated.

The namelist attribute must have one and only one recovery_uri variable name that evaluates to a defined ECMA value that is not an array or an object. Without the recovery_uri variable name the recovery file will not be written. The recovery daemon attempts content recovery by executing an HTTP POST method to the HTTP URI specified in the recovery information. When a file is being recovered, the recovery daemon uses the recording_name variable as the name of the content in the HTTP POST.

The recovery file is deleted if a new <data> element is encountered that does not have the recovery_uri name in the namelist. This allows an application to immediately disable the recovery feature if the user confirms that he does not want the message sent.

Note: Other uses of <data> as part of VoiceXML 2.1 are not supported.

Parents: <block>, <if>, <filled>, <form>, <noinput>

The <data> tag must be a descendant of a <form> tag that contains the associated <record> tag.

Child Elements: None.

Attributes:

Attribute	Description
namelist	See above.

Note: The Media Content Recovery extension currently supports audio content only.

<disconnect>

Causes the interpreter context to disconnect from the user. As a result, the interpreter context generates an event (telephone.disconnected.hangup). Upon receiving the event, the application then performs a cleanup.

<disconnect> is the only supported VoiceXML call control tag (the same effect as <exit>).

The value of this tag must be empty.

Parents: <block>, <catch>, <error>, <filled>, <help>, <noinput>, <nomatch>

Attributes: None

<dtmf>

Specifies a touch-tone grammar that defines both a set of key presses so a user can perform actions or supply information and the corresponding string values that describe those actions or information.

The value of this tag must be parsed text.

Parents: <field>, <form>, <link>

Attributes:

Attribute	Description
src	URI for the source for an external grammar.

Attribute	Description
scope	Options are document, which makes the grammar active in all dialogs of the current document (and relevant application leaf documents), or dialog, which makes the grammar active throughout the current form. If omitted, the scope is determined by the parent element.
type	x-dtmf or regex

<else>

Marks the beginning of content to be executed when the parent <if> tag and all <elseif> tags at the same level of nesting have conditions that evaluate to false. Ends at the closing <if>.

The value of this tag must be empty.

Parent: <if>

Attributes: None.

<elseif>

Marks the beginning of content to be executed when the parent <if> and all <elseif> tags evaluate to false and the cond attribute evaluates to true. Ends at the next <elseif/> or <else> or closing <if> tag, whichever comes first.

The value of this tag must be empty.

Parent: <if>

Attributes:

Attribute	Description
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.

<error>

This element receives (catches) an error event. It is shorthand for <catch event="error"> and receives all events of type 'error'.

The value can be empty or can contain one or more child elements listed below, where the element is either parsed text or the tag.

Parents: <field>, <form>, <initial>, <menu>, <record>, <subdialog>, <vxml>

Child Elements: <assign>, <audio>, <clear>, <disconnect>, <exit>, <goto>, <if>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>

Attributes:

Attribute	Description
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.
count	The occurrence of the event. (Default is 1.) The count allows you to handle different occurrences of the same event differently. Each form item and <menu> maintains a counter for each event that occurs while it is being visited; these counters are reset each time the <menu> or form item's <form> is re-entered.

<exit>

Halts all loaded documents and returns control to the interpreter context. Once <exit> returns control to the interpreter context, the interpreter context is free to do as it wishes. For example, it can play a top level menu for the user, drop the call, or the user to an operator.

The value of this tag must be empty.

Parents: <block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>

Attributes: The IP Media Server returns no values and ignores the values specified in the expr or namelist attributes if they are supplied.

<field>

An input field that collects prompt-solicited user input within a form. Acceptable input is specified by type.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parent: <form>

Child Elements: <audio>, <catch>, <dtmf>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>, <prompt>, <property>, <value>

Attributes:

Attribute	Description
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.
expr	Initial value of the form-item variable; the default is ECMAScript undefined. If initialized to a value, then the form item is not visited unless the form-item variable is cleared.

Attribute	Description
modal	If FALSE (the default), all active grammars are turned on while collecting this field. If TRUE, only the field's grammars are enabled: all others are temporarily disabled.
name	Field-item variable in the dialog scope that holds the result.
slot	Name of the grammar slot used to populate the variable. If it is absent, it defaults to the variable name.
type	ype of field, for example digits, currency, phone, date, Boolean, number, time. If not present, <grammar> and/or <dtmf> elements can be specified instead.

<filled>

Specifies an action to perform when some combination of fields are filled by user input. It can occur in two places: as a child of the <form> element, or as a child of a field item.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parents: <field>, <form>, <record>, <subdialog>

Child Elements: <assign>, <audio>, <clear>, <disconnect>, <exit>, <goto>, <if>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>

Attributes:

Note: Attributes are valid only when <filled> is a child of <form>.

Attribute	Description
mode	Any or all. If any, action is executed when any field is filled by last user input, If all, action executes when all fields are filled.
namelist	The names of the form's field items.

<form>

One of two kinds of dialogs for collecting user input. The other is <menu>.

The value of this tag can be empty or can contain one of more of the child tags listed below.

Parent: <vxml>

Child Elements: <block>, <catch>, <dtmf>, <error>, <field>, <filled>, <help>, <initial>, <link>, <noinput>, <nomatch>, <property>, <record>, <subdialog>, <var>

Attributes:

Attribute	Description
id	Name of the form.
scope	Default scope of the form's grammars.

<goto>

Used in executable content to cause a transition to another form item in the current form, another dialog in the current document, or another document.

The value of this tag must be empty.

Parents: <block>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>

Attributes: One of the following values must be specified:

Attribute	Description
expr	ECMAScript expression that yields the target URI.
expritem	ECMAScript expression that yields the name of the target form item.
next	URL to which to transition.
nextitem	Name of the next form item in the current form.
fetchtimeout	Interval to wait for the content to be returned before generating an error.badfetch event. If not specified, a value derived from the innermost fetchtimeout property is used.

<grammar>

This element is identical to <dtmf>, because the Dialogic® IP Media Server supports only DTMF grammars.

<help>

Receives a help event, and is shorthand for <catch event="help">. On the Dialogic® IP Media Server, there is currently no way for DTMF input to automatically throw a help event, unlike a speech recognition interface. The <help> tag is fully supported because an explicit grammar can define a DTMF sequence that causes the help event to be generated.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parents: <field>, <form>, <initial>, <menu>, <record>, <subdialog>, <vxml>

Child Elements: <assign>, <audio>, <clear>, <disconnect>, <else>, <elseif>, <exit>, <goto>, <if>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>

Attributes:

Attribute	Description
count	The occurrence of the event. (Default is 1.) The count allows you to handle different occurrences of the same event differently. Each form item and <menu> maintains a counter for each event that occurs while it is being visited; these counters are reset each time the <menu> or form item's <form> is re-entered.
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.

<if>

Used for conditional logic. It has optional <else> and <elseif> elements.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parents: <block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>

Child Elements: <assign>, <audio>, <clear>, <disconnect>, <else>, <elseif>, <exit>, <goto>, <if>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>

Attributes:

Attribute	Description
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.

<initial>

Declares initial logic upon entry into a mixed-initiative form.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parent: <form>

Child Elements: <audio>, <catch>, <error>, <help>, <link>, <noinput>, <nomatch>, <prompt>, <property>, <value>

Attributes:

Attribute	Description
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.
expr	Initial value of the form-item variable; the default is ECMAScript undefined. If initialized to a value, then the form item is not visited unless the form-item variable is cleared.
name	The name of a form-item variable used to track whether the <initial> is eligible to execute; defaults to an inaccessible internal variable.

<link>

Specifies a destination for a transition or an event to be thrown when input matches.

The value of this tag can be empty or can contain one or more grammar tags.

Parents: <dtmf>, <field>, <form>, <initial>, <vxml>

Attributes: Must specify one of next, expr, or event.

Attribute	Description
next	URI to go to. Either a document (perhaps with an anchor to specify the starting dialog), or a dialog in the current document (just a bare anchor).
expr	Like next, except that the URI is dynamically determined by evaluating the given ECMAScript expression.
event	The event to generate when the user matches one of the link grammars.
fetchtimeout	Interval to wait for the content to be returned before generating an error.badfetch event. If not specified, a value derived from the innermost fetchtimeout property is used.

Dialogic's VoiceXML browser uses link grammars to support skip forward, skip back, and pause operations (VCR controls) for audio prompts.

<log>

When the log dest attribute is set to a value of SNMP, the VoiceXML browser sends out the msVXMLCriticalError trap to all SNMP trap hosts configured on the IP Media Server.

Attributes:

Attribute	Description
dest	When set to SNMP, this instructs the browser to send out the msVXMLCriticalError trap to all SNMP trap hosts configured on the IP Media Server. The text value of the <log> element can be used to contain the details about the error. This content is placed in the msVXMLLastCriticalError object. This object is passed as a varbind of the msVXMLCriticalError trap. Following is an example <log> tag with the trap extension: `<log dest="snmp">Unable to contact VoiceXML application server.</log>`

<menu>

Dialog for making a selection among choices.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parents: <vxml>

Child Elements: <audio>, <catch>, <choice>, <error>, <help>, <noinput>, <nomatch>, <prompt>, <property>, <value>

Attributes:

Attributes	Description
id	Identifier of the menu. It allows the menu to be the target of a <goto> or a <submit>.
scope	Menu's grammar scope. Either dialog (the default) or document.
dtmf	TRUE or FALSE. WhenTRUE, any choices that do not have explicit DTMF elements are given implicit ones: 1, 2, etc.

<meta>

Page properties as meta-data, as in HTML.

Note: This is ignored by the VoiceXML Interpreter.

The value of this tag must be empty.

Parent: <vxml>

Attributes:

Attribute	Description
name	Name of the meta-data property.
http-equiv	Name of an HTTP response header. Either name or http-equiv must be specified, but not both.
content	Value of the meta-data property. **Required.**

<noinput>

A type of catch element event used when the user does not respond within the required timeout interval. For example, <noinput> Shorthand for <catch event="noinput">. See <catch>.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parents: <field>, <form>, <initial>, <menu>, <record>, <subdialog>, <vxml>

Child Elements: <assign>, <audio>, <clear>, <disconnect>, <exit>, <goto>, <if>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>

Attributes:

Attribute	Description
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.
count	A number that allows you to issue different prompts if the user is doing something repeatedly. If omitted, it defaults to 1.

<nomatch>

Used to catch a nomatch event. Shorthand for <catch event="nomatch">. See "<catch>" (page 256).

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parents: <field>, <form>, <initial>, <menu>, <record>, <subdialog>, <vxml>

Child Elements: <assign>, <audio>, <clear>, <disconnect>, <exit>, <goto>, <if>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>

Attributes:

Attribute	Description
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.
count	A number that allows you to issue different prompts if the user is doing something repeatedly. If omitted, it defaults to 1.

<param>

Specifies the values that are passed to subdialogs.

The value of this tag must be empty.

Parent: <subdialog>

Attributes:

Attribute	Description
name	Name to be associated with this parameter when the subdialog is invoked. **Required.**
expr	Expression that computes the value associated with name.
value	Associates a literal string value with name.
valuetype	Whether the value associated with name is data or a URI (ref).
type	MIME type of the result provided by a URI if the value type is ref; only relevant for uses of <param> in <object>.

<prompt>

Controls the output of prerecorded audio only.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parents: <block>, <catch>, <error>, <field>, <filled>, <help>, <if>, <initial>, <menu>, <noinput>, <nomatch>, <return>, <subdialog>

Child Elements: <audio>, <break>, <sayas>, <value>

Attributes:

Attribute	Description
bargein	Whether a prompt may be interrupted. (Default: TRUE.)
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.

Attribute	Description
count	A number that allows you to issue different prompts if the user is doing something repeatedly. If omitted, it defaults to 1.
timeout	How long to wait for the next input. The default noinput timeout is 5 seconds.
VCR	Whether VCR controls are to be active for the prompt block. (Default: FALSE.)

<property>

Sets a property value that affects platform behavior.

The value of this tag must be empty.

Parents: <field>, <form>, <initial>, <menu>, <record>, <subdialog>, <vxml>

Attributes:

Attribute	Description
name	Property name. **Required.**
value	Property value. **Required.**

<record>

A field item that collects a recording from the user. The recording is stored in the field item variable, which can be played back or submitted to the server. Only external storage is available.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parent: <form>

Child Elements: <audio>, <catch>, <error>, <filled>, <help>, <noinput>, <nomatch>, <prompt>, <property>, <value>

Attributes:

Attribute	Description
beep	TRUE or FALSE (the default). If TRUE, the application plays a tone just prior to recording.
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.

