Copyright and Legal Notice

Copyright © 2000-2011 Dialogic Inc. All Rights Reserved. You may not reproduce this document in whole or in part without permission in writing from Dialogic Inc. at the address provided below.

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Inc. and its affiliates or subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies 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 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 Inc. 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 may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does 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 and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 926 Rock Avenue, San Jose, California 95131 USA. Dialogic 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.

Dialogic, Dialogic Pro, Dialogic Blue, Veraz, Brooktrout, Diva, Diva ISDN, Making Innovation Thrive, Video is the New Voice, Diastar, Cantata, TruFax, SwitchKit, SnowShore, Eicon, Eicon Networks, NMS Communications, NMS (stylized), Eiconcard, SIPcontrol, TrustedVideo, Exnet, EXS, Connecting to Growth, Fusion, Vision, PowerMedia, PacketMedia, BorderNet, inCloud9, I-Gate, Hi-Gate, NaturalAccess, NaturalCallControl, NaturalConference, NaturalFax and Shiva,
among others as well as related logos, are either registered trademarks or trademarks of Dialogic Inc. and its affiliates or subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 926 Rock Avenue, San Jose, California 95131 USA. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

The names of actual companies and products mentioned herein are the trademarks of their respective owners.
Hardware Limited Warranty

Please refer to the following Dialogic web site for information on hardware warranty information, which applies unless different terms have been agreed to in a signed agreement between yourself and Dialogic Corporation or its subsidiaries. The listed hardware warranty periods and terms are subject to change without notice. For purchases not made directly from Dialogic please contact your direct vendor in connection with the warranty period and terms that they offer.

http://www.dialogic.com/warranties
Contents

About this Publication ........................................... 20
  Using this Publication ........................................ 21
  Audience and Purpose ....................................... 21
  Documentation Set ........................................... 21
  Document Conventions ..................................... 21
    Notes, Cautions, and Warnings ........................... 21
    Links in PDF .............................................. 22
  Contacting Dialogic Technical Services and Support .... 23
  Ordering Licenses ............................................ 23

1 - Introduction .................................................. 24
  Control Protocols and Services ............................ 25
  Call Control Protocols ..................................... 26
    Session Initiation Protocol (SIP) ....................... 27
      Session ................................................ 27
      Service Indicators .................................... 27
    Session Description Protocol (SDP) .................... 27
    Media Server Control Markup Language (MSCML) ....... 28
    SIP Methods with MSCML ................................ 28
  VoiceXML .................................................... 29
  Media Storage, Processing, and Supported Codecs ....... 30
    Audio .................................................... 30
      Audio Content Storage ................................... 32
    Video ..................................................... 32
    Mixed Audio/Video ....................................... 32
      WAV-based ............................................... 32
      3GPP and 3GPP2 .......................................... 33
      Mixed Audio/Video Content Storage .................... 33
  File Storage and Retrieval ................................ 33
    Network File System (NFS) ................................ 33
    HTTP ...................................................... 33
    Real Time Streaming Protocol (RTSP) .................... 34
  DTMF ......................................................... 34
  In-Band Busy Tone Detection and Reporting ............... 34
  Services ..................................................... 35
    Network Announcements ................................... 35
2 - Session Initiation Protocol (SIP) ........................................................................ 44
Conformance ........................................................................................................... 45
Media Server Availability ....................................................................................... 46
SIP Description ........................................................................................................ 47
   SIP User Agents .................................................................................................. 47
   SIP Message ....................................................................................................... 47
   SIP Transaction ................................................................................................. 47
   URLs .................................................................................................................. 48
   Service Indicators .............................................................................................. 48
   SIP Message Body Types ................................................................................... 49
SIP Requests ............................................................................................................ 50
   Request Line ....................................................................................................... 50
   Headers .............................................................................................................. 51
   Session Descriptor ............................................................................................. 51
SIP Methods ............................................................................................................. 52
   INVITE .............................................................................................................. 52
      Headers in an INVITE ...................................................................................... 53
   ACK ..................................................................................................................... 54
      Headers in an ACK ......................................................................................... 54
   CANCEL .............................................................................................................. 54
      Headers in CANCEL ....................................................................................... 55
   OPTIONS ............................................................................................................ 56
      Headers in OPTIONS ...................................................................................... 56
   BYE ..................................................................................................................... 57
      Headers in BYE ................................................................................................ 57
   INFO ................................................................................................................... 58
      Headers in INFO ............................................................................................. 58
   PRACK ............................................................................................................... 59
      Headers in a PRACK ....................................................................................... 59
SIP Headers ............................................................................................................. 60
   Format and Syntax ............................................................................................. 61
Supported Headers ................................................................................................. 61
   Accept .................................................................................................................. 61
   Call-ID (i) ............................................................................................................ 62
   Contact (m) ........................................................................................................ 62
   Content-Length (l) .............................................................................................. 62
   Content-Type (c) ............................................................................................... 63
   CSeq .................................................................................................................... 63
   From (f) .............................................................................................................. 63
   Max-Forwards ..................................................................................................... 63
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>How SIP Manages Conferences</td>
<td>101</td>
</tr>
<tr>
<td>Creating a Simple Conference</td>
<td>101</td>
</tr>
<tr>
<td>Adding a Participant to a Simple Conference</td>
<td>101</td>
</tr>
<tr>
<td>Ending a Simple Conference</td>
<td>101</td>
</tr>
<tr>
<td>Attributes for Simple Conferences and Participants</td>
<td>102</td>
</tr>
<tr>
<td>DTMF Clamping</td>
<td>102</td>
</tr>
<tr>
<td>Managing Video Switching</td>
<td>102</td>
</tr>
<tr>
<td>Call Flow and Sample Code Examples</td>
<td>104</td>
</tr>
<tr>
<td>Code for Creating a Simple Conference</td>
<td>105</td>
</tr>
<tr>
<td>Advanced Conferencing</td>
<td>106</td>
</tr>
<tr>
<td>Using MSCML for Advanced Conferencing</td>
<td>106</td>
</tr>
<tr>
<td>&lt;configure_conference&gt;</td>
<td>106</td>
</tr>
<tr>
<td>&lt;configure_leg&gt;</td>
<td>106</td>
</tr>
<tr>
<td>MSCML Attributes and Elements for &lt;configure_conference&gt;</td>
<td>106</td>
</tr>
<tr>
<td>MSCML Attributes and Elements for &lt;configure_leg&gt;</td>
<td>107</td>
</tr>
<tr>
<td>MSCML Attributes for &lt;managecontent&gt;</td>
<td>109</td>
</tr>
<tr>
<td>&lt;managecontent&gt; Examples</td>
<td>110</td>
</tr>
<tr>
<td>Creating an Advanced Conference</td>
<td>111</td>
</tr>
<tr>
<td>Modifying an Advanced Conference</td>
<td>112</td>
</tr>
<tr>
<td>Ending an Advanced Conference</td>
<td>113</td>
</tr>
<tr>
<td>Joining Participants (Legs) to an Advanced Conference</td>
<td>113</td>
</tr>
<tr>
<td>Modifying a Conference Participant</td>
<td>114</td>
</tr>
<tr>
<td>Removing Participants From a Conference</td>
<td>114</td>
</tr>
<tr>
<td>Conference Subsetting</td>
<td>114</td>
</tr>
<tr>
<td>Active Talker Events</td>
<td>115</td>
</tr>
<tr>
<td>IVR Operations during a Conference</td>
<td>116</td>
</tr>
<tr>
<td>Playing and Recording Within the Entire Conference</td>
<td>116</td>
</tr>
<tr>
<td>Playing to the Conference</td>
<td>117</td>
</tr>
<tr>
<td>Recording the Conference Output</td>
<td>117</td>
</tr>
<tr>
<td>Video Conferencing Enhancements</td>
<td>118</td>
</tr>
<tr>
<td>MSCML Changes</td>
<td>118</td>
</tr>
<tr>
<td>IVR Operations on Participant Legs</td>
<td>118</td>
</tr>
<tr>
<td>Detecting DTMF Digits On A Conference Leg</td>
<td>118</td>
</tr>
<tr>
<td>Playing Audio to a Participant Leg</td>
<td>121</td>
</tr>
<tr>
<td>Detecting and Reporting Busy Call Progress Tones in MSCML</td>
<td>121</td>
</tr>
<tr>
<td>Simultaneous Play and Record</td>
<td>124</td>
</tr>
<tr>
<td>Creating an Internal Conference Leg</td>
<td>124</td>
</tr>
<tr>
<td>Recording a Conference</td>
<td>124</td>
</tr>
<tr>
<td>MSCML Conferencing Requests</td>
<td>125</td>
</tr>
<tr>
<td>Conferencing Request Elements and Attributes</td>
<td>125</td>
</tr>
<tr>
<td>configure_conference</td>
<td>125</td>
</tr>
<tr>
<td>In INFO Message</td>
<td>126</td>
</tr>
<tr>
<td>configure_leg</td>
<td>127</td>
</tr>
<tr>
<td>Coached Conferencing</td>
<td>129</td>
</tr>
<tr>
<td>Overview</td>
<td>129</td>
</tr>
<tr>
<td>MSCML Elements and Attributes of Coached Conferencing</td>
<td>130</td>
</tr>
<tr>
<td>configure_team</td>
<td>130</td>
</tr>
<tr>
<td>Configuring a Coached Conference</td>
<td>131</td>
</tr>
<tr>
<td>Creating the Conference</td>
<td>131</td>
</tr>
<tr>
<td>Joining and Configuring the Coach</td>
<td>132</td>
</tr>
<tr>
<td>Joining and Configuring the Agent</td>
<td>132</td>
</tr>
<tr>
<td>Joining and Configuring the Client</td>
<td>133</td>
</tr>
<tr>
<td>Supervisor Query for Number of Team Members</td>
<td>134</td>
</tr>
<tr>
<td>Exiting the Conference</td>
<td>134</td>
</tr>
<tr>
<td>Using SIP INFO</td>
<td>135</td>
</tr>
<tr>
<td>MSCML Conferencing Reference</td>
<td>136</td>
</tr>
<tr>
<td>MSCML Elements</td>
<td>136</td>
</tr>
<tr>
<td>Prompt Elements</td>
<td>163</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----</td>
</tr>
<tr>
<td>IVR Attributes</td>
<td>163</td>
</tr>
<tr>
<td>barge</td>
<td>163</td>
</tr>
<tr>
<td>baseurl</td>
<td>164</td>
</tr>
<tr>
<td>beep</td>
<td>164</td>
</tr>
<tr>
<td>cleardigits</td>
<td>164</td>
</tr>
<tr>
<td>delay</td>
<td>164</td>
</tr>
<tr>
<td>duration</td>
<td>164</td>
</tr>
<tr>
<td>encoding</td>
<td>165</td>
</tr>
<tr>
<td>endsilence</td>
<td>165</td>
</tr>
<tr>
<td>endwarning</td>
<td>165</td>
</tr>
<tr>
<td>escapekey</td>
<td>165</td>
</tr>
<tr>
<td>extradigittimer</td>
<td>166</td>
</tr>
<tr>
<td>firstdigittimer</td>
<td>166</td>
</tr>
<tr>
<td>id</td>
<td>166</td>
</tr>
<tr>
<td>interdigittimer</td>
<td>166</td>
</tr>
<tr>
<td>initsilence</td>
<td>166</td>
</tr>
<tr>
<td>locale</td>
<td>167</td>
</tr>
<tr>
<td>maskdigits</td>
<td>167</td>
</tr>
<tr>
<td>maxdigits</td>
<td>168</td>
</tr>
<tr>
<td>mode</td>
<td>168</td>
</tr>
<tr>
<td>offset</td>
<td>168</td>
</tr>
<tr>
<td>promptencoding</td>
<td>168</td>
</tr>
<tr>
<td>recstopmask</td>
<td>169</td>
</tr>
<tr>
<td>recurl</td>
<td>169</td>
</tr>
<tr>
<td>recencoding</td>
<td>169</td>
</tr>
<tr>
<td>repeat</td>
<td>169</td>
</tr>
<tr>
<td>report</td>
<td>170</td>
</tr>
<tr>
<td>returnkey</td>
<td>170</td>
</tr>
<tr>
<td>stop_on_error</td>
<td>170</td>
</tr>
<tr>
<td>subtype</td>
<td>171</td>
</tr>
<tr>
<td>type</td>
<td>171</td>
</tr>
<tr>
<td>url</td>
<td>172</td>
</tr>
<tr>
<td>value</td>
<td>172</td>
</tr>
<tr>
<td>warnurl</td>
<td>172</td>
</tr>
<tr>
<td>IVR Response Elements and Attributes</td>
<td>173</td>
</tr>
<tr>
<td>Response Elements</td>
<td>173</td>
</tr>
<tr>
<td>error_info</td>
<td>173</td>
</tr>
<tr>
<td>id</td>
<td>173</td>
</tr>
<tr>
<td>Response Attributes</td>
<td>174</td>
</tr>
<tr>
<td>code</td>
<td>174</td>
</tr>
<tr>
<td>digits</td>
<td>174</td>
</tr>
<tr>
<td>playduration</td>
<td>174</td>
</tr>
<tr>
<td>reason</td>
<td>174</td>
</tr>
<tr>
<td>reclang</td>
<td>175</td>
</tr>
<tr>
<td>text</td>
<td>175</td>
</tr>
</tbody>
</table>