Attribute	Description
dtmfterm	TRUE or FALSE. If TRUE (default), a DTMF keypress terminates the recording.
expr	Initial value of the form-item variable; the default is ECMAScript undefined. If initialized to a value, then the form item is not visited unless the form-item variable is cleared.
finalsilence	Interval of silence that indicates the end of speech. (Default: 3 seconds.)
max-time	Maximum duration to record. (Default: 10 seconds.)
name	The name of a form-item variable used to track whether the <initial> is eligible to execute; defaults to an inaccessible internal variable.
type	MIME type of the recording.

<reprompt>

Used inside executable content to set a flag indicating that a new attempt should be made to issue a prompt for the current item.

The value of this tag must be empty.

Parents: <block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>

Attributes: None.

<return>

Returns execution of a subdialog and returns control and data to the calling dialog.

The value of this tag must be empty.

Parents: <block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>

Attributes:

Attribute	Description
event	Returns, then sends this event.
namelist	Variable names to be returned to the calling dialog.

<sayas>

Type of text construct contained within the element that defines the way a word or phrase is spoken, including text generated by the <value> tag. The rules to convert <sayas> text into pre-recorded speech fragments are determined by

an external script. The default URI of this script is http://localhost/snowshore/vxmlphrase.cgi. This URI can be modified to use an external server by the com.snowshore.recsrc property.

The default rules handle US English only. The value of this tag must be parsed text.

Parents: <audio>, <choice>, <prompt>

Attributes:

Attribute	Description
class	Describes the way to render the value of: • phone (NANP numbers) • digits (Each spoken with no pauses.) • number (Spoken as a positive cardinal number. Negative numbers are not supported and will produce odd output.)

<script>

Allows the specification of a block of client-side ECMAScript code.

Note: Although the Dialogic® IP Media Server supports this tag, Dialogic does not recommend its use and considers it inadvisable to execute arbitrary logic on the IP Media Server due to resource limitations.

The value of this tag must be parsed text.

Parents: <block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>, <vxml>

Attributes:

Attribute	Description
src	URL for the resource.

<subdialog>

Invokes a second dialog within the current dialog. It provides a way for invoking a new interaction, and returning to the original form. Local data, grammars, and state information are saved and available when returning to the calling document.

The value can be empty or can contain one or more of the child elements listed below, where the element is either parsed text or the tag.

Parent: <form>

Child Elements: <audio>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>, <param>, <prompt>, <property>, <value>

Attributes:

Attribute	Description
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.
expr	Initial value of the form-item variable; the default is ECMAScript undefined. If initialized to a value, then the form item is not visited unless the form-item variable is cleared.
method	Request method (get or post).
modal	Controls which grammars are active during the subdialog.
name	The result returned from the subdialog, an ECMAScript object whose properties are the ones defined in the namelist attribute of the <return> element.
namelist	List of variables to submit to the subdialog. Same as namelist in <submit>, except that the default is to submit nothing. Only valid when fetching another document.
src	URI of the subdialog. **Required.**

<submit>

This element is like <goto> because the application gets a new document. Unlike <goto>, it lets you submit a list of variables to the document server, using an HTTP GET or POST request.

The value of this tag must be empty.

Parents: <block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>

Attributes:

Attribute	Description
expr	Like next, except that the URI is dynamically determined by evaluating the given ECMAScript expression. One of next, expr, or fetchtimeout is required.
method	The request method (GET or POST). The default is GET.
namelist	The list of variables to submit.
next	The URL to which the query is submitted.
fetchtimeout	Interval to wait for the content to be returned before generating an error.badfetch event. If not specified, a value derived from the innermost fetchtimeout property is used.

<throw>

Sends an event to be received by <catch>.

The value of this tag must be empty.

Parents: <block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>

Attribute:

Attribute	Description
event	Returns, then sends this event.

<transfer>

Provides the ability to place an outgoing voice call to support transfers. A "bridged" transfer session returns the caller to the original session with the interpreter upon completion. A "blind" transfer session terminates when completed.

Parent: <form>

Attributes:

Attribute	Description
name	The outcome of the transfer attempt. For possible return values, see the following table.
expr	Initial value of the form-item variable; the default is ECMAScript undefined. If initialized to a value, then the form item is not visited unless the form-item variable is cleared.
cond	Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.
dest	The URL of the destination (phone, IP telephone address).
destexpr	An ECMAScript expression yielding the URI of the destination.
bridge	What to do once the call is connected. If bridge is TRUE, document interpretation suspends until the transferred call terminates. Blind transfer is supported by hairpinning RTP through the IP Media Server.
connect	Time to wait while trying to connect the timeout call before returning the noanswer condition. Default is platform specific.
maxtime	Time that the call is allowed to last, or 0 if it can last arbitrarily long. Only applies if bridge is TRUE. (Default: 0.)
transfer	URI of audio source to play while the audio transfer attempt is in process (before far-end answer). If the resource cannot be fetched, the error is ignored and the transfer continues.
reqUri	SIP URI to be used as the Request-URI in the outbound INVITE. The value must be compliant with the grammar for a SIP URL. If this attribute is not present the Request-URI will have the same value as the To header. The To URI value is determined by the dest attribute.
requri expr	An ECMAscript expression that evaluates to a SIP URI.
src	The SIP URI that shows up as ANI at the callee. Must be of the form sip:user@server. This is to identify the caller for billing purposes. (Dialogic-specific attribute)
srcexpr	An ECMAScript expression yielding src. (Dialogic-specific attribute).
stopdigits	Defines the list of single DTMF digits that ends the transfer.
longdigit	Enables detection of long DTMF digits or multiple, short instance of the same digit.
video	Whether video is available on the outbound call. The value can be one of suppress or allow. See "RTP Codec Selection with VoiceXML <transfer/>" (page 306).

Note: The src, srcexpr, stopdigits, longdigit, and video attributes are Dialogic-specific attributes.

Return values of Name variable:

Value	Description
busy	The endpoint refused the call.
noanswer	There was no answer within the specified time.
network busy	Some intermediate network refused the call.
near_end_disconnect	The call completed and was terminated by the caller.
far_end_disconnect	The call completed and was terminated by the callee.
network_disconnect	The call completed and was terminated by the network.

<value>

Inserts the value of an ECMAScript expression into a prompt.

The value of this tag must be empty.

Parents: <audio>, <block>, <catch>, <choice>, <error>, <field>, <filled>, <help>, <if>, <initial>, <menu>, <noinput>, <nomatch>, <record>, <subdialog>

Attributes:

Attribute	Description
expr	Expression to render. **Required.**
class	Data type of the value.
recsrc	URI used to determine text to prompt conversion. Defaults to the value of com.snowshore.recsrc if specified, otherwise it is "http://locahost/snowshore/phrase.cgi?".

<var>

Declares an ECMAScript variable. The scope of the variable is determined by the parent tag.

The value of this tag must be empty.

Parents: <block>, <catch>, <error>, <filled>, <form>, <help>, <if>, <noinput>, <nomatch>

Attributes:

Attribute	Description
name	Name of the variable. **Required.**
expr	An initial value.

<vxml>

The top-level element in a VoiceXML document.

The value of this tag must contain one of more of the child elements listed below.

Parent: None

Child Elements: <catch>, <error> <form>, <help>, <link>, <menu>, <meta>, <noinput>, <nomatch>, <property>, <script>, <var>

Attributes:

Attribute	Description
version	Version of VoiceXML. **Required.**
base	Base URL. The root document URL, if any.

VoiceXML Attributes Reference

This section describes the VoiceXML attributes supported by the Dialogic® IP Media Server.

application

Specifies the URI of the root document.

Attribute for: <vxml>

bargein

Specifies whether user input can interrupt a prompt.

The IP Media Server supports the energy bargein type. The prompt is stopped if a DTMF tone is detected.

Attribute for: <prompt>

Values:

Value	Description
true (default)	Allow input to interrupt.
false	Do not allow input to interrupt.

bargeintype

Type of interrupt (bargein) the IP Media Server performs in response to DTMF inputs.

Attribute for: <prompt>

Values:

Value	Description
true	The default value of bargein is true.
false	The default value of bargein is false.

Note: Dialogic recommends running several tests using all combinations of the prompt element and its attributes.

base

The base URL for interpreting relative URLs.

Attribute for: <vxml>

beep

Whether to play a tone before recording.

Attribute for: <record>

Values:

Value	Description
true	Plays a tone when recording starts.
false (default)	Does not play tone when recording starts.

bridge

Controls the platform behavior once the call is connected. If set to true, the application stops interrupting documents until the transferred call terminates, at which time document processing resumes.

If set to false, document interpretation is also suspended but terminates immediately after the transferred call ends. This behavior does not follow the VoiceXML specification which calls for the interpreter to throw a telephone.disconnect.transfer event as soon as the call connects. Bridged transfer is currently implemented by hair-pinning media from both calls through the IP Media Server, which requires the VoiceXML session to continue its existence for the lifetime of the call transfer.

Attribute for: <transfer>

Values:

Value	Description
true	Suspends document interpretation until the transferred call terminates, at which time document processing resumes.
false (default)	Suspends document interpretation but terminates immediately after the transferred call ends.

class

The data type of the value.

Attribute for: <sayas> <value>

Values:

Value	Description
phone	The value is spoken as a phone number, with a leaning toward the North American Numbering Plan (NANP). 10 digit numbers have breaks after the 3rd and 6th digits. 7 digit numbers have breaks after the 3rd digit. 4 digit numbers are spoken as is. 11 digit numbers, where the first digit is a one, are spoken with a pause after the 1st, 4th, and 7th digit. Other length numbers are spoken with a pause after the 2nd digit, and each 3rd digit thereafter.
digits	Each digit is spoken with no pauses.
number	The value is spoken as a positive cardinal number. Negative numbers are not supported and will produce odd output.

cond

Boolean ECMAScript expression that serves as a guard.

Attribute for: <block>, <catch>, <elseif>, <error>, <field>, <help>, <if>, <initial>, <noinput>, <nomatch>, <prompt>, <record>, <subdialog>, <transfer>

Values:

Value	Description
true (default)	The form item is executed.
false	The form item is ignored or skipped; the form is not visited.

connect-timeout

Time to wait while trying to connect the call before returning a noanswer condition.

Attribute for: <transfer>

Values: Positive integers expressed in milliseconds or seconds when a qualifier “s” is appended to the value, for example 1s.

content

Value of the meta-data property.

Attribute for: <meta>

count

Repeat count for prompt or handler.

Attribute for: <catch>, <error>, <help>, <noinput>, <nomatch>, <prompt>

Values:

- For prompts, the minimum count for the prompt to be played. Set to zero when dialog is initialized. Increases by one each time user is prompted. The <clear> tag resets it to zero.
- For event handlers, the minimum count for the handler to be eligible to handle an event. Set to zero at initialization and increased by one each time event is triggered.

dest

Destination address for outbound calls initiated using <transfer/>.

Attribute for: <transfer>

Value: The dest attribute must be in the form of a valid SIP URI, for example sip:19783678400@gateway.carrier.net. (Default: none.)

destexpr

An ECMAScript expression yielding the destination address for outbound calls initiated via <transfer/>.

Attribute for: <transfer>

Value: The expression must produce a valid SIP URI, for example sip:19783678400@gateway.carrier.net. (Default: none.)

dtmf

Specifies touch-tone.

Attribute for: <choice>, <menu>

- choice Text /CDATA (touch-tone digit for this choice).
- menu Dialog for making a selection among choices. See "<menu>" (page 265).

Values:

Value	Description
true	Assigns touch-tone digits.
false	Does not assign touch-tone digits.

dtmfterm

Whether to allow touch-tone interruption.

Attribute for: <record>

Values:

Value	Description
true (default)	Touch-tone terminates recording.
false	Touch-tone does not terminate recording.

event

Name of an event.

Attribute for: <catch>, <choice>, <link>, <return>, <throw>

expr

ECMAScript expression to evaluate. Depending on the tag, used to set a variable, a URL, or for other purposes.

Attribute for: <assign>, <audio>, <block>, <choice>, <exit>, <field>, <goto>, <initial>, <link>, <param>, <record>, <subdialog>, <submit>, <value>, <var>, <transfer>

Value: Text (CDATA).

expritem

ECMAScript that provides item name.

Attribute for: <goto>

Value: Text (CDATA).

finalsilence

Interval of silence that indicates end of speech.

Attribute for: <record>

Value: Text (CDATA). (Default: 3 seconds.)

http-equiv

The HTTP response header field name.

Attribute for: <meta>

Value: Text (NMTOKEN)

id

Names an element for later reference.

Attribute for: <form>, <menu>

Value: URL#anchorname is the syntax.

longdigit

Turns detection on or off on long DTMF digits or multiple, short instances of the same digit in a two-second interval.

Attribute for: <transfer>

Values:

Value	Description
yes	Turns DTMF detection on for long digits.
no (default)	Turns DTMF detection off for long digits.

max-time

Maximum time the call is allowed to last, or 0 (zero) if call duration is unlimited. This attribute only applies when the bridge attribute is set to true.

Attribute for: <transfer>

Values: Positive integers expressed in milliseconds or seconds (when a qualifier "s" is appended to the value, for example 1s). (Default: 0. (zero))

method

The request method.

Attribute for: <subdialog> <submit>

Values:

Value	Description
get (default)	HTTP GET method.
post	HTTP POST method.

modal

Whether to enable outside grammars.

Attribute for: <field>, <subdialog>

Values:

Value	Description
true (default)	Disable higher-level grammars in the scope.
false	Enable higher-level grammars in the scope.

mode

Specifies mode for performing action when fields are filled in a form.

Attribute for: <filled>

Values:

Value	Description
all (default)	Action executes when all fields are filled.
any	Action is executed when any field is filled by user input.

msecs

Number of milliseconds to pause.

Attribute for: <break>

Value: Text. A length of time in milliseconds as a numerical value.

name

Name of an item, variable, or parameter.

Attribute for: <assign>, <block>, <field>, <initial>, <meta>, <param>, <property>, <record>, <subdialog>, <var>, <transfer>

Values: Text.

namelist

List of variable names.

Attribute for: <clear>, <data>, <exit>, <filled>, <return>, <subdialog>, <submit>

Value: A white space-separated list of variable or field names.

next

URL of the next page or dialog. This element is similar to <goto>, because it results in a new dialog being obtained. Unlike <goto>, it lets you submit a list of variables to the document server using an HTTP GET or POST request.

Attribute for: <choice>, <goto>, <link>

nextitem

First item to visit in the next dialog.

Attribute for: <goto>

Value: Text (NMTOKEN)

recsrc

Method to use when concatenating audio.

Attribute for: <value>

Value: The value of expr is expanded into a list of audio files by a CGI script external to the VoiceXML process. The default URI of this script is http://localhost/Cantata/vxmlphrase.cgi. This URI can be modified to use an external server by the com.Cantata.recsrc property, or by setting the recsrc attribute. The class and value of the phrase to be rendered are sent as properties of the CGI script "class" and "value".

reqUri

SIP URI to be used as the Request-URI in the outbound INVITE. The value must be compliant with the grammar for a SIP URL. If this attribute is not present the Request-URI will have the same value as the To header. The To URI value is determined by the dest attribute.

scope

Range where an element can be used.

Attribute for: <form>, <grammar>, <menu>

Values:

Value	Description
dialog	Scope is the current dialog.
document	Scope is the current document.

size

Relative pause duration.

Attribute for: <break>

Values:

Value	Description
none	0 ms
small	50 ms
medium (default)	200 ms
large	500 ms

slot

Name of the grammar slot used to populate the variable. If the grammar slot is absent, it defaults to the variable name. This attribute is useful when the grammar format being used has a mechanism for returning sets of slot/value pairs and the slot names differ from the field item variable names. If the grammar returns only one slot, as do the built-in type grammars like Boolean, then no matter what the slot's name, the field item variable gets the value of that slot.

Attribute for: <field>

Value: Text (NMTOKEN)

src

Calling party identification (ANI) used when placing the outbound call through <transfer>. This identifies the caller for billing purposes.

Attribute for: <transfer>

Value: Must be in the form of a valid SIP URI, for example subscriber103@vm.carrier.net. (Default: none.)

srcexpr

ECMAScript expression yielding the calling party identification.

Attribute for: <transfer>

Value: The expression must produce a valid SIP URI, for example `sip:subscriber103@vm.carrier.net`.

stopdigits

List of DTMF digits that terminate a transfer.

Attribute for: <transfer>

Values: 0-9, #, A-D. (Default: #.)

timeout

Maximum time to wait before triggering a <noinput> event.

Attribute for: <prompt>

Value: Text. A numerical value with optional decimals, followed by "s" for seconds or "ms" for milliseconds.

transfer-audio

URI of the audio source to play while the transfer attempt is in process. If the resource cannot be fetched, the application ignores the error and continues the transfer.

Attribute for: <transfer>

Value: Valid HTTP or file scheme URI. (No default.)

type

Indicates how to interpret text to be spoken. The meaning of this attribute depends on the tag: for dtmf, grammar, object, param, and record, it is a content type (MIME type); for field and say-as, it is a data type.

Attribute for: <dtmf>, <field>, <grammar>, <object> ,<param>, <record>, <sayas>

Values:

Value	Description
dtmf	x-dtmf or regex.
field	A built-in type (Boolean, digits, phone, date, currency, number, time).
grammar	Content type of the grammar.
object	Content type of the object data.
param	MIME type of the result provided by a URI if the value type is ref; only relevant for uses of <param> in <object>.
record	MIME type of the recording. See "MIME Recording Encoding Types" (page 287).
sayas	Data type to use in interpreting the value for the following: telephone, date, digits, number.

value

Value to assign to the parameter or property named by the name attribute.

Attribute for: <param>, <property>

valuetype

Value to assign to the parameter or property named by the name attribute.

Attribute for: <param>

Values:

Value	Description
data (default)	Value is data.
ref	Value is a URL pointing to data.

version

Version of VoiceXML. Required.

Attribute for: <vxml>

Value: 1.0.

VCR

Whether VCR controls are active

Attribute for: <prompt>

Values:

Value	Description
true	VCR controls are enabled.

video

Whether video media is offered on outgoing calls.

Attribute for: <transfer>

Values:

Value	Description
allow	Video media is offered on outgoing calls.
suppress	Video media is not offered on outgoing calls.

See "RTP Codec Selection with VoiceXML <transfer/>" (page 306) for additional information on the detailed interaction of the video attribute and RTP codec selection.

VoiceXML Properties

com.snowshore.criticaldigit_timer

Sets a special digit timer that specifies the interval to wait for additional user input before returning a match to an active grammar. This property is used when the application modifies the DTMF matching behavior.

Uses the VoiceXML standard "n {ms}|{s}" timer notation. Default unit is milliseconds (ms).

Values:

Value	Description
-2	The default critical digit timer is set to the same value as the interdigit timer (thus it has no effect).
-1	Immediate. Use “shortest match first”.
0	Infinite. Use “longest match first”, but waits infinitely once a match is found. Thus, matched input must end with a return digit or restart digit.
>0	Uses “longest match first”, and waits that amount of time when a match is found.

MIME Recording Encoding Types

The MIME type and optional codec parameters define the file format and audio encoding for the recording. Supports both headerless (raw) and .wav file formats. Table 52 lists the MIME type, file format, and audio encoding.

Table 52. Supported MIME Types, Formats, and Audio Encodings

MIME Type	File Format	Audio Encoding
audio / basic	raw	G.711 µ-law
audio/ x-alaw-basic	raw	G.711 A law

There are various commonly-used methods of specifying .wav format and the desired audio encodings. The official IANA registered MIME type for wave is audio/vnd.wave but audio/wav and audio/x-wav are used frequently. Table 53 lists the accepted WAVE format specifiers.

Table 53. Accepted WAVE Format Specifiers

MIME Type	File Format	Notes
audio/vnd.wave	WAVE	IANA standard
audio/wav	WAVE	
audio/x-wav video/x-wav	WAVE WAVE	Proprietary

Specify audio encoding by adding a codec parameter and value to the base MIME type, for example:

```
audio/vnd.wave;codec=7
```

The official values for the codec parameter are defined by the IANA registry (http:/www.iana.org/assignments/wave-avi-codec-registry), but other commonly used forms are also supported.

Table 54 provides a summary of the supported codec parameters and values and their corresponding audio encodings.

Table 54. Codec Parameter Values

Value	Audio Encoding	Notes
7	G.711 µ-law	IANA standard
6	G.711 A law	IANA standard
31	Microsoft® GSM	IANA standard
ulaw	G.711 µ-law	
alaw	G.711 A law	
msgsm	Microsoft® GSM	

The video/x-wav MIME type always produces a file with a G.711 audio track and an H.263 video track. No codec parameters are supported for this type.

ECMAScript Functionality

Table 55 summarizes the categories and features supported by the IP Media Server's VoiceXML interpreter.

Note: The ECMAScript Language Specification can be downloaded for free.

Table 55. ECMAScript Functionality

Category	Feature/Keyword
Array Handling	Array join, length, reverse, sort
Assignments	Assign (=) Compound Assign (OP=)
Booleans	Boolean
Comments	/*...*/ or //
Constants/Literals	NaN null true, false Infinity undefined
Control flow	Break continue for for...in if...else return while
Dates and Time	Date getDate, getDay, getFullYear, getHours, getMilliseconds, getMinutes, getMonth, getSeconds, getTime, getTimezoneOffset, getYear, getUTCDate, getUTCDay, getUTCFullYear, getUTCHours, getUTCMilliseconds, getUTCMinutes, getUTCMonth, getUTCSeconds, setDate, setFullYear, setHours, setMilliseconds, setMinutes, setMonth, setSeconds, setTime, setYear, setUTCDate, setUTCFullYear, setUTCHours, setUTCMilliseconds, setUTCMinutes, setUTCMonth, setUTCSeconds, toGMTString, toLocaleString, toUTCString, parse, UTC

Table 55. ECMAScript Functionality (continued)

Category	Feature/Keyword
Declarations	Function new this var with
Function Creation	Function arguments, length
Global Methods	Global escape, unescape eval isFinite, isNaN parseInt, parseFloat
Math	Math abs, acos, asin, atan, atan2, ceil, cos, exp, floor, log, max, min, pow, random, round, sin, sqrt, tan, E, LN2, LN10, LOG2E, LOG10E, PI, SQRT1_2, SQRT2
Numbers	Number MAX_VALUE, MIN_VALUE NaN NEGATIVE_INFINITY, POSITIVE_INFINITY
Object Creation	Object new constructor, prototype, instanceof, toString, valueOf
Operators	Addition (+), Subtraction (-) Modulus arithmetic (%) Multiplication (*), Division (/) Negation (-) Equality (==), Inequality (!=) Less Than (<), Less Than or Equal To (<=) Greater Than (>) Greater Than or Equal To (>=) Logical And(&&), Or (||), Not (!) Bitwise And (&), Or (|), Not (~), Xor (^) Bitwise Left Shift (<<), Shift Right (>>) Unsigned Shift Right (>>>) Conditional (?:) Comma (,) delete, typeof, void Decrement (--), Increment (++)

Table 55. ECMAScript Functionality (continued)

Category	Feature/Keyword
Objects	Array Boolean Date Function Global Math Number Object String
Strings	String charAt, charCodeAt, fromCharCode indexOf, lastIndexOf split toLowerCase, toUpperCase length

Support for VoiceXML Extended Session Variables

The Dialogic® VoiceXML browser supports the following VoiceXML 1.0 session variables:

- session.telephone.ani
- session.telephone.dnis
- session.telephone.rdnis (always set to NULL)
- session.telephone.redirect_reason

In addition to the above, other session variables are supported so that a VoiceXML script can access a complete set of information from the SIP call. Some of the new session variables are defined in VoiceXML 2.0, and others are Dialogic® extensions as noted on the following page.

VoiceXML 2.0 Recommendations

session.connection.local.uri

Value: Set to the SIP Request-URI; note that any SIP escapes (%xx) in the Request-URI are still in this variable as well.

session.connection.remote.uri

Value: Set to the SIP From-URI.

session.connection.protocol.name

Value: Always SIP.

session.connection.protocol.version

Value: Always 2.0.