6 - VoiceXML Version 2.0 and Dialog Service ........................................... 176

About VoiceXML ........................................... 177
VoiceXML Interpreter .................................... 177
Dialog Service Indicator and Request URI .................. 177
VoiceXML Launcher ........................................ 178
Default Script .......................................... 178
VoiceXML Concepts ...................................... 178
Syntax .................................................. 178
Scope ................................................... 178
Resource Fetching .......................................................... 179
VoiceXML Application and Its Documents .......................... 179
Dialogs ............................................................................. 179
Scripting .......................................................................... 180
DTMF Grammars Guidelines ............................................. 180
<grammar> ...................................................................... 180
<tag> ................................................................................. 181
<ruleref> ......................................................................... 181
<rule> .............................................................................. 181
MRCP ................................................................................ 181
Session Variables ............................................................. 181
File Storage and Retrieval ................................................. 182
Media Content Recovery Extension ................................. 182
VoiceXML Elements Reference ......................................... 183
<assign> ........................................................................... 183
<audio> .......................................................................... 183
<brick> .............................................................................. 184
<catch> ............................................................................ 184
<choice> ........................................................................... 185
<clear> ............................................................................. 186
<content> ........................................................................ 187
<data> ............................................................................. 188
<disconnect> ................................................................... 189
<else> .............................................................................. 189
<elseif> ......................................................................... 189
<enumerate> ................................................................... 189
<error> ........................................................................... 190
<exit> ............................................................................... 190
<field> ............................................................................ 190
<filled> ........................................................................... 191
<font> ............................................................................. 192
<form> ............................................................................ 193
<goto> ............................................................................. 193
<grammar> .................................................................... 194
<help> ............................................................................ 196
<if> .................................................................................. 196
<initial> ......................................................................... 197
<link> ............................................................................ 197
<log> ............................................................................... 197
<menu> ............................................................................. 199
<meta> ............................................................................ 200
<noinput> ....................................................................... 200
<nomatch> ...................................................................... 201
<object> .......................................................................... 201
<option> ........................................................................... 201
<overlay> ........................................................................ 202
<param> .......................................................................... 203
<prompt> .......................................................................... 204
<property> ..................................................................... 205
<record> .......................................................................... 205
<reprompt> ...................................................................... 206
$return> ............................................................................ 206
<say-as> .......................................................................... 207
<script> ........................................................................... 208
<scroll> .......................................................................... 208
<subdialog> ..................................................................... 209
<submit> .......................................................................... 210

Application Developer's Guide
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>VoiceXML Properties</td>
<td>217</td>
</tr>
<tr>
<td>com.snowshore.criticaldigit_timer</td>
<td>217</td>
</tr>
<tr>
<td>MIME Types Supported</td>
<td>218</td>
</tr>
<tr>
<td>ECMAScript Functionality</td>
<td>220</td>
</tr>
<tr>
<td>VoiceXML Extended Session Variables</td>
<td>223</td>
</tr>
<tr>
<td>VoiceXML 2.0 Recommendations</td>
<td>223</td>
</tr>
<tr>
<td>Dialogic® Extensions</td>
<td>223</td>
</tr>
<tr>
<td>Example</td>
<td>224</td>
</tr>
<tr>
<td>RTP Codec Selection Using the <code>&lt;transfer&gt;</code> Element</td>
<td>226</td>
</tr>
<tr>
<td>Image Overlay and Text Overlay Functionality</td>
<td>227</td>
</tr>
<tr>
<td>Overlay Overview</td>
<td>227</td>
</tr>
<tr>
<td>Characteristics of Image Overlay</td>
<td>227</td>
</tr>
<tr>
<td>Characteristics of Text Overlay</td>
<td>228</td>
</tr>
<tr>
<td>Text Overlay Restrictions</td>
<td>228</td>
</tr>
<tr>
<td>Installing Font Files</td>
<td>228</td>
</tr>
<tr>
<td>Dynamic Fonts</td>
<td>229</td>
</tr>
<tr>
<td>Cached Fonts</td>
<td>229</td>
</tr>
<tr>
<td>trcr.cfg Description</td>
<td>229</td>
</tr>
<tr>
<td>Sample trcr.cfg</td>
<td>231</td>
</tr>
<tr>
<td>Overlay Demos</td>
<td>231</td>
</tr>
<tr>
<td>7 - Sample Code and Call Flows</td>
<td>232</td>
</tr>
<tr>
<td>Announcements Sample Code and Call Flows</td>
<td>233</td>
</tr>
<tr>
<td>Play an Announcement as Early Media</td>
<td>233</td>
</tr>
<tr>
<td>Call Flow for an Early Media Announcement</td>
<td>235</td>
</tr>
<tr>
<td>Playing an Announcement as Normal Media</td>
<td>236</td>
</tr>
<tr>
<td>Call Flow for a Normal Media Announcement</td>
<td>238</td>
</tr>
<tr>
<td>Stopping Media—Hold</td>
<td>239</td>
</tr>
<tr>
<td>Call Flow for Stopping Media—Hold</td>
<td>241</td>
</tr>
<tr>
<td>Repeating the Audio</td>
<td>242</td>
</tr>
<tr>
<td>Conferences Sample Code and Call Flows</td>
<td>243</td>
</tr>
<tr>
<td>Creating a Simple Conference</td>
<td>243</td>
</tr>
<tr>
<td>Call Flow for a Simple Conference (Normal Media)</td>
<td>244</td>
</tr>
<tr>
<td>Creating an Advanced Conference</td>
<td>245</td>
</tr>
<tr>
<td>Call Flow to Set up an Advanced Conference</td>
<td>247</td>
</tr>
<tr>
<td>Modifying Conference Using Subscribe</td>
<td>248</td>
</tr>
<tr>
<td>Providing Communication for Participant in an Advanced Conference</td>
<td>249</td>
</tr>
<tr>
<td>Joining a Participant Using Special Attributes</td>
<td>250</td>
</tr>
<tr>
<td>Suspending Communications within a Conference</td>
<td>251</td>
</tr>
<tr>
<td>Response to Mute a Conference Participant</td>
<td>252</td>
</tr>
<tr>
<td>Playing Audio to Conference Participant</td>
<td>253</td>
</tr>
<tr>
<td>Changing Mixmode to Parked</td>
<td>253</td>
</tr>
<tr>
<td>Response to Parked</td>
<td>254</td>
</tr>
<tr>
<td>Playing the Audio</td>
<td>254</td>
</tr>
<tr>
<td>Response to Message to Play Audio</td>
<td>255</td>
</tr>
<tr>
<td>Changing Participant Mixmode Back to Full</td>
<td>255</td>
</tr>
<tr>
<td>Response to Mixmode Change</td>
<td>256</td>
</tr>
<tr>
<td>Playcollect and Playrecord in a Conference</td>
<td>256</td>
</tr>
<tr>
<td>Changing Participant Mixmode to Parked</td>
<td>256</td>
</tr>
<tr>
<td>Response to Parked</td>
<td>257</td>
</tr>
<tr>
<td>PlayCollect</td>
<td>257</td>
</tr>
</tbody>
</table>
### IVR with MSCML Sample Code and Call Flows