Dialogic® Extensions

session.connection.media

An array where each element represents an RTP media stream. Each array element has the following associated attributes:

- "type" - indicates the MIME type of media stream (currently either "audio" or "video")
- "subtype" - indicates the MIME sub-type, or encoding, of the media stream (e.g. PCMU, PCMA, H264, etc.)

session.connection.protocol.sip.parameter

Value: An array of name/value pairs parsed from the SIP Request-URI; the name and value are "UnEscaped," meaning any SIP escapes (%xx) in the URI are expanded before being assigned.

session.connection.protocol.sip.parameter[n].name

Value: The name of the nth parameter.

session.connection.protocol.sip.parameter[n].value

Value: The "value" (if any, can be empty) of the nth parameter.

session.connection.protocol.sip.uri

Value: Set to the SIP Request-URI. Note that any SIP escapes (%xx) in the Request-URI are still in this variable as well. This variable duplicates the session.connection.local.uri by design.

session.connection.protocol.sip.to

Value: Set to the SIP To-URI. Since the VoiceXML 2.0 defined session.connection.local.uri variable is mapped to the request URI a separate variable is required to contain the contents of the SIP To header.

session.connection.protocol.sip.from

Value: Set to the SIP From-URI. This variable duplicates the session.connection.remote.uri by design.

session.connection.protocol.sip.call_id

Value: Set to the SIP Call ID.

Example

The following example shows the mapping between SIP headers and the newly supported VoiceXML session variables.

Given the following SIP INVITE request:

```
INVITE
sip:dialog@10.102.4.26;voicexml=http://10.102.4.9:9020
   /ivr/sip_init.vxml%3fdnis%3d961234567%26hasvideo%3d
   1;dogs=nice;user=phone
SIP/2.0
From: <sip:968037431@10.102.4.45:5060>;tag=2d04660a-
   13c4-40ed433f-9ec19fd-1786
To: <sip:961234567@10.102.4.9>
Call-ID: 6a7b774-2d04660a-13c4-40ed433f-9ec19fd-
   7061@10.102.4.45
CSeq: 1 INVITE
Via: SIP/2.0/UDP
   10.102.4.134:5060;branch=z9hG4bKi8MKi8i!yi8MK2SsMu8
   Uyake2q0OUqMi8i!y.1-1d9515c
Via: SIP/2.0/UDP
   10.102.4.45:5060;received=10.102.4.45;branch=z9hG4b
   K-40ed433f-9ec19fd-847
Max-Forwards: 69
User-Agent: RADVision ViaIP GW Vers. 1.0
Call-Info: <Media:Video>;purpose=info
Contact: <sip:968037431@10.102.4.45:5060>
Content-Length:0
```

The Media Server populates the new VoiceXML session variables, as shown:

```
session.connection.protocol.name = "SIP"
session.connection.protocol.version = "2.0"
session.connection.local.uri =
   sip:dialog@10.102.4.26;voicexml=http://10.102.4.9:9
   020/ivr/sip_init.vxml%3fdnis%3d961234567%26hasvideo
   %3d1;dogs=nice;user=phone"
session.connection.remote.uri =
   <sip:968037431@10.102.4.45:5060>;tag=2d04660a-13c4-
   40ed433f-9ec19fd-1786
session.connection.protocol.sip.parameter[0].name =
   "voicexml"
session.connection.protocol.sip.parameter[0].value =
   "http://10.102.4.9:9020/ivr/sip_init.vxml?dnis=9612
   34567&hasvideo=1"
session.connection.protocol.sip.parameter[1].name =
   "dogs"
session.connection.protocol.sip.parameter[1].value =
   "nice"
session.connection.protocol.sip.parameter[2].name =
   "user"
session.connection.protocol.sip.parameter[2].value =
   "phone"
session.connection.protocol.sip.uri =
   sip:dialog@10.102.4.26;voicexml=http://10.102.4.9:9
   020/ivr/sip_init.vxml%3fdnis%3d961234567%26hasvideo
   %3d1;dogs=nice;user=phone"
session.connection.protocol.sip.to =
   <sip:961234567@10.102.4.9>
session.connection.protocol.sip.from =
   <sip:968037431@10.102.4.45:5060>;tag=2d04660a-13c4-
   40ed433f-9ec19fd-1786
session.connection.protocol.sip.call_id = 6a7b774-
   2d04660a-13c4-40ed433f-9ec19fd-7061@10.102.4.45
```

C - MSCML Schema

This appendix provides the MSCML schema, based on draft-vandyke-mscml-09, “Media Server Control Markup Language (MSCML) and Protocol”, Van Dyke, J., Burger, E., June 2006.

Example 46. MSCML 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>
```

D - RTP Codec Selection with VoiceXML <transfer/>

This appendix describes how the Real-Time Protocol (RTP) codec can work with the VoiceXML transfer element.

When <transfer/> is enabled, the IP Media Server sends a SIP INVITE with an SDP offer to the URI specified in the VoiceXML script. The contents of this offer, combined with capabilities of the called device, determine the type of media streams established. By default, the offer matches the capabilities of the existing, inbound call leg.

For example, if the inbound leg has both audio and video streams, the SDP offer might include media lines for both. In addition, customers have requested application-level control to override this standard behavior. With such control, the application can force an audio-only call even if the called device supports video.

To enable this control, the IP Media Server supports a proprietary VoiceXML attribute named video of the <transfer/> tag. This attribute has two supported values: allow and suppress.

Table 56 shows the media streams that are included in the SDP offer based on the negotiated SDP of the inbound call and the value of the video attribute. The default value of the video attribute is allow.

Table 56. SDP Offer Sent in Outbound SIP INVITE

Video Attribute Values			
		Allow Value	Suppress Value
Inbound Leg Negotiated SDP	Audio	Audio	Audio
	Audio+Video	Audio+Video	Audio

The called device on the outbound leg generates an SDP answer based on the IP Media Server's offer and its own capabilities. This answer determines the actual media streams that are created. So, the video attribute cannot guarantee that video media are used; only that the option is presented.

The IP Media Server also allows the client application to determine what media types were negotiated on the outbound call. The client application uses this function for billing purposes. A custom shadow variable to <transfer/> provides this information.

The IP Media Server standard supports:

```
<transfer name="name".../> shadow variables
   name$.duration, name$.inputmode and name$.utterance
```

The IP Media Server also supports the proprietary:

name$.media.

If the transfer did not connect to the far end, the value for the media shadow variable is "". If the media is connected with audio only, then the media shadow variable is set to audio. If the media is connected with both audio and video, then the shadow variable is set to audio+video.

E - Using AMR-NB

The Dialogic® IP Media Server supports the following features of the Adaptive Multi-Rate - Narrow Band (AMR-NB) codec:

- Octect-alignment: **Bandwidth Efficient** (bit-aligned) or **Octet-aligned** (byte-aligned) Mode.
- Mode-set: 0 – 7 (bit rates from 4.75–12.2)
- Simple payload sorting.
- 1 channel per session.

The following features are not supported by the IP Media Server:

- Mode switching / CMR (Codec Mode Request).
- Unequal Bit-error Detection (UED) and Unequal Bit-error_Protection (UEP), along with the associated checksums and or frame CRCs. (Refer to section 3.6 in rfc3267.)
- Forward Error Correction (FEC) or frame interleaving / redundant transmission. (Refer to section 3.7 in rfc3267.)
- Robust Sorting.
- Interleaving.
- maxptime > 40ms.

Note: Using the AMR-NB resource in connection with a Dialogic® product does not grant the right to practice the AMR-NB standard. To seek a patent license agreement to practice the standard, contact the VoiceAge Corporation at http://www.voiceage.com/licensing.php. For a listing of the AMR codec "essential patents," visit: http://www.voiceage.com/ess_patent.php

---

Using AMR-NB

To use AMR-NB encoded media with your application, send a SIP INVITE request to the IP Media Server including an SDP section that describes the requested AMR-NB session.

Note: Using the AMR-NB resource in connection with a Dialogic® product does not grant the right to practice the AMR-NB standard. To seek a patent license agreement to practice the standard, contact the VoiceAge Corporation at http://www.voiceage.com/licensing.php. For a listing of the AMR codec "essential patents," visit: http://www.voiceage.com/ess_patent.php

The following example is a typical SDP description for an AMR-NB session:

```
m=audio 49120 RTP/AVP 97
a=rtpmap:97 AMR/8000
a=fmtp:97 mode-set=0,2,5,7; octet-align=1
a=maxptime:20
```

The component lines of an SDP description are described in the following sections.

Note: Only the first two lines (m= and a=rtpmap:) are required.

Media Description Header

The “m=” media description header specifies the media name and transport address.

```
m=<media> <port> <transport> <fmt list>
```

Parameter	Description
media	“audio”
port	Port number
transport	“RTP/AVP”
fmt list	RTP dynamic payload type (number from 96–127)

For example, the line `m=audio 49120 RTP/AVP 97`, indicates the following:

- audio = audio media
- 49120 = using port number 49120
- RTP/AVP = IETF's Realtime Transport Protocol using the Audio/Video Profile carried over UDP
- 97 = RTP dynamic payload type ID

Dynamic Payload Type

The "a= rtpmap:" attribute line defines a mapping between a codec and a dynamic payload type number. The following description for the "a= rtpmap:" attribute line shows how AMR-specific information is provided.

```
a=rtpmap:<payload type> <encoding name>/
   <clock rate>[/<encoding parameters>]
```

Parameter	Description
payload type	RTP dynamic payload type specified in the `m=` line
encoding name	AMR
clock rate	8000
encoding parameters	Number of channels (Default:1)

For example, the attribute line `a=rtpmap:97 AMR/8000/1` indicates the following:

- 97= RTP dynamic payload type ID (97)
- AMR = Encoding type (name) is "AMR".
- 8000 = Clock rate of 8000 samples per second
- 1 = Using 1 channel (optional value).

Format Specific Parameters

The "a= ftmp:" attribute line specifies the characteristics of the AMR connection. The following description for the "a= ftmp:" attribute line shows how AMR-specific information is provided.

```
a=fmtp:<dpt> <format specific parameters>
```

Parameter	Description
dpt	RTP dynamic payload type (same value used in "m=" and "a=rtpmap:" lines)
format specific parameters	For AMR, a semi-colon separated list of **parameter=value** pairs (which may contain white space)

The AMR-specific parameters that can be received via the `a=fmtp:` line are described in the following table.

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>octet-align=X</td><td>Whether data is octet aligned..
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>Octet-aligned (byte-aligned) operation.</td></tr>
<tr><td>0
(default)</td><td>Bandwidth efficient (bit-aligned) operation.</td></tr>
</table></td></tr>
<tr><td>mode-set=X</td><td>Requested set of AMR modes; restricts the set of active codec modes to a subset of all modes. Possible values are a comma-separated list of modes from the set: 0,...,7 (representing the 8 possible bit rates from lowest to highest). (Default: all.)

The IP Media Server always employs the highest transmission rate available for sending AMR-encoded audio.</td></tr>
<tr><td>mode-change-period=N</td><td>Interval (number of frames) at which codec mode changes are allowed. The initial phase of the interval is arbitrary, but changes must be separated by multiples of N frames. If this parameter is not present, mode changes are allowed at any time during the session.

Note: The IP Media Server never initiates a mode change.</td></tr>
<tr><td>mode-change-neighbor=X</td><td>Whether mode changes are allowed to the neighboring modes in the active codec mode set. Neighboring modes are the ones closest in bit rate to the current mode, either the next higher or next lower rate.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>Mode changes are allowed only to the neighboring modes in the active codec mode set.</td></tr>
<tr><td>0
(default)</td><td>Change is allowed between any two modes in the active codec mode set.</td></tr>
</table>
Note: The IP Media Server never initiates a mode change.</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>crc=X</td><td>Whether CRCs are included in the payload.

<table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>Frame CRCs may be included in the payload.
crc=1 implies that octet-aligned operation must be used for the session.</td></tr><tr><td>0
(default)</td><td>Frame CRCs shall not be included in the payload.</td></tr></table>
Note: The IP Media Server does not support including CRSCs in the payload. The IP Media Server rejects the call if the far end (caller) specifies that CRC information is included in the AMR payload.</td></tr>
<tr><td>robust-sorting=X</td><td>Whether the IP Media Server supports robust sorting.

<table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>The payload uses robust sorting.
robust-sorting=1 implies that octet-aligned operation must be used for the session.</td></tr><tr><td>0
(default)</td><td>The payload uses simple payload sorting.</td></tr></table>
Note: The IP Media Server does not support robust payload sorting. The IP Media Server rejects the call if the far end (caller) specifies robust sorting.</td></tr>
<tr><td>interleaving= ?</td><td>Note: The IP Media Server does not support interleaving. The IP Media Server rejects the call if the far end (caller) specifies interleaving.</td></tr>
<tr><td>channels=</td><td>Number of audio channels. (Default: 1.)
The IP Media Server supports only a value of 1.</td></tr>
</table>

For example, the attribute line `a=fmtp:97 mode-set=0,2,5,7; octet-align=1` indicates the following:

- 97= RTP dynamic payload type ID (97)
- `mode-set=0,2,5,7` = Available AMR modes are modes 0, 2, 5, and 7.
- `octet-align=1` = AMR data is byte (octet) aligned.

Optional Format Parameters

Additional optional attribute lines may also be included in the SDP section, for example:

Packet Time

```
a=ptime:<milliseconds>
```

Length of time represented by the media in a packet, in milliseconds. This is probably only meaningful for audio data. It should not be necessary to know ptime to decode RTP audio, and it is intended as a recommendation for the encoding / packetization of audio.

Maximum Packet Time

```
a=maxptime:20
```

Maximum amount of media that can be encapsulated in a payload packet, expressed as time in milliseconds. The time is calculated as the total time represented by the media present in the packet. The time should be a multiple of the frame size. If this parameter is not present, the sender MAY encapsulate a maximum of 16 encoded frames into one RTP packet. This is a media attribute, and is not dependent on charset.

Values supported by the IP Media Server: 20 (the desired value), 40 (the maximum value).

F - Dial Pulse Detection

This appendix describes the Dial Pulse Detection feature in the IP Media Server product.

Overview

In addition to the existing checks for Dual Tone Multi Frequency (DTMF) and Call Progress Analysis (CPA) tones, the IP Media Server can now be configured to detect Dial Pulses.

Dial Pulse and DTMF

This feature supports all of the features associated with the DTMF signal detection. It is designed to minimize the considerations that an application developer has to address when supporting both DTMF and Dial Pulse detection.

The following applies to Dial Pulse detection and DTMF detection:

- Dial Pulse detector can detect pulse dialed digits in any call state that allows DTMF detection.
- Dial Pulse detection runs concurrently with DTMF detection. You can configure the Dial Pulse detector to disable itself when it first receives a DTMF digit. This option is configured by the parameter DP_detectAfterDtmf (See Table on page 316.)
- The mserv process detects Dial Pulse and DTMF within a single stream. As a result, when mserv detects each type of signaling, it checks for the completion of any previously detected digits of the alternate type.

Consistency

In order for mserv to detect Dial Pulse digits, they must be fairly consistent in form and meet the criteria defined in the configuration parameters. You can manipulate these parameters to meet the dial pulse criteria for differing networks and implementations.

MSCML and VXML 1.0/2.0 Support

Dial Pulse detection is compatible with MSCML and VXML applications - both VXML versions 1.0 and 2.0.

When collecting dial pulse digits under VXML, the following VXML properties can be set so collection will not time out. Values will depend on the application and average dial pulse timeout.

```
<property name="timeout" value="10s"/>
<property name="interdigittimeout" value="10s"/>
```

The default values of these properties for VXML 1.0 and 2.0 are different. If the interdigit timing is large, VXML may declare a timeout while collecting digits. VXML 2.0 is more sensitive to this issue as its default values are smaller. Use the VXML code provided to change these default values.

Configuration

Dial Pulse detection is configured separately from DTMF detection. DTMF is always detected and Dial Pulse detection is configurable.

Dial Pulse is configured with the DP_DetectAfterDtmf parameter. If that parameter is set to Y, the IP Media Server detects Dial Pulse after detecting a DTMF. If configured to N, the IP Media Server no longer detects Dial Pulse after detecting a DTMF.

Parameters

The mserv process detects the properties of a dial pulse such as those represented by the parameters in the following table. You can configure these parameters only in the snowshore.cfg file (keyword=value)

Table 57. Dial Pulse Parameters

Parameter	Description	Default (in Decimal)	Range
DP_enabled	DP Enabled Determines if the Dial Pulse detection is enabled or disabled system-wide	N	Y or N
DP_detectAfterDtmf	DP Detect After Dtmf Determines if Dial Pulses should be detected on a call leg after DTMF has been detected on that leg.	N	Y or N
DP_low_Thresh	Low Threshold Level Sample level required to be below in order to declare sample as a Low Signal	5,000	0 to 32768
DP_lowThreshCnt	Low Threshold Count Number of consecutive Low samples to be considered a post Dial Pulse Low (or trough)	17	0 to 32768
DP_peakThresh	Peak Threshold Sample value required for signal to be declared a dial pulse peak	14,000	20 to 32768

Parameter	Description	Default (in Decimal)	Range
DP_maxTransCnt	Max Transition Count Maximum low to high transition time (samples). For a peak to be declared, the ramp time has to be less than this number of samples	30,000	0 to 32768
DP_settleByCnt	Settle By Count Number of consecutive samples that the signal must be below the threshold to declare the signal as a dial pulse.	400	0 to 65535
DG_InterDigitQuietCnt	Inter-Digit Quiet Count Number of 10 ms frames between the last pulse of a previous digit and the first pulse of the next digit.	30	20 to 32768

The following figure illustrates these parameters and the description follows.

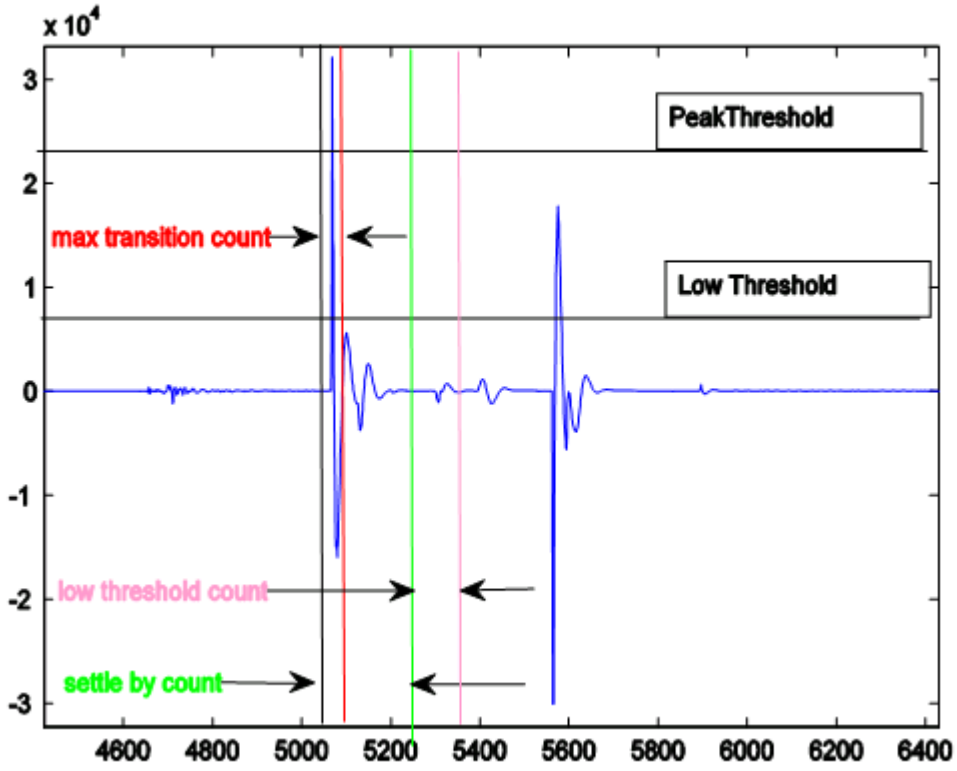

Figure 16. Dial Pulse Parameters

Description of Dial Pulse Algorithm

The first representative pulse is a digit 1. Note the pair of pulses. Each of the configurable parameters is indicated by a line on the figure. The second pulse is ignored by the algorithm because it does not exceed the peak threshold at any time.

The first pulse exceeds the Peak Threshold of approximately 25,000 within a small enough number of samples (defined by Max Trans Count) to constitute the start of Dial Pulse.

The first pulse then returns below the Low Threshold and remains there (settles) between the red and green vertical lines.

The number of packets received during this interval is the Settle By Count. This point is the end of a detected dial pulse.

Between the green and pink lines, mserv counts the consecutive samples the signal remains below the Low Threshold. This information is useful when determining the time spacing between pulses used to determine if two consecutive pulses are from the same digit or from different digits.

G - MRCP Session Management

This appendix describes MRCP Session Management and includes the following sections:

Overview of MRCP Session Management

Applications that include Automated Speech Recognition (ASR) and Text-to-Speech (TTS) features are most commonly written in VoiceXML. However, certain applications may require ASR and TTS capabilities which are not exposed through existing versions of the VoiceXML standard. Further, some developers may not wish to write or rewrite their applications using VoiceXML.

To address these situations the IP Media Server supports MSCML extensions to create and terminate MRCP v2 sessions on behalf of the application.

The application is responsible for implementing all required MRCP client functions. The IP Media Server's role is simply to manage the MRCP session and associated media streams.

Note: MRCP Session Management is supported with the following packages: Nuance OSR MRCP 3.0.10 and SWMS 4.0 running under Linux.

Features Enabled

MRCP Session Management enables the following features:

- speech synthesis
- speech recognition
- speech recording

Process

Figure 17 outlines the MRCP Session Management process. Each stage is explained in the table following the figure.

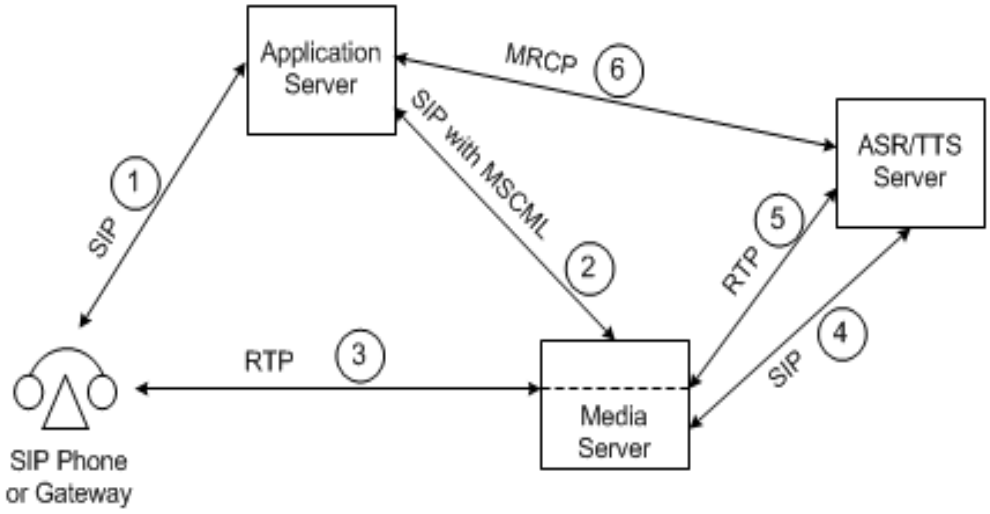

Figure 17. MRCP Session Management

Table 58. Description of MRCP Session Management Topology

Stage	Description
1	SIP session between a caller and the application server that invokes the application.
2	SIP session between the application server and IP Media Server used to set up RTP communication between the caller and IP Media Server and to control media processing. The application server uses this session to send MSCML <mrcp_session_create> and <mrcp_session_terminate> requests.
3	RTP session with caller. This is normally full duplex, as shown.
4	SIP session between the IP Media Server and the ASR/TTS server, which supports MRCP V2.
5	RTP session between the IP Media Server and the ASR/TTS server. This is normally half-duplex, since ASR and TTS rarely reside on the same server. In this example, media is sent from the IP Media Server to an ASR server, which performs speech recognition.
6	MRCP session between the application server and the ASR/TTS server. The application server uses the information contained in the IP Media Server's response to the <mrcp_session_create> request to establish this connection. Once the connection is established, the application server communicates directly with the ASR/TTS server using MRCP and starts a recognition session.

MSCML Requests

The following MSCML requests support MRCP Session Management. The formats and attributes of each are provided below.

- mrcp_session_create
- mrcp_session_terminate

These MSCML requests are scoped to the SIP dialog in which the requests are sent and apply to the dialog's associated RTP streams. Any usage of MRCP resources will create at least one new SIP dialog with the desired MRCP resource.

If multiple resource types are supported on a single MRCP server, the media server may reuse the existing dialog and simply send a re-INVITE with the SDP that describes the additional resource desired. However, different resources such as ASR and TTS are often deployed on separate servers for performance reasons so a separate SIP dialog will get established for each MRCP resource type in this configuration. The SDP for that dialog contains a single resource type.

The lifetimes of any SIP dialogs with MRCP resources are tied to the original SIP dialog with the application server. The media server must clean up any MRCP related SIP dialogs when the original SIP dialog is terminated. Standard session timers apply to MRCP dialogs.

Established MRCP sessions can only be ended by the <mrcp_session_terminate/> request or termination of the parent SIP dialog as described above. Unlike the <play>, <playrecord> and <playcollect> requests, the MRCP sessions are not stopped when another MSCML request is received.

Create session request format

The following is the MSCML format of the mrcp_session_create request.

<?xml version="1.0" encoding="UTF-8"?>