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Playing a Simple Announcement</td>
<td>271</td>
</tr>
<tr>
<td>Expected Response</td>
<td>271</td>
</tr>
<tr>
<td>Playing a Sequenced Announcement</td>
<td>271</td>
</tr>
<tr>
<td>Expected Response</td>
<td>271</td>
</tr>
<tr>
<td>Stopping a Play Command</td>
<td>271</td>
</tr>
<tr>
<td>Request 1 Payload</td>
<td>271</td>
</tr>
<tr>
<td>Request 2 Payload</td>
<td>272</td>
</tr>
<tr>
<td>Expected Response to Request 1 Payload</td>
<td>272</td>
</tr>
<tr>
<td>Expected Response to Request 2 Payload</td>
<td>272</td>
</tr>
<tr>
<td>PlayCollect</td>
<td>273</td>
</tr>
<tr>
<td>Payload</td>
<td>273</td>
</tr>
<tr>
<td>Expected Response</td>
<td>273</td>
</tr>
<tr>
<td>Playing a Recording</td>
<td>273</td>
</tr>
<tr>
<td>Payload</td>
<td>273</td>
</tr>
<tr>
<td>Expected Response</td>
<td>273</td>
</tr>
<tr>
<td>Stopping a Recording</td>
<td>273</td>
</tr>
<tr>
<td>Request 1 Payload</td>
<td>273</td>
</tr>
<tr>
<td>Request 2 Payload</td>
<td>274</td>
</tr>
<tr>
<td>Expected Response Request 1 Payload</td>
<td>274</td>
</tr>
<tr>
<td>Expected Response Request 2 Payload</td>
<td>274</td>
</tr>
<tr>
<td>Asynchronous DTMF</td>
<td>275</td>
</tr>
<tr>
<td>Subscribing to Standard Digit Events</td>
<td>275</td>
</tr>
<tr>
<td>Subscribing to Long Digit Events</td>
<td>275</td>
</tr>
<tr>
<td>Subscribing to Both Standard and Long Digit Events</td>
<td>275</td>
</tr>
<tr>
<td>Turning Off Digit Event Reporting</td>
<td>276</td>
</tr>
<tr>
<td>Example Responses</td>
<td>276</td>
</tr>
<tr>
<td>Call Flow for IVR with MSCML</td>
<td>278</td>
</tr>
<tr>
<td>Call Flow for PIN Collection, IVR with MSCML</td>
<td>279</td>
</tr>
<tr>
<td>Explanation of Call Flow</td>
<td>279</td>
</tr>
<tr>
<td>Call Flow for Recording a Message, IVR with MSCML</td>
<td>281</td>
</tr>
<tr>
<td>Explanation of Call Flow</td>
<td>281</td>
</tr>
<tr>
<td>Call Flow for MSCML MRCP Session Management</td>
<td>283</td>
</tr>
<tr>
<td>Scenario</td>
<td>283</td>
</tr>
<tr>
<td>Call Flow</td>
<td>283</td>
</tr>
<tr>
<td>VoiceXML Sample Code and Call Flows</td>
<td>289</td>
</tr>
<tr>
<td>Playing an Announcement</td>
<td>289</td>
</tr>
<tr>
<td>PIN Collection</td>
<td>289</td>
</tr>
<tr>
<td>Call Flow for VoiceXML</td>
<td>291</td>
</tr>
<tr>
<td>Transferring a Call</td>
<td>292</td>
</tr>
<tr>
<td>T.30 Fax Detection</td>
<td>294</td>
</tr>
<tr>
<td>Fax Call Transfer Call Flow</td>
<td>294</td>
</tr>
<tr>
<td>CNG Tone Detection and Event Notification</td>
<td>296</td>
</tr>
<tr>
<td>VoiceXML Implementation</td>
<td>296</td>
</tr>
<tr>
<td>VoiceXML Properties</td>
<td>297</td>
</tr>
<tr>
<td>Fax Detection—Example Script</td>
<td>297</td>
</tr>
<tr>
<td>T.38 Fax Detection, Termination, and Initiation</td>
<td>299</td>
</tr>
<tr>
<td>Detection and Termination</td>
<td>299</td>
</tr>
<tr>
<td>Initiation (Email to Fax)</td>
<td>299</td>
</tr>
<tr>
<td>snowshore.cfg Parameters</td>
<td>299</td>
</tr>
</tbody>
</table>
Fax Call Termination Call Flow .................................................. 300
CNG Tone Detection and Event Notification ................................. 301
VoiceXML Implementation ....................................................... 301
Fax Detection—Example VXML Script ........................................ 302
VXML Fax Record ................................................................. 303
Call Record - Example VXML Script ......................................... 303
Using VCR and Gain Controls .................................................. 304
Image Overlay and Text Overlay .............................................. 308
Simple Text Overlay .............................................................. 308
Scrolling Text Overlay ............................................................ 308
Image Overlay ..................................................................... 309
Late Media Sample Code ....................................................... 310

A - Audio Library ................................................................. 312

Sound Library .............................................................................. 313
Phrases and Messages ............................................................. 313
Numbers ................................................................................. 316
Dates and Ordinal Numbers ..................................................... 317
Letters ....................................................................................... 318
Time and Money ........................................................................ 319
Press Keys ............................................................................... 319
Quantities ................................................................................. 321
Miscellaneous Words ............................................................... 322
Generic Audio Files ................................................................. 324

B - VoiceXML Version 1.0 and Dialog Service. .......................... 326

About VoiceXML ................................................................. 327
VoiceXML Interpreter ............................................................. 327
Dialog Service Indicator and Request URI ................................. 327
VoiceXML Launcher ............................................................... 328
VoiceXML Concepts ............................................................... 328
Syntax ...................................................................................... 328
Scope ....................................................................................... 328
Resource Fetching ................................................................. 328
VoiceXML Application and Its Documents ................................. 328
Dialogs .................................................................................... 329
Grammar and Scripting ......................................................... 329
Session Variables ................................................................. 330
File Storage and Retrieval .................................................... 330
Media Content Recovery Extension ....................................... 330
VoiceXML Elements Reference ............................................... 332
<assign> ............................................................................... 332
<audio> .................................................................................. 332
<block> .................................................................................. 333
<brack> ............................................................................... 333
<catch> ..................................................................................... 333
<choice> .................................................................................. 334
<clear> ................................................................................... 334
<data> ...................................................................................... 335
<disconnect> .......................................................................... 336
<dtmf> .................................................................................... 336
<else> ...................................................................................... 336
<elseif> ................................................................................... 337
<error> .................................................................................... 337
<exit> ....................................................................................... 337
<table>
<thead>
<tr>
<th>VoiceXML Attributes</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>333</td>
</tr>
<tr>
<td>beep</td>
<td>334</td>
</tr>
<tr>
<td>bridge</td>
<td>335</td>
</tr>
<tr>
<td>class</td>
<td>336</td>
</tr>
<tr>
<td>cond</td>
<td>337</td>
</tr>
<tr>
<td>content</td>
<td>338</td>
</tr>
<tr>
<td>count</td>
<td>339</td>
</tr>
<tr>
<td>dest</td>
<td>340</td>
</tr>
<tr>
<td>destexpr</td>
<td>341</td>
</tr>
<tr>
<td>dtmf</td>
<td>342</td>
</tr>
<tr>
<td>dtmfterm</td>
<td>343</td>
</tr>
<tr>
<td>event</td>
<td>344</td>
</tr>
<tr>
<td>expr</td>
<td>345</td>
</tr>
<tr>
<td>expritem</td>
<td>346</td>
</tr>
<tr>
<td>finalsilence</td>
<td>347</td>
</tr>
<tr>
<td>http-equiv</td>
<td>348</td>
</tr>
<tr>
<td>id</td>
<td>349</td>
</tr>
<tr>
<td>longdigit</td>
<td>350</td>
</tr>
<tr>
<td>max-time</td>
<td>351</td>
</tr>
<tr>
<td>method</td>
<td>352</td>
</tr>
<tr>
<td>modal</td>
<td>353</td>
</tr>
<tr>
<td>mode</td>
<td>354</td>
</tr>
<tr>
<td>msecs</td>
<td>355</td>
</tr>
<tr>
<td>connect-timeout</td>
<td>356</td>
</tr>
<tr>
<td>id</td>
<td>357</td>
</tr>
<tr>
<td>longdigit</td>
<td>358</td>
</tr>
<tr>
<td>max-time</td>
<td>359</td>
</tr>
<tr>
<td>method</td>
<td>360</td>
</tr>
<tr>
<td>modal</td>
<td>361</td>
</tr>
<tr>
<td>mode</td>
<td>362</td>
</tr>
<tr>
<td>msecs</td>
<td>363</td>
</tr>
</tbody>
</table>

**VoiceXML Attributes Reference**

application ........................................... 353
bargein ............................................. 353
bargeintype ......................................... 353
class .................................................. 354
cond .................................................... 355
count ................................................... 356
dest ..................................................... 356
destexpr ............................................. 356
dtmf .................................................... 356
dtmfterm .............................................. 356
event ................................................... 357
expr ................................................... 357
expritem .............................................. 357
finalsilence ......................................... 357
http-equiv ........................................... 358
id ...................................................... 358
longdigit ............................................. 358
max-time .............................................. 358
method ................................................ 358
modal .................................................. 358
mode ................................................... 358
msecs .................................................. 358
List of Figures

Figure 1. Call Control Architecture ........................................... 26
Figure 2. Network Announcements ........................................... 35
Figure 3. Role of the IP Media Server with RTSP Server. ................. 42
Figure 4. Call Flow: SIP Provisional Response ............................. 78
Figure 5. Playing an Announcement Sequence .............................. 93
Figure 6. Call Flow for Simple Conference (Normal Media)............ 104
Figure 7. MRCP Session Management ....................................... 156
Figure 8. Sharing of Root Document in VoiceXML Application .......... 179
Figure 9. Call Flow: Announcement, Early Media .......................... 235
Figure 10. Call Flow: Announcement, Normal Media .................... 238
Figure 11. Call Flow: Stopping Media Using Hold ....................... 241
Figure 12. Call Flow: Simple Conference ................................. 244
Figure 13. Call Flow: Advanced Conference .............................. 247
Figure 14. Call Flow: IVR with MSCML ................................. 278
Figure 15. Call Flow: PIN Collection, IVR with MSCML. .............. 279
Figure 16. Call Flow: Recording, IVR with MSCML. .................... 281
Figure 17. Call Flow: VoiceXML ........................................... 291
Figure 18. T.30 Fax Call Transfer ........................................... 294
Figure 19. T.38 Fax Call Transfer ........................................... 300
Figure 20. Sharing of Root Document in VoiceXML Application ........ 329
Figure 21. Dial Pulse Detection Parameters ............................... 385
List of Tables

Table 1. SIP Service Indicators ................................................................. 27
Table 2. MSCML Conferencing and MSCML IVR Requests ............................... 28
Table 3. Functionality Supported by VoiceXML 1.0 and VoiceXML 2.0 ............... 29
Table 4. Supported Audio Codecs ............................................................... 30
Table 5. Supported Video Codecs ............................................................... 30
Table 6. Audio Encoding and File Storage Formats ....................................... 32
Table 7. Multimedia Encoding and File Storage Formats ................................. 33
Table 8. Application Service Indicators ....................................................... 48
Table 9. Valid Content (MIME) Types ......................................................... 49
Table 10. Supported SIP Methods ............................................................... 52
Table 11. SIP Headers in an INVITE ............................................................ 53
Table 12. SIP Headers in an ACK Method ..................................................... 54
Table 13. SIP Headers in a CANCEL Method ............................................... 55
Table 14. SIP Headers in an OPTIONS Method ............................................. 56
Table 15. SIP Headers in a BYE Method ...................................................... 57
Table 16. SIP Headers in an INFO Method .................................................. 58
Table 17. SIP Headers in a PRACK Method ................................................ 59
Table 18. Supported SIP Headers ............................................................... 60
Table 19. SIP Return Codes Generated by the IP Media Server ........................ 68
Table 20. SDP Description Headers ............................................................. 71
Table 21. SDP Time Description Headers .................................................... 72
Table 22. SDP Media Description Headers .................................................. 73
Table 23. Action Classes ........................................................................ 73
Table 24. Unreserved Characters in SIP Request ......................................... 79
Table 25. Unreserved Characters in User Portion ......................................... 79
Table 26. Unreserved Characters for Parameters ......................................... 80
Table 27. Simple Conference Attributes .................................................... 102
Table 28. Advanced Conferencing Attributes ............................................. 107
Table 29. Subscribe Element ................................................................. 107
Table 30. Attributes for Participant Legs in Advanced Conference ................... 107
Table 31. Subscribe Element for Participant Legs ......................................... 108
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 32.</td>
<td>Attributes for <code>&lt;managecontent&gt;</code> in an Advanced Conference</td>
</tr>
<tr>
<td>Table 33.</td>
<td>Configure_Leg Attributes in Advanced Conference</td>
</tr>
<tr>
<td>Table 34.</td>
<td>Configure_Conference Attributes in INVITE Message</td>
</tr>
<tr>
<td>Table 35.</td>
<td>Subscribe Element in INVITE with Configure_Conference</td>
</tr>
<tr>
<td>Table 36.</td>
<td>Configure_Conference Attributes in INFO Message</td>
</tr>
<tr>
<td>Table 37.</td>
<td>Subscribe Element in INFO with Configure_Conference</td>
</tr>
<tr>
<td>Table 38.</td>
<td>Configure_Leg Attributes</td>
</tr>
<tr>
<td>Table 39.</td>
<td>Subscribe Element for Configure_Leg</td>
</tr>
<tr>
<td>Table 40.</td>
<td>Configure_Team Attributes</td>
</tr>
<tr>
<td>Table 41.</td>
<td>MSCML Elements and Attributes for Creating Coached Conferencing</td>
</tr>
<tr>
<td>Table 42.</td>
<td>Timing Attributes</td>
</tr>
<tr>
<td>Table 43.</td>
<td>Additional Attributes of PlayRecord</td>
</tr>
<tr>
<td>Table 44.</td>
<td>Description of MRCP Session Management Topology</td>
</tr>
<tr>
<td>Table 45.</td>
<td>Attributes of <code>mrcp_create_session</code></td>
</tr>
<tr>
<td>Table 46.</td>
<td>Attributes of <code>mrcp_session_response</code></td>
</tr>
<tr>
<td>Table 47.</td>
<td>Attributes of <code>mrcp_terminate_session_request</code></td>
</tr>
<tr>
<td>Table 48.</td>
<td>MSCML Elements for IVR</td>
</tr>
<tr>
<td>Table 49.</td>
<td>Supported MIME Types, Formats, and Audio Encodings</td>
</tr>
<tr>
<td>Table 50.</td>
<td>Audio Codec Parameter Values</td>
</tr>
<tr>
<td>Table 51.</td>
<td>Supported MIME Types, Formats, and Video Encodings</td>
</tr>
<tr>
<td>Table 52.</td>
<td>ECMAScript Functionality</td>
</tr>
<tr>
<td>Table 53.</td>
<td>SDP Offer Sent in Outbound SIP INVITE</td>
</tr>
<tr>
<td>Table 54.</td>
<td>T.38 Fax Parameters</td>
</tr>
<tr>
<td>Table 55.</td>
<td>VCR and Gain Controls Support for RTSP in VoiceXML 2.0</td>
</tr>
<tr>
<td>Table 56.</td>
<td>Prompt Library: Standard Phrases</td>
</tr>
<tr>
<td>Table 57.</td>
<td>Prompt Library: Cardinal Numbers</td>
</tr>
<tr>
<td>Table 58.</td>
<td>Prompt Library: Dates and Ordinal Numbers</td>
</tr>
<tr>
<td>Table 59.</td>
<td>Prompt Library: Time and Money Phrases</td>
</tr>
<tr>
<td>Table 60.</td>
<td>Prompt Library: Press Key Phrases</td>
</tr>
<tr>
<td>Table 61.</td>
<td>Prompt Library: Quantities</td>
</tr>
<tr>
<td>Table 62.</td>
<td>Prompt Library: Miscellaneous Words</td>
</tr>
<tr>
<td>Table 63.</td>
<td>Generic Prompt Phrases</td>
</tr>
<tr>
<td>Table 64.</td>
<td>Supported MIME Types, Formats, and Audio Encodings</td>
</tr>
<tr>
<td>Table 65.</td>
<td>Accepted WAVE Format Specifiers</td>
</tr>
<tr>
<td>Table 66.</td>
<td>Codec Parameter Values</td>
</tr>
<tr>
<td>Table 67.</td>
<td>ECMAScript Functionality</td>
</tr>
<tr>
<td>Table 68.</td>
<td>Dial Pulse Detection Parameters</td>
</tr>
</tbody>
</table>
About this Publication

The Dialogic® PowerMedia™ IP Media Server (which is also referred to herein as "IP Media Server", "IPMS", 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 an economical 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 covers the following topics:

♦ Using this Publication
♦ Contacting Dialogic Technical Services and Support
Using this Publication

Audience and Purpose

This manual is for application developers who choose to use or employ the IP Media Server to deploy network announcements, conferences, Interactive Voice Response (IVR) and more in a voice over IP environment. 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.

Documentation Set

Dialogic provides the following publications for the IP Media Server:

- The Software Quick Start Guide describes how to install and configure Red Hat Enterprise Linux and IP Media Server software, configure a softphone, and run a demo.
- 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.
- The Command Line Interface Reference Guide describes the CLI utility which can be used to configure and troubleshoot the IP Media Server.
- Installing Red Hat Enterprise Linux 5 for the IP Media Server describes how to install and configure Red Hat Enterprise Linux 5 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 IP Media Server.
- Upgrading from Release 3.0.0 to 3.1.0 on Red Hat Enterprise Linux Platform provides information and instructions for upgrading to the current IP Media Server release from the previous release on platforms running Red Hat Enterprise Linux. It also includes instructions for downgrading in the event that you need to restore your previous configuration.

These publications, as well as Release Notes, are available in PDF format at http://www.dialogic.com/manuals.

Document Conventions

Conventions used in this document are described here.

Notes, Cautions, and Warnings

Notes contain information of general interest.

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 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 are blue. You can click on a cross-reference link 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 Services and Support

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

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

When reporting an issue to Technical Services and 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 support page.

Ordering Licenses

You must have a software license to use the IP Media Server. For directions on how to acquire licenses, see the IP Media Server License Activation Guide.
1 - Introduction

This chapter provides an overview of the features and functionality of the Dialogic® PowerMedia™ IP Media Server (also referred to in this document as “IP Media Server”, “IPMS”, or “Media Server”).

This chapter includes the following sections:

◆ Control Protocols and Services
◆ Call Control Protocols
◆ Media Storage, Processing, and Supported Codecs
◆ Services
◆ Advanced Video Features
◆ Video Transcoding
◆ Image Overlays and Text Overlays
◆ Real Time Streaming Protocol (RTSP)
Control Protocols and Services

This section 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. The IP Media Server can also provide video solutions such as Interactive Voice and Video Response (IVVR), multimedia ringback tone (MRBT), video portal, video mail, and video advertisements.

This section contains the following major topics:

- Call Control Protocols
- Media Storage, Processing, and Supported Codecs
- Services
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:

- Request the service
- Set up the connection
- Modify the connection, if necessary
- Tear 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.

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 bidirectional 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>
<thead>
<tr>
<th>Service Indicator</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>annc</td>
<td>Announcements</td>
</tr>
<tr>
<td>conf</td>
<td>Conferencing (simple and advanced with MSCML)</td>
</tr>
<tr>
<td>ivr</td>
<td>IVR with MSCML</td>
</tr>
<tr>
<td>dialog</td>
<td>VoiceXML</td>
</tr>
</tbody>
</table>

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 on the SIP page of the Web User Interface (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 about 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 more details about 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 shows the pages where these requests are defined.

<table>
<thead>
<tr>
<th>MSCML Conferencing</th>
<th>MSCML IVR Requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>configure_conference (page 125)</td>
<td>play (page 161)</td>
</tr>
<tr>
<td>configure_leg (page 127)</td>
<td>playcollect (page 161)</td>
</tr>
<tr>
<td></td>
<td>playrecord (page 161)</td>
</tr>
<tr>
<td></td>
<td>stop (page 162)</td>
</tr>
</tbody>
</table>

For details about 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 371).

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.

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

<table>
<thead>
<tr>
<th>Functionality</th>
<th>VoiceXML 1.0</th>
<th>VoiceXML 2.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRCP 1.0 for Automated Speech Recognition (ASR)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>MRCP 1.0 for Text to Speech (TTS)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RTSP</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>T.30 fax detection</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>T.38 fax detection, termination, and initiation</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Text overlay and image overlay</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

For further details about VoiceXML 1.0, see “VoiceXML Version 1.0 and Dialog Service” (page 326). For further details about VoiceXML 2.0, see Chapter 6, “VoiceXML Version 2.0 and Dialog Service”.
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. Supported audio codecs are summarized in Table 4.

Table 4. Supported Audio Codecs

<table>
<thead>
<tr>
<th>Codec</th>
<th>RTP payload type</th>
<th>Mode</th>
<th>Bandwidth (Kbps)</th>
<th>Packet time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMR-NB</td>
<td>96... 127</td>
<td>0</td>
<td>4.75</td>
<td>20, 40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>5.15</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>5.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>6.7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>7.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>7.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>10.2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>12.2</td>
<td></td>
</tr>
<tr>
<td>G.711 ulaw</td>
<td>0</td>
<td>N/A</td>
<td>64</td>
<td>10, 20, 30</td>
</tr>
<tr>
<td>G.711 alaw</td>
<td>8</td>
<td>N/A</td>
<td>64</td>
<td>10, 20, 30</td>
</tr>
<tr>
<td>G.726</td>
<td>2</td>
<td>N/A</td>
<td>32</td>
<td>10, 20, 30</td>
</tr>
<tr>
<td>G.729</td>
<td>18</td>
<td>N/A</td>
<td>8</td>
<td>10, 20, 30</td>
</tr>
</tbody>
</table>

Supported video codecs are summarized in Table 5.

Table 5. Supported Video Codecs

<table>
<thead>
<tr>
<th>Codec</th>
<th>RTP Payload Type</th>
<th>Profile and Level</th>
<th>Maximum bit rate (Kbps)</th>
<th>Maximum frame rate (fps)</th>
<th>Frame size</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.263</td>
<td>34</td>
<td>Baseline (0) Level 10, 20, 30</td>
<td>384</td>
<td>30</td>
<td>CIF, QCIF</td>
</tr>
<tr>
<td>H.263-1998 (H.263+)</td>
<td>96...127</td>
<td>Baseline (0) Level 10, 20, 30</td>
<td>384</td>
<td>30</td>
<td>CIF, QCIF</td>
</tr>
<tr>
<td>H.263-2000 (H.263++)</td>
<td>96...127</td>
<td>Baseline (0) Level 10, 20, 30, 45</td>
<td>384</td>
<td>30</td>
<td>CIF, QCIF</td>
</tr>
<tr>
<td>H.264</td>
<td>96...127</td>
<td>Baseline Level 1, 1b, 1.1, 1.2, 1.3</td>
<td>768</td>
<td>30</td>
<td>CIF, QCIF</td>
</tr>
</tbody>
</table>

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 30).
You can license the G.726, G.729AB, and AMR-NB codecs to run on the host.

**G.729**  
The IP Media Server supports G.729AB as an RTP codec. (G.729A for encode/decode. G.729AB for decode only.)

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 32 Kbps as an RTP codec.

**AMR-NB**  
The IP Media Server supports AMR-NB (Adaptive Multi-Rate Narrow Band) as an RTP codec. The IP Media Server supports the following features of AMR-NB:

- Octet-alignment: Bandwidth Efficient (bit-aligned) mode or Octet-aligned (byte-aligned) mode
- Mode-set: 0 to 7 (bit rates between 4.75 and 12.2 Kbps)
- Simple payload sorting
- One channel per session

The host-based AMR-NB codec supports true adaptive rates. These rates can be controlled during an active bidirectional AMR-NB session by Codec Mode Requests (CMR) in the AMR-NB media received from the remote device, so you can specify the mode-set and the far end can select from that list when choosing a CMR value. The far end may opt to send a CMR value of 15, indicating no preference.

When the IP Media Server receives a CMR embedded in the incoming AMR-NB data, it changes its AMR-NB output bit rate to comply with the request; however, it will always output a CMR value of 15 (No Mode Request Present) in its outgoing AMR-NB data.

The following features are **not** supported by the IP Media Server:

- 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.

For more information on using AMR-NB, see “Using AMR-NB” (page 84).
Audio Content Storage

The IP Media Server can retrieve and play audio files with content encoding and file storage formats shown in Table 6.