```
<MediaServerControl version="1.0">
   <mrcp_session_create id="1234"
   resource="speechrecog"
   url="sip:mrcpserver_address" proto=”TCP/MRCPv2”
   rtpdirection=”sendonly”/>
</MediaServerControl>
```

Attributes

The following are the attributes of the mrcp_create_session request.

Table 59. Attributes of mrcp_create_session

Attribute	Mandatory	Description
id	No	An application-provided transaction identifier.
resource	Yes	The resource to be provided by the MRCP server
url	Yes	The complete SIP URL of the MRCP server. Note that in actual deployments this will most likely be a SIP proxy, which will select a specific server and forward the request.
proto	No	The transport protocol to be used for the MRCP session. Alternatives are “TCP/MRCPv2” (default) and “TCP/TLS/MRCPv2”. This value is treated as case-sensitive in SDP and will be treated likewise here.
rtpdirection	No	The value to be used in the direction attribute of the SDP offer created by the media server. The attribute is interpreted from the perspective of the media server. This value is treated as case-sensitive in SDP and will be treated likewise here. If this attribute is not set the SDP, offer will be based on the resource required. For example the RTP stream to a “speechrecog” resource is send-only from the perspective of the media server.

Create session response format

The following is the MSCML format of the mrcp_session_response request.

```
<?xml version="1.0" encoding="UTF-8"?>
  <MediaServerControl version="1.0">
    <response id="1234" request="mrcp_session_create"
   code="200" text="OK"
      connection="192.168.16.2" port="32416"
   proto=”TCP/MRCPv2”
      channel="32AECB234338@speechrecog"/>
  </MediaServerControl>
```

Table 60. Attributes of mrcp_session_response

Attribute	Required	Description
id	No	An application-provided transaction identifier.
connection	Yes	The IP address of the MRCP server the client connects to.
port	Yes	The port of the MRCP server that the client connects to.
proto	Yes	The transport protocol to be used for the MRCP session. Alternatives are “TCP/MRCPv2” and “TCP/TLS/MRCPv2”.
connectiontype	Yes	The value of the “a=connection” attribute associated with the MRCP control channel from the MRCP server’s SDP answer. Alternatives are “new” and “existing.” There is no MSCML default value - the MRCP server is required to provide this information in its SDP answer.
channel	Yes	The channel identifier returned from the MRCP server. The client uses it to communicate with the server.

Terminate session request format

```
<?xml version="1.0" encoding="UTF-8"?>
  <MediaServerControl version="1.0">
    <mrcp_session_terminate id="1234"
   channel=”32AECB234338@speechrecog"/>
  </MediaServerControl>
```

Table 61. Attributes of mrcp_terminate_session_request

Attribute	Required	Description
id	No	An application-provided transaction identifier.
channel	Yes	The channel identifier returned from the MRCP server. The client uses it to communicate with the server.

Terminate session response format

```
<?xml version="1.0" encoding="UTF-8"?>
  <MediaServerControl version="1.0">
   <response id="1234" request="mrcp_session_terminate"
   code="200" text="OK"/>
  </MediaServerControl>
```

An <error_info> element will be included in the response if the MRCP server returns an error. This enables the MRCP server error code and text, as well as the target URL that generated the error, to be returned to the application.

Sample Call Flow

This section provides a sample call flow of the following MRCP session creation scenario.

Scenario

- After creating a SIP dialog with the media server the application establishes an MRCP session with a speech recognition resource and starts a recognition session via MSCML requests to the IP Media Server.
- The application then requests the media server to begin playing a prompt which is barged by speech input from the user. The MRCP server sends the results of the speech recognition to the application.
- When finished the application terminates the MRCP session and media server session.

The following sequence of interactions assume that the initial dialog between the application server and media server is already established.

Call Flow

1 Create MRCP Session

The application sends an MSCML request to create an MRCP session on an existing dialog using a SIP INFO message. The MSCML request must specify the type of resource needed and must include the URL of the MRCP server to contact. The request may optionally include an MSCML request ID for the purpose of matching the request with its response.

The follow is an example of the MSCML payload:

```
<?xml version="1.0" encoding="UTF-8"?>
   <MediaServerControl version="1.0">
      <mrcp_session_create id="1234"
         resource="speechrecog"
         url="sip:speechresources@server.example.com"
         proto=TCP/MRCPv2″
         rtpdirection=″sendonly″/>
   </MediaServerControl>
```

a. As a result, the media server generates a SIP INVITE to the target address as shown below.

```
INVITE sip:speechresources@server.example.com SIP/2.0
   Via:SIP/2.0/TCP ssmediaserver.example.com:5060;
   branch=z9hG4bK74bf9
   Max-Forwards:6
   To:sip:speechresources@server.example.com
   From:SnowShoreMediaServer
   <sip:ssms@example.com>;tag=1928301774
```

```
Call-ID:a84b4c76e66710
CSeq:314161 INVITE
Contact:<sip:ssms@example.com>
Content-Type:application/sdp
Content-Length: 230

v=0
o=SnowShoreMediaServer 2890844526 2890842808
IN IP4 126.16.64.4
s=-
c=IN IP4 224.2.17.12
m=application 9 TCP/MRCPv2
a=setup:active
a=connection:new
a=resource:speechrecog
a=cmid:1
m=audio 49170 RTP/AVP 0 101
a=rtpmap:0 pcmu/8000
a=sendonly
a=mid:1
```

b. The speech server generates a response.

```
SIP/2.0 200 OK
Via:SIP/2.0/TCP ssmediaserver.example.com:5060;
branch=z9hG4bK74bf9

To:sip:speechresources@server.example.com;tag=a6c85cf
From:SnowShoreMediaServer
   <sip:ssms@example.com>;tag=1928301774
   Call-ID:a84b4c76e66710
   CSeq:314161 INVITE
   Contact:<sip:ssms@example.com>
   Content-Type:application/sdp
   Content-Length: 249

v=0
o=speechresources 2890844526 2890842808 IN IP4
   126.16.64.4
s=-
c=IN IP4 224.2.17.12
m=application 32416 TCP/MRCPv2
a=setup:passive
a=connection:new
a=channel:32AECB234338@speechrecog
a=cmid:1
m=audio 48260 RTP/AVP 00 101
a=rtpmap:0 pcmu/8000
a=recvonly
a=mid:1
```

c. The media server acknowledges the response.

```
ACK sip:mresources@server.example.com SIP/2.0
Via:SIP/2.0/TCP ssmediaserver.example.com:5060;
branch=z9hG4bK74bf9
Max-Forwards:6
To:sip:speechresources@server.example.com;tag=a6c85cf

From:SnowShoreMediaServer
<sip:ssms@example.com>;tag=1928301774
Call-ID:a84b4c76e66710
CSeq:314162 ACK
Content-Length:0
```

1. The media server sends an MSCML response to the <mrcp_session_create> request, as depicted in the following example.

```
<?xml version="1.0" encoding="UTF-8"?>
<MediaServerControl version="1.0">
   <response id="1234" request="mrcp_session_create"
      code="200" text="OK" connect="224.2.17.2"
      port="32416" channel="32AECB234338@speechrecog"/>
</MediaServerControl>
```

Note: The IP Media Server's response must contain the connection and port information for the control channel and the channel identifier returned from the MRCP server. These are required to establish the control channel and identify all future requests for the associated resource.

2. The application initiates the start speech recognition by making an MRCP RECOGNIZE request on the control channel with the correct channel id and grammar to match. Following is an example request:

```
MRCP/2.0 487 RECOGNIZE 32416
   Channel-Identifier:32AECB23433801@speechrecog
   Confidence-Threshold:0.9
   Content-Type:application/srgs+xml
   Content-Id:<request1@form-level.store>
   Content-Length:104

<?xml version="1.0"?>

<!-- the default grammar language is US English -->
<grammar xmlns="http://www.w3.org/2001/06/grammar"
      xml:lang="en-US" version="1.0" root="request">

   <!-- single language attachment to tokens -->
      <rule id="yes">
         <one-of>
            <item xml:lang="fr-CA">oui</item>
```

```
            <item xml:lang="en-US">yes</item>
         </one-of>
      </rule>

   <!-- single language attachment to a rule expansion -->
      <rule id="request">may I speak to
         <one-of xml:lang="fr-CA">
            <item>Alex Michaels</item>
            <item>Bill Fuller</item>
         </one-of>
      </rule>
</grammar
```

3. The MRCP server responds that recognition is in progress.

```
MRCP/2.0 48 32614 200 IN-PROGRESS
   Channel-Identifier:32AECB23433801@speechrecog
```

4. The application requests the media server to play a sequence of prompts by sending a request like the following.

```
<MediaServerControl version="1.0">
  <request>
    <play id="33298">
      <prompt stoponerror="yes"
          baseurl="file:////var/mediaserver/prompts/"
         <audio url="prompt1.wav"/>
         <audio url="prompt2.wav"/>
         <audio url="prompt3.wav"/>
      </prompt>
    </play>
  </request>
</MediaServerControl>
```

5. The MRCP server sends a response when speech input has started.

```
MRCP/2.0 49 START-OF-INPUT 32614 IN-PROGRESS
   Channel-Identifier:32AECB23433801@speechrecog
```

6. The application tells the media server to terminate prompt play to implement barge-in based on speech input. The MSCML request would look like:

```
<?xml version="1.0" encoding="UTF-8"?>
  <MediaServerControl version="1.0">
    <stop id="5678"/>
  </MediaServerControl>
```

7. The IP Media Server sends a response to the <stop> request.

```
<?xml version="1.0" encoding="UTF-8"?>
  <MediaServerControl version="1.0">
    <response id="5678" request="stop" code="200"
        text="OK"/>
  </MediaServerControl>
```

8. The IP Media Server sends a response for the stopped <play> request similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<MediaServerControl version="1.0">
  <response id="33298" request="play" code="200"
     text="OK" reason="stopped" playduration="12620">
  </response>
</MediaServerControl>
```

9. The MRCP server sends a response when recognition has completed with the results of the recognition. The response would look like the following:

```
MRCP/2.0 465 RECOGNITION-COMPLETE 32614 COMPLETE
          Channel-Identifier:32AECB23433801@speechrecog
            Completion-Cause:000 success
            Waveform-
   URI:<http://web.media.com/session123/audio.wav>;
         size=342456;duration=25435
            Content-Type:application/nlsml+xml
            Content-Length:276

<?xml version="1.0"?>
   <result grammar="session:request1@form-level.store">
      <interpretation>
         <instance name="Person">
            <Person>
               <Name> Alex Michaels</Name>
            </Person>
         </instance>
         <input> may I speak to Alex Michaels</input>
      </interpretation>
   </result
```

10. The application issues an MRCP GET-RESULT request to recompute and retrieve the results. Following is an example request.

```
MRCP/2.0 73 GET-RESULT 32614
   Channel-Identifier:32AECB23433801@speechrecog
   Confidence-Threshold:0.9
```

11. The MRCP server responds to the GET-RESULT request with a message similar to the one below.

```
MRCP/2.0 487 32614 200 COMPLETE
   Channel-Identifier:32AECB23433801@speechrecog
   Content-Type:application/nlsml+xml
   Content-Length:276

<?xml version="1.0"?>
<result grammar="session:request1@form-level.store">
   <interpretation>
      <instance name="Person">
         <Person>
            <Name> Alex Michaels </Name>
         </Person>
      </instance>
      <input> may I speak to Alex Michaels </input>
```

```
      </interpretation>
    </result
```

12. The application sends a BYE to the media server which causes the media server to also issue a BYE to the MRCP server. The result is that resources on both servers are freed.