Table 6. Audio Encoding and File Storage Formats

<table>
<thead>
<tr>
<th>Content Encoding</th>
<th>File Storage Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>G.711</td>
<td>raw, .wav, .au</td>
</tr>
<tr>
<td>MSGSM</td>
<td>raw, .wav</td>
</tr>
<tr>
<td>3GPP (AMR-NB)</td>
<td>.3gp</td>
</tr>
<tr>
<td>3GPP2 (AMR-NB)</td>
<td>.3g2</td>
</tr>
</tbody>
</table>

Raw (headerless) G.711 a-law/μ-law content is represented as *.alaw and *.ulaw. Raw (headerless) MSGSM content is represented as *.msgsm or *.ms_gsm.

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

Video

The IP Media Server can transmit and receive RTP video packets encoded in H.263, H.263+, and H.264. See Table 5 (page 30).

Video is currently supported by the MSCML (conf) and VoiceXML (dialog) services.

Video transcoding is only supported by VoiceXML, and only during playback; it is not supported for recording. Video recordings are done natively. For more information about video transcoding, see “Video Transcoding” (page 39).

Mixed Audio/Video

**Note:** The IP Media Server will not play a video file that is not accompanied by audio. If a file has video that extends beyond the end of the audio track, the playback will be terminated at the end of the audio track.

WAV-based

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+, H.263++, or H.264 video track.

This content type is currently supported by the MSCML (conf) and VoiceXML (dialog) service.
3GPP and 3GPP2

The IP Media Server supports playback of an associated video and audio stream of 3rd Generation Partnership Project (3GPP) and 3rd Generation Partnership Project 2 (3GPP2) file formats, provided that the video stream is stored in an H.263+, H.263++ or H.264 video track, and the audio stream is stored in a separate AMR-NB audio track.

On playback, the IP Media Server packetizes the video track into RTP streams, and the audio track into RTP packets in one of the supported RTP encodings (G.711, G.726, G.729, AMR-NB).

The IP Media Server also supports recording in the 3GPP file format (it does not currently support recording in 3GPP2). For an example of how to do this in VoiceXML 2.0, see “MIME Types Supported” (page 218).

The MIME type for 3GPP is video/3gpp. The MIME type for 3GPP2 is video/3gpp2.

The 3GPP and 3GPP2 file formats are currently supported by the MSCML (conf) and VoiceXML (dialog) services.

Mixed Audio/Video Content Storage

The IP Media Server can retrieve and play multimedia files with content encoding and file storage formats shown in Table 7.

<table>
<thead>
<tr>
<th>Content Encoding</th>
<th>File Storage Formats</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>3GPP</td>
<td>.3gp</td>
<td>Audio: AMR-NB Video: H.263+, H.263++, or H.264</td>
</tr>
<tr>
<td>3GPP2</td>
<td>.3g2</td>
<td>Audio: AMR-NB Video: H.263+, H.263++, or H.264</td>
</tr>
<tr>
<td>WAV-based</td>
<td>.wav</td>
<td>Audio: G.711 Video: H.263, H.263+, H.263++, or H.264</td>
</tr>
</tbody>
</table>

File Storage and Retrieval

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

Network File System (NFS)

The IP Media Server can retrieve, play, store, and record data files using the Network File System (NFS) protocol, as defined in 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.
Real Time Streaming Protocol (RTSP)

The IP Media Server can retrieve and play files stored on a Real Time Streaming Protocol (RTSP) server, as defined in RFC 2326. The files stored on the RTSP server must have audio and video codecs that are supported by the local IP Media Server file playback; see Table 4 (page 30) and Table 5 (page 30).

DTMF

The IP Media Server supports the in-band tone detection of the DTMF digit set (0-9, #, *, A-D), 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 more information about 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
- Interactive Voice Response (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.

Example:

```plaintext
INVITE sip:ann@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 312)).

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 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, 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:annccMS_IP;play=circuit_busy.ulaw SIP/2.0
```

as:

```
INVITE sip:annccMS_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 a 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.

Interactive Voice Response (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 2.0 and 1.0 are supported on the IP Media Server.

Refer to “VoiceXML Version 2.0 and Dialog Service” (page 176) for information about VoiceXML 2.0.

Refer to “VoiceXML Version 1.0 and Dialog Service” (page 326) for information about VoiceXML 1.0.
Advanced Video Features

This section describes the advanced video features supported by the IP Media Server:

- Video Transcoding
- Image Overlays and Text Overlays
Video Transcoding

Video transcoding is the process of converting video media from one video codec type to another (for example, from H.264 to H.263) between two endpoints to suit the requirements of the device at each endpoint. Transcoding involves decoding and encoding each frame of a video stream.

In the process of transcoding, features such as frame rate conversion, bit rate control, image resizing, and text and image overlay can be applied. These features are not applicable if video transcoding is not being used.

Video transrating adjusts the number of video frames per second between two endpoints to suit the requirements of the device at each endpoint.

Bit rate control limits the video input bit rate to a lesser value at the output while applying the algorithms to control video quality.

Image resizing converts video from one image size to another (for example, from CIF to QCIF) between two endpoints to suit the requirements of the device at each endpoint.

In this document, the term video transcoding or video transcoder encompasses video transcoding, video transrating, bit rate control, and image resizing.

For a list of video codecs supported by the video transcoder, see Table 5 (page 30).

Note: Use of the video transcoder requires an advanced video license from Dialogic. Each license supports a half-duplex (unidirectional) video transcoding session.

Video Transcoder Settings

The video transcoder supports the following settings, which are configured in the Web UI, Configure SIP page:

Auto

Automatically enables or disables video transcoding on each video connection according to a set of rules (see Video Transcoder Automatic Setting Rules). This is the default setting.

Force

Enables video transcoding for each video connection. This option supports as many video connections as there are video transcoding licenses.

Turn off

Disables video transcoding for each video connection.

Video Transcoder Automatic Setting Rules

The automatic (auto) setting for video transcoding allows the IP Media Server to determine when video transcoding is required on a video connection and enable it as needed. This setting results in optimal use of video transcoding resources.
A video transcoding license is consumed and video transcoding is enabled for the automatic setting when any of the following conditions is true:

- The source codec does not match the target codec.
- The source image size does not match the target image size.
- The source frame rate is greater than the target frame rate.
- The source bit rate is greater than the target bit rate.
- A text overlay or image overlay is being applied to the video stream.

The video transcoding license is released and video transcoding is disabled when all of these conditions are false.

**Video Transcoder Restrictions**

If a session without transcoding is in progress, and playback begins of a new video file whose header indicates that it is not compatible with the endpoint, the IP Media Server continues streaming the video without transcoding.
Image Overlays and Text Overlays

An image overlay provides the ability to superimpose one or more images (graphics) over a video stream. A text overlay provides the ability to superimpose one or more text elements over a video stream. The resulting video stream contains both the overlay elements and the original video stream.

Overlay elements can be of varying sizes, and in the case of text elements, also of varying fonts and colors. These elements can be inserted in real time. For example, you can create an interactive menu or display a corporate logo as an overlay over a video stream.

Overlay functionality is implemented in VoiceXML 2.0.

The IP Media Server supports the following file types for image overlay:

- JPEG
- PNG

The IP Media Server supports the following text formats for text overlay:

- ASCII
- UTF-8 as defined in IETF RFC 3629
- GB18030 as defined in the Chinese National Standard GB 18030-2005

For more information on overlays, see “Image Overlay and Text Overlay Functionality” (page 227).

**Note:** Use of a text overlay or image overlay requires an advanced video license from Dialogic.
Real Time Streaming Protocol (RTSP)

The IP Media Server supports the Real Time Streaming Protocol (RTSP) as a client, as defined in RFC 2326. RTSP is a network control protocol used by RTSP clients to control streaming media servers.

The IP Media Server provides the capability to stream media from RTSP servers to SIP endpoints. This capability is implemented through the RTSP client stack included with the IP Media Server (the RTSP server stack is not included). The SIP endpoints can control playback of the media stream from the RTSP server.

Applications for RTSP include large-scale broadcasts, video voice-mail, and audio/video-on-demand streaming.

RTSP provides a means to choose and establish delivery channels, such as UDP, and to control media sessions between endpoints. RTSP clients can control playback of media using pause, fast forward, and rewind commands. The IP Media Server does not support recording to an RTSP server.

RTSP functionality is supported with VoiceXML 2.0. Properties provided in the VoiceXML 2.0 browser are used to pause, fast forward, and rewind audio and video playback from an RTSP server. For more information, see “Using VCR and Gain Controls” (page 304).

Figure 3 illustrates the role of the IP Media Server in streaming media from an RTSP server to a SIP endpoint.

Figure 3. Role of the IP Media Server with RTSP Server

Configuring Session Keep Alive

The IP Media Server assumes that RTSP servers use RTCP RR (Receiver Reports) for session keep alive.

If an RTSP server uses a different method (listed below) for session keep alive, you will need to specify and configure the following parameters in snowshore.cfg for the RTSP client:

RtspSetParam

Enable or disable SET_PARAMETER method. Values are: 0=auto, 1=enable, 2=disable. Default value is 2.

RtspGetParam

Enable or disable GET_PARAMETER method. Values are: 0=auto, 1=enable, 2=disable. Default value is 2.
RtspPingFlag

Enable or disable PING flag. This flag controls a ping option used by some servers with GET_PARAMETER method. Values are: 0=disable, 1=enable. Default value is 0.

RtspOptionsFlag

Enable or disable OPTIONS flag. Values are: 0=disable, 1=enable. Default value is 0.

The following parameter is used to change the time value for keep alive.

RtspKeepAliveTimeValue

Time interval in seconds between successive sessions keep alive. Default value is 30 seconds.
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 includes the following major sections:

- Conformance
- Media Server Availability
- SIP Description
- SIP Requests
- SIP Methods
- SIP Headers
- SIP Responses
- Session Description Protocol
- Ports
- Using AMR-NB
Conformance

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


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.

Please contact Dialogic Technical Support if you need to enable this feature.
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 multiplayer 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 INVITE message 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:ann@MS_IP.

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

Table 8 lists the SIP services and associated indicators.

### Table 8. Application Service Indicators

<table>
<thead>
<tr>
<th>Service</th>
<th>Indicator</th>
<th>Example*</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Announcements</td>
<td>annnc</td>
<td>INVITE sip:annnc@MS_IP; play=(etc.) SIP/2.0</td>
<td>82</td>
</tr>
<tr>
<td>Conferencing</td>
<td>conf</td>
<td>INVITE sip:conf=confid@MS_IP SIP/2.0</td>
<td>99</td>
</tr>
<tr>
<td>IVR</td>
<td>ivr</td>
<td>INFO sip:ivr@MS_IP SIP/2.0</td>
<td>82</td>
</tr>
<tr>
<td>VoiceXML</td>
<td>dialog</td>
<td>INVITE sip:dialog@MS_IP; voicexml=<a href="http://path/filename.vxml">http://path/filename.vxml</a> SIP/2.0</td>
<td>82 and 177</td>
</tr>
</tbody>
</table>

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 User Interface as follows:

1. Select Media Server > Configure > 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 > Configure > VOICEXML menu.
SIP Message Body Types

Table 9 lists the valid message body content types.

Table 9. Valid Content (MIME) Types

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

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.
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:annnc@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:annnc@192.168.12.155:5060>
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:annnc@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

```plaintext
INVITE sip:ann@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`.

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

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 45).

**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 94) for more information.

**Headers in an INVITE**

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

<table>
<thead>
<tr>
<th>Header</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call-ID (i)</td>
<td>62</td>
</tr>
<tr>
<td>CSeq</td>
<td>63</td>
</tr>
<tr>
<td>From (f)</td>
<td>63</td>
</tr>
<tr>
<td>To (t)</td>
<td>65</td>
</tr>
<tr>
<td>Via (v)</td>
<td>65</td>
</tr>
<tr>
<td>Contact (m)</td>
<td>62</td>
</tr>
<tr>
<td>Accept</td>
<td>61</td>
</tr>
<tr>
<td>Content-Length (l)</td>
<td>62</td>
</tr>
<tr>
<td>Record-Route</td>
<td>64</td>
</tr>
<tr>
<td>Require</td>
<td>64</td>
</tr>
<tr>
<td>Route</td>
<td>64</td>
</tr>
<tr>
<td>Session-Expires</td>
<td>64</td>
</tr>
<tr>
<td>Subject</td>
<td>64</td>
</tr>
<tr>
<td>Supported</td>
<td>65</td>
</tr>
<tr>
<td>Unsupported</td>
<td>65</td>
</tr>
</tbody>
</table>
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 12 lists the mandatory and optional headers in a SIP ACK method.

Table 12. SIP Headers in an ACK Method

<table>
<thead>
<tr>
<th>Header</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory</td>
<td></td>
</tr>
<tr>
<td>Call-ID (i)</td>
<td>62</td>
</tr>
<tr>
<td>CSeq</td>
<td>63</td>
</tr>
<tr>
<td>From (f)</td>
<td>63</td>
</tr>
<tr>
<td>To (t)</td>
<td>65</td>
</tr>
<tr>
<td>Via (v)</td>
<td>65</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>Content-Length (l)</td>
<td>62</td>
</tr>
<tr>
<td>Content-Type (c)</td>
<td>63</td>
</tr>
<tr>
<td>Record-Route</td>
<td>64</td>
</tr>
<tr>
<td>Require</td>
<td>64</td>
</tr>
<tr>
<td>Route</td>
<td>64</td>
</tr>
<tr>
<td>Supported</td>
<td>65</td>
</tr>
<tr>
<td>Unsupported</td>
<td>65</td>
</tr>
</tbody>
</table>

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

**Table 13. SIP Headers in a CANCEL Method**

<table>
<thead>
<tr>
<th>Header</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mandatory</strong></td>
<td></td>
</tr>
<tr>
<td>Call-ID (i)</td>
<td>62</td>
</tr>
<tr>
<td>CSeq</td>
<td>63</td>
</tr>
<tr>
<td>From (f)</td>
<td>63</td>
</tr>
<tr>
<td>To (t)</td>
<td>65</td>
</tr>
<tr>
<td>Via (v)</td>
<td>65</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td></td>
</tr>
<tr>
<td>Accept</td>
<td>61</td>
</tr>
<tr>
<td>Content-Length (l)</td>
<td>62</td>
</tr>
<tr>
<td>Record-Route</td>
<td>64</td>
</tr>
<tr>
<td>Require</td>
<td>64</td>
</tr>
<tr>
<td>Supported</td>
<td>65</td>
</tr>
<tr>
<td>Unsupported</td>
<td>65</td>
</tr>
</tbody>
</table>
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 14 lists the mandatory and optional headers in a SIP OPTIONS method.

Table 14. SIP Headers in an OPTIONS Method

<table>
<thead>
<tr>
<th>Header</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandatory</td>
<td></td>
</tr>
<tr>
<td>Call-ID (i)</td>
<td>62</td>
</tr>
<tr>
<td>CSeq</td>
<td>63</td>
</tr>
<tr>
<td>From (f)</td>
<td>63</td>
</tr>
<tr>
<td>To (t)</td>
<td>65</td>
</tr>
<tr>
<td>Via (v)</td>
<td>65</td>
</tr>
<tr>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>Accept</td>
<td>61</td>
</tr>
<tr>
<td>Content-Length (l)</td>
<td>62</td>
</tr>
<tr>
<td>Record-Route</td>
<td>64</td>
</tr>
<tr>
<td>Require</td>
<td>64</td>
</tr>
<tr>
<td>Supported</td>
<td>65</td>
</tr>
<tr>
<td>Unsupported</td>
<td>65</td>
</tr>
</tbody>
</table>
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 15 lists the mandatory and optional headers in a SIP BYE method.

<table>
<thead>
<tr>
<th></th>
<th>Header</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mandatory</strong></td>
<td>Call-ID (i)</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>CSeq</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>From (f)</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>To (t)</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>Via (v)</td>
<td>65</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td>Accept</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td>Content-Length (l)</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>Content-Type (c)</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>Record-Route</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>Require</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>Route</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>Supported</td>
<td>65</td>
</tr>
<tr>
<td></td>
<td>Unsupported</td>
<td>65</td>
</tr>
</tbody>
</table>
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 16 lists the mandatory and optional headers in a SIP INFO method.

<table>
<thead>
<tr>
<th>Mandatory</th>
<th>Header</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call-ID (i)</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>CSeq</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>From (f)</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>To (t)</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>Via (v)</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>Optional</td>
<td>Accept</td>
<td>61</td>
</tr>
<tr>
<td>Content-Length (l)</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>Content-Type (c)</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>Record-Route</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>Require</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>Supported</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>Unsupported</td>
<td>65</td>
<td></td>
</tr>
</tbody>
</table>
**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 4 (page 78)* shows a call flow for a provisional response.

**Headers in a PRACK**

Table 17 lists the SIP headers in a PRACK Method.

<table>
<thead>
<tr>
<th>Table 17. SIP Headers in a PRACK Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
</tr>
<tr>
<td><strong>Mandatory</strong></td>
</tr>
<tr>
<td>Call-ID (i)</td>
</tr>
<tr>
<td>CSeq</td>
</tr>
<tr>
<td>From (f)</td>
</tr>
<tr>
<td>To (t)</td>
</tr>
<tr>
<td>Via (v)</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
</tr>
<tr>
<td>Accept</td>
</tr>
<tr>
<td>Content-Length (l)</td>
</tr>
<tr>
<td>Record-Route</td>
</tr>
<tr>
<td>Require</td>
</tr>
<tr>
<td>Supported</td>
</tr>
<tr>
<td>Unsupported</td>
</tr>
</tbody>
</table>
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 18 lists the SIP headers that the IP Media Server supports. The headers are described in “Supported Headers” (page 61). Unsupported headers are parsed syntactically, but then discarded.

<table>
<thead>
<tr>
<th>SIP Header</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Indicates acceptable message types.</td>
<td>61</td>
</tr>
<tr>
<td>Call-ID (i)</td>
<td>Identifies a session or a call.</td>
<td>62</td>
</tr>
<tr>
<td>Contact (m)</td>
<td>URL where the user can be reached.</td>
<td>62</td>
</tr>
<tr>
<td>Content-Length (l)</td>
<td>Size of message body.</td>
<td>62</td>
</tr>
<tr>
<td>Content-Type (c)</td>
<td>Media type in the message.</td>
<td>63</td>
</tr>
<tr>
<td>CSeq (Mandatory)</td>
<td>Orders different requests within the same session and matches requests against responses.</td>
<td>63</td>
</tr>
<tr>
<td>From (f) (Mandatory)</td>
<td>Initiator of the request (name and SIP URL).</td>
<td>63</td>
</tr>
<tr>
<td>Max-Forwards (Mandatory)</td>
<td>Upper limit on the number of intermediary proxies.</td>
<td>63</td>
</tr>
<tr>
<td>Record-Route</td>
<td>Forces routing through a proxy.</td>
<td>64</td>
</tr>
<tr>
<td>Require</td>
<td>Options that must be supported.</td>
<td>64</td>
</tr>
<tr>
<td>Route</td>
<td>Forces routing.</td>
<td>64</td>
</tr>
<tr>
<td>Session-Expires</td>
<td>Serves as watchdog time out.</td>
<td>64</td>
</tr>
<tr>
<td>Server</td>
<td>Provides MS identifying information in IP Media Server request messages.</td>
<td>64</td>
</tr>
<tr>
<td>Subject</td>
<td>Indicates client or server.</td>
<td>64</td>
</tr>
<tr>
<td>Supported</td>
<td>One or more options.</td>
<td>65</td>
</tr>
</tbody>
</table>
Table 18. Supported SIP Headers

<table>
<thead>
<tr>
<th>SIP Header</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>To (t)</td>
<td>Recipient of the request (name and SIP URL).</td>
<td>65</td>
</tr>
<tr>
<td>(Mandatory)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unsupported (Mandatory)</td>
<td>Features not supported by the server.</td>
<td>65</td>
</tr>
<tr>
<td>User-Agent</td>
<td>Provides IP Media Server identifying information in IP Media Server response messages.</td>
<td>65</td>
</tr>
<tr>
<td>Via (v)</td>
<td>Records the route taken by a request.</td>
<td>65</td>
</tr>
<tr>
<td>(Mandatory)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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 9 (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 draft-ietf-sip-rfc2543bis-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.

For example, this identifier is not guaranteed to be unique in the network:

```
Call-ID: 123456
```