H - T.30 Fax Detection

The Fax Detection feature enables the IP Media Server to detect T.30 Fax CNG tones embedded in a G.711 RTP audio stream, and to notify a VXML application script via a VXML event. The VXML script then transfers the media stream to a target Fax machine for further processing.

The transfer is "hairpinned", and does not go through the IP Media Server conference mixer, which in turn keeps the audio (fax) signals cleaner and does not introduce unnecessary delay.

Note: These enhancements are implemented only in the VoiceXML 1.0 browser.

Please refer to the Release Notes (Release 2.0.5) for important information regarding this feature.

Fax Call Transfer Call Flow

Figure 18 outlines the call flow between the initiating (sending) Fax machine and the terminating (receiving) Fax machine. Each step is explained in the Description that follows the figure.

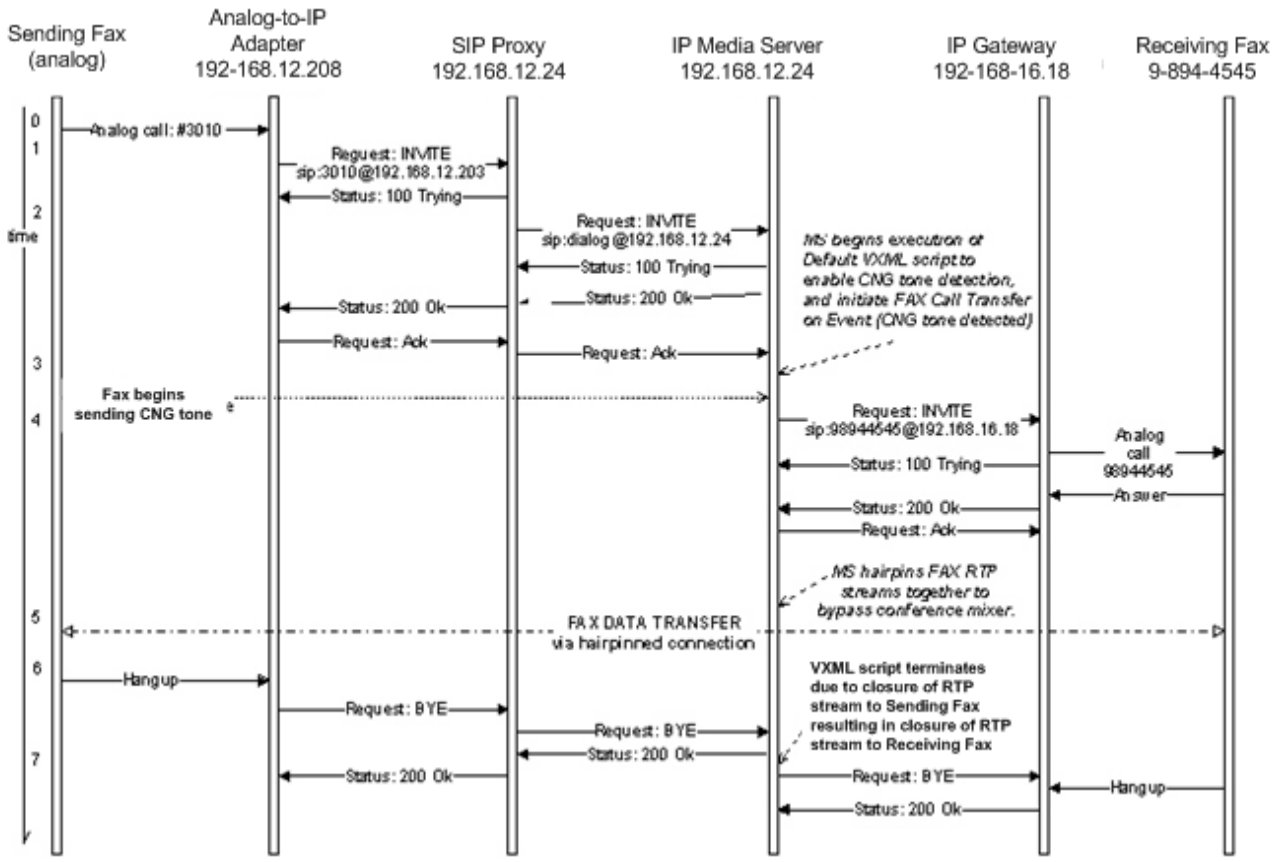

Figure 18. Fax Call Transfer

Description

The following contains a detailed description of the call flow for the Fax Call Transfer scenario.

Time	Description
1	The Sending Fax machine initiates an analog call (dialed number is 3010) to the Analog-to-IP Adapter.
2	The SIP proxy server has been configured to respond to the INVITE 3010 by issuing a SIP INVITE request to the IP Media Server. Note that the INVITE request specifies using the dialog service.
3	A connection is established between the Sending Fax machine and the IP Media Server. The IP Media Server begins executing the default VXML script, which has requested the CNG tone detection. The Sending Fax machine is generating the CNG tone.
4	The CNG tone is detected by the mserv process and a notification event (CNG) is sent to the VXML script. In response to this event, the VXML script performs a Fax call <transfer> specifying bridge=”forward”) This step results in an INVITE 98944545@192.168.16.18 sent to the IP Gateway. The IP Gateway calls the Receiving Fax (via the PBX) at the number 9-894-4545. When the call is established, the VXML script instructs the mserv process to hairpin together the Receiving Fax and the Sending Fax RTP streams.
5	The Sending Fax machine and the Receiving Fax server are connected together via hairpinning RTP streams. The two Fax machines continue to perform the fax data transfer.
6	The Sending Fax completes sending the Fax to the Receiving Fax and hangs up. The Analog-to-IP Adapter issues a SIP BYE request to the SIP Proxy which results in the SIP Proxy sending a BYE request to the IP Media Server.
7	The VXML script terminates when the Sending Fax RTP stream closes which closes the Receiving Fax stream.

CNG Tone Detection and Event Notification

The VXML 1.0 interpreter supports the ability to enable and disable CNG tone detection, and to handle event notification (CNG-tone-detected events). The VXML application (or VXML script) employs this capability to handle incoming Fax calls.

VoiceXML 1.0 Implementation

The property "***com.snowshore.signal***" enables detection of various signals on the IP Media Server platform when the property is in scope in the VoiceXML interpreter.

Note: The property has a value string that allows the scriptwriter to specify which signal or signals that should be detected. The only signal type implemented at this time is Fax CNG tone. For example,

```
<property name="com.snowshore.signal" value="CNG"/>
```

enables detection of CNG tone. Multiple signals could be enabled by placing them all in the "value" attribute, separated by commas.

When the CNG tone signal is detected by the IP Media Server platform, the VoiceXML interpreter throws the "***com.snowshore.signal***" event. The VXML script should handle this event and then disable signal detection. The "_message" variable is set to the type of signal, and can be detected as shown in the following example:

```
<catch event="com.snowshore.signal">
   <if cond="_message == 'CNG'">
      <!—Perform Fax Call Transfer when CNG tone is detected -->
   </if>
```

Note that `<catch event=>` does not block; events are handled in an asynchronous manner, which permits other operations to take place.

In order to disable CNG tone detection, the application must clear the ***"com.snowshore.signal"*** property in the correct scope. For example,

```
<property name="com.snowshore.signal" value="OFF"/>
```

The following list describes the interaction of CNG tone detection with other VXML input collection modes:

- If there is a DTMF recognition session, it is terminated and any digits collected up to that point are dropped.
- If a prompt that allows barge-in is being played, it is stopped.

The following list describes the VXML interpreter's actions when it is not in an input collection mode:

- If the prompt does not allow barge-in, the VXML interpreter processes the event after the prompt completes.
- If the VXML interpreter is in a transition state that does not require the collection of input, the event is left queued until the VXML interpreter enters the input state.

VoiceXML Properties

com.snowshore.signal

Enables detection of various signals on the IP Media Server platform when the property is in scope in the VoiceXML interpreter.

Note: The property has a value string that allows the scriptwriter to specify which signal or signals that should be detected. The only signal type implemented at this time is Fax CNG tone. For example,

```
<property name="com.snowshore.signal" value="CNG"/>
```

enables detection of CNG tone. Multiple signals could be enabled by placing them all in the "value" attribute, separated by commas.

When the CNG tone signal is detected by the IP Media Server platform, the VoiceXML interpreter throws the "***com.snowshore.signal***" event. The VXML script should handle this event and then disable signal detection. The "_message" variable is set to the type of signal, and can be detected as shown in the following example:

```
<catch event="com.snowshore.signal">
   <if cond="_message == 'CNG'">
      <!—Perform Fax Call Transfer when CNG tone is detected -->
   </if>
```

Note that `<catch event=>` does not block; events are handled in an asynchronous manner, which permits other operations to take place.

Fax Detection—Example Script

The following example illustrates the syntax to enable CNG tone detection and to catch the tone-detected event.

Welcome.vxml:

```
<?xml version="1.0" encoding="UTF-8"?>

<vxml version="1.0">

<property name="universals" value="help"/>
<property name="timeout" value="3000"/>
<property name="com.snowshore.signal" value="CNG"/>

<form id="f1">

  <field name="F_1">
    <dtmf> [1-3]
    </dtmf>
    <prompt bargein="false">
       <audio
   src="http://localhost/snowshore/prompts/gmvoices/en
   _US/susan/msg040"/>
    </prompt>

    <help count="1" cond="dtmf">
     <prompt> since the condition is set to false, this
   handler will be skipped
        <audio
   src="http://localhost/snowshore/prompts/gmvoices/en
   _US/susan/msg060"/>
      </prompt>
    </help>

    <noinput count="1" cond="true">
      <prompt> this is the first occurrence of the help
   event. say help again.</prompt>
    </noinput>

    <noinput count="2" cond="true">
     <prompt> this is the second occurrence of the help
   event. Do something!!!.</prompt>
    </noinput>

    <noinput count="3" cond="true">
      <prompt> this is the third occurrence of the help
   event. goodbye.</prompt>
      <disconnect/>
    </noinput>

    <catch event="com.snowshore.signal">
```

```
        <prompt> GOT THE com.snowshore.signal
      event.</prompt>
        <if cond="_message=='CNG'">
           <prompt> Found CNG message. GOTO fax transfer
      support</prompt>
            <goto next='transfer_call.xml'/>
        </if>
        <disconnect/>
        <prompt> goodbye.</prompt>
       </catch>

    </field>

    <filled>
      <prompt>why did you fill the field?</prompt>
    </filled>

  </form>
```

Fax Call Transfer

The VXML script can perform a call transfer (using the VXML <transfer> element). The transfer initiates a call to the Fax termination server (to which the Fax is sent). When the call is successful (i.e., connection has been established between the IP Media Server and the Fax termination server), the RTP streams are joined to a conference.

The VoiceXML 1.0 specification defines the <transfer> element for establishing a second call leg and allowing it to interact with the initial leg. The IP Media Server extends the <transfer> element by adding a new value "forward" to the `bridge` attribute. The `forward` bridge mode causes the two RTP streams to be "hairpinned" together. Hairpinning prevents the audio data from being mixed within the conference mixer.

Note: Only two RTP streams can be hairpinned together, and they must employ the same encoding type (otherwise end-to-end communication is not possible).

Call Transfer—Example Script

The following example script illustrates the use of Fax Call Transfer (i.e., hairpinning the RTP streams together) and call termination.