In this example, this identifier is guaranteed to be unique in the network:

```
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 draft-ietf-sip-rfc2543bis-03, 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 draft-ietf-sip-rfc2543bis-03 "SIP: Session Initiation Protocol", 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 19 lists the return codes that can be sent in responses from the IP Media Server, and notes when and why they are sent.

**Table 19. SIP Return Codes Generated by the IP Media Server**

<table>
<thead>
<tr>
<th>Code</th>
<th>Code Classes</th>
<th>Reason Phrase</th>
<th>Examples of When / Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Provisional</td>
<td>Trying</td>
<td>INVITE received.</td>
</tr>
<tr>
<td>183</td>
<td>Provisional</td>
<td>Session progress</td>
<td>Early media requested in INVITE for announcement (annc with early=yes).</td>
</tr>
<tr>
<td>200</td>
<td>Success</td>
<td>OK</td>
<td>• Request accepted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CANCEL and BYE received.</td>
</tr>
<tr>
<td>400</td>
<td>Client Error</td>
<td>Bad request</td>
<td>• Error detected with a message body (such as the failure to execute an XML script).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Request received to create a conference that already exists.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Request received, but the session is not stable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Request cannot be decoded by the SIP parser.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td>401</td>
<td>Client Error</td>
<td>Unauthorized</td>
<td>Request missing required headers.</td>
</tr>
<tr>
<td>403</td>
<td>Client Error</td>
<td>Forbidden</td>
<td>Requested SDP is not supported; for example, G.729a is requested for a conference.</td>
</tr>
<tr>
<td>404</td>
<td>Client Error</td>
<td>File not Found</td>
<td>Announcement content not found</td>
</tr>
<tr>
<td>405</td>
<td>Client Error</td>
<td>Method not allowed</td>
<td>Stable session established, but UAC attempts to perform an unsupported method.</td>
</tr>
<tr>
<td>415</td>
<td>Client Error</td>
<td>Unsupported media type</td>
<td>Requested SDP contains unsupported media type.</td>
</tr>
</tbody>
</table>
Table 19. SIP Return Codes Generated by the IP Media Server (continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Code Classes</th>
<th>Reason Phrase</th>
<th>Examples of When / Why</th>
</tr>
</thead>
<tbody>
<tr>
<td>420</td>
<td>Client Error</td>
<td>Bad Extension</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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. |
| 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. |
### Table 19. SIP Return Codes Generated by the IP Media Server (continued)

<table>
<thead>
<tr>
<th>Code</th>
<th>Code Classes</th>
<th>Reason Phrase</th>
<th>Examples of When / Why</th>
</tr>
</thead>
</table>
| 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 multimedia 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 20 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 20 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 23 (page 73).

**Table 20. SDP Description Headers**

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
<th>Action (When Received/When Sent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>Protocol version which is indicated by ‘v=0’.</td>
<td>Ignored/Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Protocol version header is ignored when received in an offer and set to zero on a created offer or response.</td>
</tr>
<tr>
<td>o</td>
<td>Owner/creator and session identifier.</td>
<td>Ignored/Static</td>
</tr>
<tr>
<td></td>
<td>o=&lt;username&gt; &lt;session id&gt;&lt;version&gt; &lt;network type&gt;</td>
<td>The origin header is ignored when received as an offer and set to static and created values.</td>
</tr>
<tr>
<td></td>
<td>&lt;address type&gt; &lt;address&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Application Developer’s Guide  71
Table 21 lists and defines the SDP time description headers.

**Table 21. SDP Time Description Headers**

<table>
<thead>
<tr>
<th>Header</th>
<th>Description</th>
<th>Action (When Received/When Sent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>Session name (one per session description)</td>
<td>Ignored/Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ignored when offered. Static</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value set to &quot;SnowShore SDP&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>when created.</td>
</tr>
<tr>
<td>*i</td>
<td>Session information</td>
<td>Ignored</td>
</tr>
<tr>
<td>*u</td>
<td>URI of description</td>
<td>Ignored</td>
</tr>
<tr>
<td>*e</td>
<td>Email address</td>
<td>Ignored</td>
</tr>
<tr>
<td>*p</td>
<td>Phone number</td>
<td>Ignored</td>
</tr>
<tr>
<td>*c</td>
<td>Connection information.</td>
<td>Extracted/Static/Created</td>
</tr>
<tr>
<td></td>
<td>c=&lt;network type&gt; &lt;address type&gt; &lt;connection address&gt;</td>
<td>Extracted when offered.</td>
</tr>
<tr>
<td></td>
<td>Not required if included in all media descriptions. See “c=</td>
<td>Static when not.</td>
</tr>
<tr>
<td></td>
<td>(Connection data)” (page 74) for details.</td>
<td>Created for late media.</td>
</tr>
<tr>
<td>*b</td>
<td>Bandwidth information</td>
<td>Ignored</td>
</tr>
<tr>
<td>*z</td>
<td>Time zone adjustments</td>
<td>Ignored</td>
</tr>
<tr>
<td>*k</td>
<td>Encryption key</td>
<td>Ignored</td>
</tr>
<tr>
<td>*a</td>
<td>Zero or more attribute lines.</td>
<td>Extracted/Static/Created</td>
</tr>
<tr>
<td></td>
<td>See page 75 for details.</td>
<td></td>
</tr>
</tbody>
</table>

**Time Description Headers**

Table 21 lists and defines the SDP time description headers.
Media Description Headers

Table 22 lists and defines SDP media description headers.

**Table 22. SDP Media Description Headers**

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

Header Action Classes

Table 23 lists and defines the four different types of SDP header action classes associated with the SDP headers.

**Table 23. Action Classes**

<table>
<thead>
<tr>
<th>Header</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ignored</td>
<td>The header and/or header fields are ignored.</td>
</tr>
<tr>
<td>Static</td>
<td>The header and/or header fields are static within the session control stack.</td>
</tr>
<tr>
<td>Extracted</td>
<td>The header and/or header fields are extracted from the parsed SIP method or SDP.</td>
</tr>
<tr>
<td>Created</td>
<td>The header and/or header fields are dynamic.</td>
</tr>
</tbody>
</table>

Header Definitions

The following sections describe the SDP headers and their respective attributes.
**m= (Media information)**

\[ m=<\text{media}> <\text{port}> <\text{transport}> <\text{fmt list}> \]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;\text{media}&gt;)&lt;\text{port}&gt;</td>
<td>Media type is set to audio or video when offered, followed by the starting port number. All other media types are ignored.</td>
</tr>
<tr>
<td>(&lt;\text{transport}&gt;)</td>
<td>Transport protocol is always set to RTP/AV.</td>
</tr>
<tr>
<td>(&lt;\text{fmt list}&gt;)</td>
<td>Static or dynamic payload codec type(s).</td>
</tr>
</tbody>
</table>

**Note:** For a list of supported media formats, see “Media Storage, Processing, and Supported Codecs” (page 30) and Table 4 (page 30).

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 *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)**

\[ c=<\text{network type}> <\text{address type}> <\text{connection_address}> \]

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network type</td>
<td>IN</td>
</tr>
<tr>
<td>Address type</td>
<td>IP4</td>
</tr>
<tr>
<td>Connection address</td>
<td>IP address is set to the address designated for the media stream</td>
</tr>
</tbody>
</table>

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 81) 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>direction</td>
<td>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.</td>
</tr>
<tr>
<td>packet time (ptime)</td>
<td>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.</td>
</tr>
<tr>
<td>dynamic payload type (rtpmap)</td>
<td>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.</td>
</tr>
</tbody>
</table>
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, RtpPortLow 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=1001356403
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 4 shows the call flow with a SIP provisional response. The numbers that follow indicate their respective sections listed inside Figure 4.

1. Initial INVITE request contains Supported header with 100rel.
2. 1xx response contains required 100rel and Rseq headers.
3. Each 1xx response requires the UAC to return a PRACK request which makes the Media Server send a 200 OK 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 24 lists the characters that are unreserved in a SIP request, as defined in RFC 2976.

### Table 24. Unreserved Characters in SIP Request

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>hyphen</td>
</tr>
<tr>
<td>_</td>
<td>underscore</td>
</tr>
<tr>
<td>.</td>
<td>period</td>
</tr>
<tr>
<td>!</td>
<td>exclamation point</td>
</tr>
<tr>
<td>~</td>
<td>tilde</td>
</tr>
<tr>
<td>*</td>
<td>asterisk</td>
</tr>
<tr>
<td>'</td>
<td>single quote</td>
</tr>
<tr>
<td>( )</td>
<td>parentheses</td>
</tr>
<tr>
<td>/</td>
<td>forward slash</td>
</tr>
</tbody>
</table>

Table 25 lists the characters that are unreserved in the user portion (before the @ sign).

### Table 25. Unreserved Characters in User Portion

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>ampersand</td>
</tr>
<tr>
<td>=</td>
<td>equal sign</td>
</tr>
<tr>
<td>+</td>
<td>plus sign</td>
</tr>
<tr>
<td>$</td>
<td>dollar sign</td>
</tr>
<tr>
<td>,</td>
<td>comma</td>
</tr>
<tr>
<td>;</td>
<td>semi-colon</td>
</tr>
<tr>
<td>?</td>
<td>question mark</td>
</tr>
<tr>
<td>/</td>
<td>forward slash</td>
</tr>
</tbody>
</table>
Table 26 lists the characters that are unreserved for parameters.

**Table 26. Unreserved Characters for Parameters**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>brackets</td>
</tr>
<tr>
<td>/</td>
<td>forward slash</td>
</tr>
<tr>
<td>:</td>
<td>colon</td>
</tr>
<tr>
<td>&amp;</td>
<td>ampersand</td>
</tr>
<tr>
<td>+</td>
<td>plus sign</td>
</tr>
<tr>
<td>$</td>
<td>dollar sign</td>
</tr>
</tbody>
</table>

**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**

```plaintext
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=snow-bound

--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
```
Ports

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 82).

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 (ann)

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.
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.

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>
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>media</td>
<td>“audio”</td>
</tr>
<tr>
<td>port</td>
<td>Port number</td>
</tr>
<tr>
<td>transport</td>
<td>“RTP/AVP”</td>
</tr>
<tr>
<td>fmt list</td>
<td>RTP dynamic payload type (number from 96–127)</td>
</tr>
</tbody>
</table>

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>]
```
Using AMR-NB

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>
```

The AMR-specific parameters that can be received via the `a=fmtp:` line are described in the following table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| octet-align=\texttt{X} | Whether data is octet aligned.  
  - 1 = Octet-aligned (byte-aligned) operation.  
  - 0 (default) = Bandwidth efficient (bit-aligned) operation. |
| mode-set=\texttt{X}     | 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, except when overridden by AMR-NB Codec Mode Requests received in AMR-NB audio from the receiver. |
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<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.</td>
<td>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.</td>
<td>The IP Media Server never initiates a mode change.</td>
</tr>
<tr>
<td></td>
<td>• 1 = Mode changes are allowed only to the neighboring modes in the active codec mode set.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 0 (default) = Change is allowed between any two modes in the active codec mode set.</td>
<td></td>
</tr>
<tr>
<td>crc=X</td>
<td>Whether CRCs are included in the payload.</td>
<td>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></td>
<td>• 1 = Frame CRCs may be included in the payload. crc=1 implies that octet-aligned operation must be used for the session.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 0 (default) = Frame CRCs shall not be included in the payload.</td>
<td></td>
</tr>
<tr>
<td>robust-sorting=X</td>
<td>Whether the IP Media Server supports robust sorting.</td>
<td>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></td>
<td>• 1 = The payload uses robust sorting. robust-sorting=1 implies that octet-aligned operation must be used for the session.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• 0 (default) = The payload uses simple payload sorting.</td>
<td></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>
<td></td>
</tr>
<tr>
<td>channels=</td>
<td>Number of audio channels. (Default: 1.)</td>
<td>The IP Media Server supports only a value of 1.</td>
</tr>
</tbody>
</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).
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 includes 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 System (NFS)

The IP Media Server stores and retrieves data files on the Network File System server (RFC 1094 and RFC 1813).

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 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 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:ann@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:ann@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:ann@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 94). 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 IP Media Server. The following example shows an application having a small audio segment library which consists of the following files:

<table>
<thead>
<tr>
<th>File</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>welcome.wav</td>
<td>Welcome to…</td>
</tr>
<tr>
<td>reached.wav</td>
<td>You have reached…</td>
</tr>
<tr>
<td>hours.wav</td>
<td>Our normal business hours are 9AM to 6PM.</td>
</tr>
<tr>
<td>name.wav</td>
<td>The Museum of Transportation</td>
</tr>
<tr>
<td>closed.wav</td>
<td>The museum is currently closed.</td>
</tr>
<tr>
<td>current.wav</td>
<td>We are now offering an exhibit of vintage motorcycles.</td>
</tr>
<tr>
<td>website.wav</td>
<td>For a preview of this exhibit, visit our Website: <a href="http://www.transport.com">www.transport.com</a>.</td>
</tr>
</tbody>
</table>

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
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:ann@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 this:

```
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.

![Diagram of Announcement Sequence Process]

**Figure 5. 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 96).
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**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI</td>
<td>The complete path for the audio prompt file or servlet, including parameters. Accepts NFS syntax (file:///) or HTTP syntax (http://).</td>
</tr>
</tbody>
</table>

**Examples**

- play=file:///net/nfsserver.carrier.net/prompts/intro.wav
- 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>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 235) for a call flow example.</td>
</tr>
<tr>
<td>no</td>
<td>The announcement plays only after 200 OK.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>For regular announcements, if the announcement file is not found, the IP Media Server will return a SIP 404 Response to the INVITE method.</td>
</tr>
<tr>
<td>no (default)</td>
<td>Usual behavior</td>
</tr>
</tbody>
</table>

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 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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdy</td>
<td>20021015 is spoken as &quot;October Fifteenth Two Thousand Two&quot;.</td>
</tr>
<tr>
<td>dmy</td>
<td>20021015 is spoken as &quot;Fifteen October Two Thousand Two&quot;.</td>
</tr>
<tr>
<td>ymd</td>
<td>20021015 is spoken as “Two Thousand Two October Fifteen”.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndn</td>
<td>is spoken with North American dialing phone number phrasing (NPA-NXX-XXXX), with appropriate pauses.</td>
</tr>
<tr>
<td>gen</td>
<td>is spoken as generic digits (one, five, zero).</td>
</tr>
</tbody>
</table>

**duration**

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3600</td>
<td>is spoken as “one hour”.</td>
</tr>
<tr>
<td>3660</td>
<td>is spoken as “one hour and one minute”.</td>
</tr>
<tr>
<td>3661</td>
<td>is spoken as “one hour, one minute, and one second”.</td>
</tr>
</tbody>
</table>

**money**

Currency value in USD, spoken in dollars and cents.