Transfer_call.xml:

```
<?xml version="1.0"?>
<!-- 10.10.10.111 is the SnowShore MPS -->
<!-- 1234567@192.169.12.207 is a valid SIP destination-
   ->
<!-- of the receiving Fax server. -->
<!-- 1234567 is associated with techsupport@cantata.com
   -->

<!-- This script first plays "star", followed by "One"
   -->
<!-- Then attempts to do the transfer. If the transfer
   is   -->
<!-- success, when the call is ended, you'll hear the
   reason-->
<!-- number (see below in the script) followed by the
   number of-->
<!-- seconds the call is connected between the caller
   and   -->
<!-- the callee. -->

<vxml version="1.0">
<form name="genericTransfer">
   <block>
```

```
    <audio
src="http://localhost/snowshore/prompts/gmvoices/en
_US/susan/star.neu" />
</block>
<var name="fax2emailURI"
expr="session.telephone.dnis + '@192.168.12.207'"/>
<var name="subscriber" expr="'sip:' +
session.telephone.dnis + '@cantata.com'"/>
<transfer name="makethecall" destexpr="subscriber"
requriexpr="fax2emailURI" connecttimeout="10s"
bridge="forward">
  <prompt>
      <audio
src="http://localhost/snowshore/prompts/gmvoices/en
_US/susan/1.ris" />
  </prompt>
  <filled>
    <if cond="makethecall == 'busy'">
      <prompt>
        <audio
src="http://localhost/snowshore/prompts/gmvoices/en
_US/susan/2.ris" />
      </prompt>
    <elseif cond="makethecall == 'noanswer'"/>
      <prompt>
        <audio
src="http://localhost/snowshore/prompts/gmvoices/en
_US/susan/3.ris" />
      </prompt>
    <elseif cond="makethecall ==
'near_end_disconnect'"/>
      <prompt>
        <audio
src="http://localhost/snowshore/prompts/gmvoices/en
_US/susan/4.ris" />
      </prompt>
    <elseif cond="makethecall ==
'far_end_disconnect'"/>
      <prompt>
        <audio
src="http://localhost/snowshore/prompts/gmvoices/en
_US/susan/5.ris" />
      </prompt>
    <elseif cond="makethecall ==
'network_disconnect'"/>
      <prompt>
        <audio
src="http://localhost/snowshore/prompts/gmvoices/en
_US/susan/6.ris" />
      </prompt>
    <elseif cond="makethecall == 'network_busy'"/>
      <prompt>
```

```
            <audio
      src="http://localhost/snowshore/prompts/gmvoices/en
      _US/susan/7.ris" />
          </prompt>
        <elseif />
          <prompt>
            <value class="number"
      expr="makethecall$.duration" />
            <audio
      src="http://localhost/snowshore/prompts/gmvoices/en
      _US/susan/seconds.neu" />
          </prompt>
        </if>
      </filled>
    </transfer>
    <block>
      <audio
    src="http://localhost/snowshore/prompts/gmvoices/en
    _US/susan/star.neu" />
    </block>
  </form>
  </vxml>
```

I - T.38 Fax Detection, Termination, and Initiation

This chapter explains the Fax Detection, Termination, and Initiation feature.

The fax termination server is incorporated into the IP Media Server using the Dialogic® Brooktrout® T.38 host fax termination software. Refer to the *Dialogic IP Media Server Installation and Operations Guide* for the instructions to configure the fax software.

Detection and Termination

This feature enables the Dialogic® IP Media Server to detect CNG Fax Tones embedded in a G.711 RTP audio stream, and notify a VXML application script via a VXML event. The IP Media Server then initiates a SIP Re-INVITE method to the endpoint requesting a switch from the current voice coder mode to T.38 fax relay mode. Upon the completion of the SIP Re-INVITE method, the Dialogic® IP Media Server will record the fax transmission to a TIFF file stored locally on the IP Media Server and optionally convert the file to PDF format.

Initiation (Email to Fax)

This feature enables the IP Media Server to initiate a T.38 Fax Transmission of a file received as an attachment in an SMTP request.

Note: This feature is currently supported in the VoiceXML 1.0 Interpreter.

Users can send an email to the IP Media Server in the following format:

```
user@phonenumber.fax
```

Once The IP Media Server receives the email it parses the email into components. The IP Media Server can parse the following formats and convert them to TIFF format:

- JPG
- GIF
- TIFF
- PDF
- Text

One the IP Media Server converts the email to TIFF format, it saves it with a unique filename in the `/tmp` directory.

The IP Media Server notifies the T.38 application about the new fax file to transmit. The T.38 application accepts the destination phone number and the filename as input. The T.38 API transmits the fax and returns a status once the transmission is complete.

Snowshore.cfg Parameters

You must configure the following parameters in the snowshore.cfg file to enable this feature.

Table 62. T.38 Fax Parameters

Parameter	Description	Default (in Decimal)	Range
SR140APP_PORT	First port for the T.38 fax server	9000	9000-40000
SR140APP_PROXY_PORT	Host name for proxy fax SIP Server	5060	
SR140APP_PROXY_IP	Required field for sending fax	No default	

Fax Call Termination Call Flow

Figure 19 outlines the call flow between the initiating (sending) fax server and the terminating (receiving) fax server. Each step is explained in the Description that follows the figure.

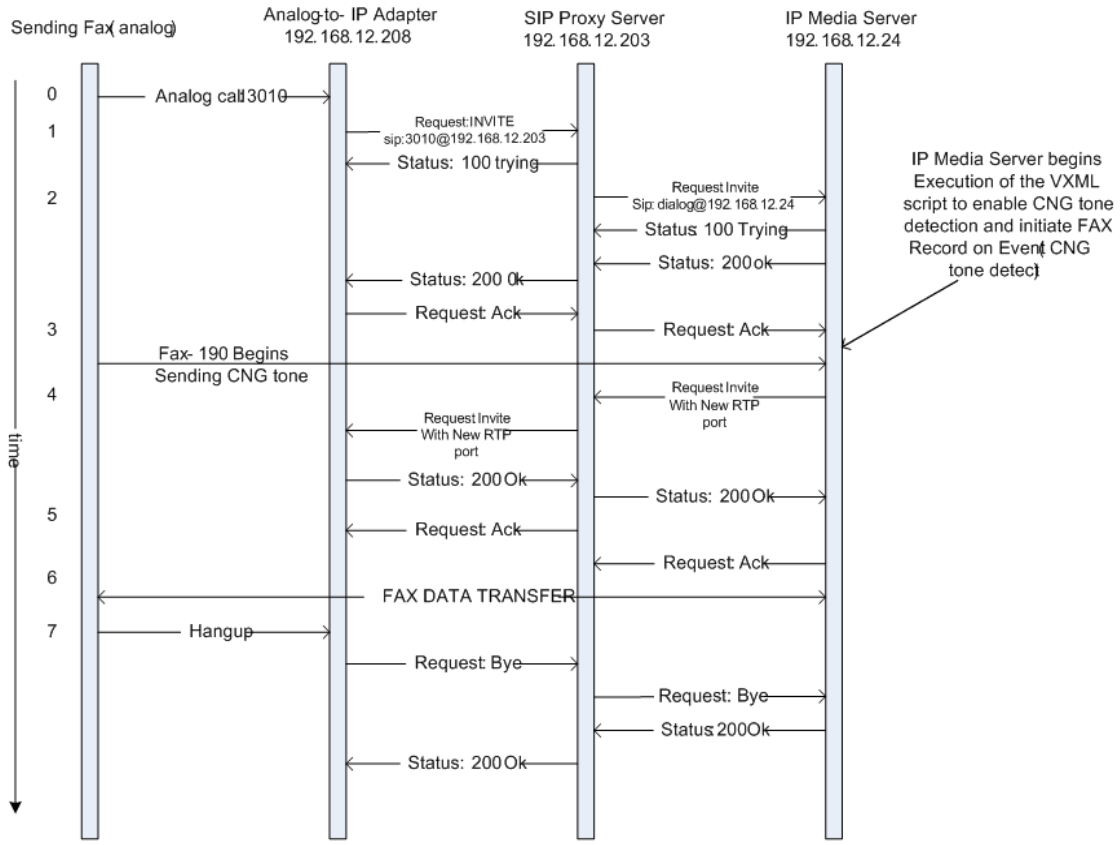

Figure 19. Fax Call Transfer

Description

The following is a detailed description of the call flow for the Fax Call Transfer scenario.

Time	Description
1	The Sending Fax server initiates an analog call (dialed number is 3010) to the Analog-to-IP Adapter.
2	The Analog-to IP Adapter accepts the analog call and issues a SIP INVITE request to the SIP Server it is configured to use (for example, the SIP Proxy Server @192.168.12.203).
3	The SIP Proxy Server is configured to respond to the INVITE 3010 by issuing a SIP INVITE request to the IP Media Server. The INVITE request specifies the use of the dialog service.
4	A connection is established between the Sending Fax server and the IP Media Server. The Media Server begins execution of the default VXML script which has requested CNG tone detection. The Sending Fax server generates a CNG tone.
5	The CNG tone is detected by the mserv application and a notification event (CNG detected) is sent to the VXML script. In response to the event, the VXML script performs a fax record <record> (specifying dest_enc="tiff"). This results in a RE-INVITE request sent to 192.168.12.203 with the new RTP port for the fax data.
6	The Sending Fax server starts sending fax data to the T.38 Host Server RTP port to be terminated. The two fax systems continue to perform the fax data transfer.
7	The VXML script sends the fax tiff file to a file server.

CNG Tone Detection and Event Notification

The VXML 1.0 interpreter supports the ability to enable and disable CNG tone detection, and to handle event notification (CNG-tone-detected events). The VXML application (or VXML script) employs this capability to handle incoming Fax calls.

The mserv application enables and disables the CNG tone detection through the report Media Server Protocol (MSP) command's Call Progress Analysis parameters. In addition, the mserv application provides event notification back to the VXML 1.0 browser / application.

VoiceXML 1.0 Implementation

The property "***com.snowshore.signal***" enables detection of various signals on the IP Media Server platform when the property is in scope in the VoiceXML interpreter.

Note: The property has a value string that allows the scriptwriter to specify which signal or signals that should be detected. The only signal type implemented in this release is Fax CNG tone. For example:

```
<property name="com.snowshore.signal" value="CNG"/>
```

This example enables detection of CNG tone. Multiple signals could be enabled by placing them all in the "value" attribute, separated by commas.

When the CNG tone signal is detected by the IP Media Server platform, the VoiceXML interpreter throws the "*com.snowshore.signal*" event. The VXML script should handle this event and then disable signal detection. The "_message" variable is set to the type of signal, and can be detected as shown in the following example:

```
<catch event="com.snowshore.signal">
   <if cond="_message == 'CNG'">
      <!—Perform Fax Call Transfer when CNG tone is detected -->
   </if>
```

Note that `<catch event=>` does not block; events are handled in an asynchronous manner, which permits other operations to take place.

In order to disable CNG tone detection, the application must clear the *"com.snowshore.signal"* property in the correct scope. For example,

```
<property name="com.snowshore.signal" value="OFF"/>
```

The following list describes the interaction of CNG tone detection with other VXML input collection modes:

- If there is a DTMF recognition session, it is terminated and any digits collected up to that point are dropped.
- If a prompt that allows barge-in is being played, it is stopped.

The following list describes the VXML interpreter's actions when it is not in an input collection mode:

- If the prompt does not allow barge-in, the VXML interpreter processes the event after the prompt completes.
- If the VXML interpreter is in a transition state that does not require the collection of input, the event is left queued until the VXML interpreter enters the input state.

Fax Detection—Example VXML Script

The example below shows the syntax to enable CNG tone detection and catch the event.

```
<form id="f1">
  <field name="field1">
    <property name="com.snowshore.signal" value="CNG"/>
    <noinput>
     <!-- sets the value of this field, so that the FIA
   doesn't select this again -->
      <assign name="field1" expr="true"/>
    </noinput>
    <catch event="com.snowshore.signal">
       <if cond="_message == 'CNG'">
       <!-- do stuff when CNG is detected -->
    </if>
    </catch>
    <!-- This field waits for the CNG tone -->
    .....
  </field>
  <field name="Voicemail"/>
    <!-- When this field is visited, CNG is not enabled
   (the property is not set) -->
    <prompt> Welcome to VoiceMail.  To leave a message
   ... </prompt>
    ....
  </field>
</form>
```

VXML Fax Record

The VXML script can perform a fax record (using the VXML <record> element with the mimetype "image/tiff"). The record will initiate the media sever to redirect the incoming media to the T.38 fax termination server (the fax will be recorded as a tiff file).

The Fax Record feature is supported by the VoiceXML 1.0 browser through enhancements to the <record> element mimetype "image/tiff" and/or “image/pdf”

Call Record - Example VXML Script

This section contains a sample VoiceXML 1.0 FAX Call Record script. The script demonstrates use of FAX Call Record and HTTP post of the file.

```
<vxml version="1.0">
<var name="recordedMedia"/>

<form id="recordMessage">
   <record name="deposit" beep="true" type="image/tiff"
   >
       <filled>
          <assign name="recordedMedia" expr="deposit"/>
          <goto next=" #deliverMessage"/>
       </filled>
    </record>
</form>

<form id="deliverMessage">
    <block>
<submit method="post" namelist="recordedMedia"
   next="http://localhost/snowshore/htdocs/depositfax.
   cgi?ani='session.telephone.ani'"/>
    </block>
</form>
```

Index

B

C

D

E

F

G

H

I

J

K

L

M

N

P

Q

R

S

T

U

V

W