**Subtype**

USD (Format: $$¢¢)

**Examples**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2500</td>
<td>is spoken as “25 dollars”.</td>
</tr>
<tr>
<td>25</td>
<td>is spoken as “25 cents”.</td>
</tr>
<tr>
<td>1</td>
<td>is spoken as “1 cent”.</td>
</tr>
<tr>
<td>100</td>
<td>is spoken as “one dollar”.</td>
</tr>
<tr>
<td>101</td>
<td>is spoken as “one dollar and one cent”.</td>
</tr>
<tr>
<td>1025</td>
<td>is spoken as “ten dollars and twenty five cents”.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crd</td>
<td>1 is spoken as “one”.</td>
</tr>
<tr>
<td>ord</td>
<td>1 is spoken as “first”.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t12</td>
<td>1700 is spoken as “five p.m.”</td>
</tr>
<tr>
<td>t24</td>
<td>1700 is spoken as “seventeen hundred hours”.</td>
</tr>
</tbody>
</table>

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

- Simple Conferencing
- Advanced Conferencing
- IVR Operations during a Conference
- Simultaneous Play and Record
- MSCML Conferencing Requests
- Coached Conferencing
- MSCML Conferencing Reference

**Note:** If you opt to perform the operations in this chapter, familiarize yourself with the SIP concepts and operations explained in Chapter 2, “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 27 (page 102) 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

The following table 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>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>talker</td>
<td>All participants (legs) are talkers.</td>
</tr>
<tr>
<td>mixmode</td>
<td>full</td>
<td>All audio is mixed.</td>
</tr>
<tr>
<td>toneclamp</td>
<td>no</td>
<td>Audio is not removed (clamped) when the IP Media Server detects certain repetitive, non-voice signals.</td>
</tr>
<tr>
<td>dtmfclamp</td>
<td>no</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 equals 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 232).

The following example shows the call flow for creating a simple conference (normal media).

![Call Flow Diagram](image)

**Figure 6. Call Flow for Simple Conference (Normal Media)**
**Code for Creating a Simple Conference**

The following sample code 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 371) 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:
The reservedtalkers and reserveconfmedia are set in the initial <configure_conference> INVITE message and are the only attributes specified in that message. The reservedtalkers and subscribe for the entire conference can be modified in subsequent <configure_conference> INFO requests directed to the URI representing the conference. The subscribe element can also be sent in the initial INVITE.

The following table describes the attributes and default values for advanced conferences.

### Table 28. Advanced Conferencing Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Description</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserveconfmedia</td>
<td>yes</td>
<td>Allocates resources to play or record audio to or from the entire conference.</td>
<td>140</td>
</tr>
<tr>
<td>reservedtalkers</td>
<td>N/A</td>
<td>Indicates the maximum number of conference participants (legs).</td>
<td>141</td>
</tr>
</tbody>
</table>

The following table describes the subscribe element for advanced conferencing.

### Table 29. Subscribe Element

<table>
<thead>
<tr>
<th>Element</th>
<th>Default Description</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscribe</td>
<td>Empty, meaning no event subscriptions are active.</td>
<td>137</td>
</tr>
</tbody>
</table>

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

### MSCML Attributes and Elements for <configure_leg>

The following table 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 30. Attributes for Participant Legs in Advanced Conference

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Description</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>mixmode</td>
<td>full</td>
<td>Audio from all talker participants is mixed.</td>
<td>139</td>
</tr>
</tbody>
</table>
Table 30. Attributes for Participant Legs in Advanced Conference

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Description</th>
<th>See Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtmfclamp</td>
<td>yes</td>
<td>Detected DTMF digits are removed from audio for all participants.</td>
<td>138</td>
</tr>
<tr>
<td>toneclamp</td>
<td>yes</td>
<td>Audio is removed (clamped) when the IP Media Server detects certain repetitive, non-voice signals.</td>
<td>141</td>
</tr>
<tr>
<td>type</td>
<td>talker</td>
<td>All participants are talker participants.</td>
<td>141</td>
</tr>
</tbody>
</table>

The following table describes the subscribe element for participant legs.

Table 31. Subscribe Element for Participant Legs

<table>
<thead>
<tr>
<th>Element</th>
<th>Default Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscribe</td>
<td>Empty, meaning no event subscriptions are active.</td>
</tr>
</tbody>
</table>
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.

The following table describes the attributes of the `<managecontent>` element.

**Table 32. Attributes for `<managecontent>` in an Advanced Conference**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 src or dest 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:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extension</th>
<th>MIME Type</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

If the content does not match one of the above, the mimetype attribute must be populated.

| fetchtimeout | Maximum time permitted for operation in milliseconds. (Default: 10000 ms.) |
<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"
   </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 server, 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 81).

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

```xml
<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 137.

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

The following example shows the response to the subscription request.
Example 8. Response to Modify Request

```xml
<?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 the following table 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.

The following table describes the advanced conference attributes and their default values in the `<configure_leg>` message.

### Table 33. Configure_Leg Attributes in Advanced Conference

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtmfclamp</td>
<td>{yes</td>
<td>no}</td>
<td>Yes</td>
</tr>
<tr>
<td>mixmode</td>
<td>{full</td>
<td>private</td>
<td>preferred</td>
</tr>
<tr>
<td>type</td>
<td>{talker</td>
<td>listener</td>
<td>internal}</td>
</tr>
<tr>
<td>toneclamp</td>
<td>{yes</td>
<td>no}</td>
<td>Yes</td>
</tr>
</tbody>
</table>
The following example 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
<?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. The following example shows a configure_leg request that mutes the leg.

**Example 10. Modifying a Conference Leg**

```xml
<?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 248).

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. The following example shows how the XML is generated by the active talker event.

Example 11. Notification of Active Talker Events

```xml
<?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 124)
- 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

IP Media Server gives application developers greater control over the media mix that is delivered to conference participants. Specifically, a lecture mode feature is available which specifies 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 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 139) 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 the following example) 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
<?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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Do not report anything (disable reporting if enabled).</td>
</tr>
</tbody>
</table>
The maskdigits values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Disables Clear Text Logging. DTMF data that is normally written into the IP Media Server log files is masked (replaced with asterisks) or discarded.</td>
</tr>
<tr>
<td>No (default)</td>
<td>Enables Clear Text Logging.</td>
</tr>
</tbody>
</table>

Example 13. Keypress Response Attributes and Values

```xml
<?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 (in the previous example) contains the following attributes (and the <status> element):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>digit</td>
<td>Digit that was detected. Possible values are: 0–9, A–D, #, *</td>
</tr>
<tr>
<td>length</td>
<td>Digit length. Values: standard, long.</td>
</tr>
<tr>
<td>method</td>
<td>Method used to collect digits. Values:</td>
</tr>
<tr>
<td></td>
<td>* standard - normal digits</td>
</tr>
<tr>
<td></td>
<td>* long - long duration digits (more than 1 second)</td>
</tr>
</tbody>
</table>
The `<status>` element has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interdigittime</td>
<td>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.</td>
</tr>
</tbody>
</table>

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
<?xml version="1.0"?>
```
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.

The following example shows the busy event subscription, response and event notification process.

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

**Request**

```xml
<?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>
```

**Response**

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

**Notification**

```xml
<?xml version="1.0"?>
<MediaServerControl version="1.0">
    <!-- notification message -->
</MediaServerControl>
```
Un-subscribe to busy events and response request

```xml
<?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
<?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 136). This reference is an alphabetical listing in tabular format.
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
<?xml version='1.0'?>
<MediaServerControl version='1.0'>
  <request>
    <playrecord
        recurl='file:///var/snowshore/rec/temp.ulaw'
        recencoding='ulaw' init silence="infinite"
        end silence="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 ‘init silence="infinite”’ and ‘end silence="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 371).

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.

The following table describes the attributes for an <configure_conference> element in an INVITE message.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reservedtalkers</td>
<td>2-available IP Media Server capacity</td>
<td></td>
<td>Specifies the number of legs allocated when the conference is created.</td>
</tr>
</tbody>
</table>
The following table describes the subscribe element.

**Table 35. Subscribe Element in INVITE with Configure_Conference**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscribe</td>
<td>Used when subscribing and unsubscribing to active talker events.</td>
</tr>
</tbody>
</table>

**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.

The following table describes the attributes for an `<configure_conference>` element in an INFO message.

**Table 36. Configure_Conference Attributes in INFO Message**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reservedtalkers</td>
<td>2-available IP Media Server capacity</td>
<td>Specifies the number of legs allocated when the conference is created.</td>
</tr>
</tbody>
</table>
The following table describes the subscribe element.

Table 37. Subscribe Element in INFO with Configure_Conference

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscribe</td>
<td>Used when subscribing and unsubscribing to active talker events.</td>
</tr>
</tbody>
</table>

**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.

The following table lists and describes the attributes for the configure_leg element.

Table 38. Configure_Leg Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtmfclamp</td>
<td>{yes</td>
<td>no}</td>
<td>Yes</td>
</tr>
<tr>
<td>mixmode</td>
<td>{full</td>
<td>private</td>
<td>preferred</td>
</tr>
<tr>
<td>type</td>
<td>{talker</td>
<td>listener</td>
<td>internal}</td>
</tr>
<tr>
<td>toneclamp</td>
<td>{yes</td>
<td>no}</td>
<td>Yes</td>
</tr>
<tr>
<td>outputgain</td>
<td>Integer between -12dB and 18dB.</td>
<td></td>
<td>The attribute value is an integer indicating a decibel level change. For example +10 is louder. -10 is lower. 0 is no change.</td>
</tr>
</tbody>
</table>
The following table describes the subscribe element.

**Table 39. Subscribe Element for Configure_Leg**

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscribe</td>
<td>Used when subscribing and unsubscribing to asynchronous DTMF events.</td>
</tr>
</tbody>
</table>
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 125) and “configure_leg” (page 127).

**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.

The following table describes the attributes for the `configure_team` element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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. |
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.

The following table summarizes how MSCML elements and attributes are used to create a coached conferencing configuration and the expected results.

### Configuring a Coached Conference

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>alphanumeric characters; must be less than 64</td>
<td>unique ID (within the conference)</td>
<td>Identifies each participant uniquely in a coached conference.</td>
</tr>
</tbody>
</table>

**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.

### Table 41. MSCML Elements and Attributes for Creating Coached Conferencing

<table>
<thead>
<tr>
<th>Participant</th>
<th>ID</th>
<th>Team Members</th>
<th>Mixmode</th>
<th>Hears</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervisor</td>
<td>supervisor</td>
<td>Agent</td>
<td>Private</td>
<td>Customer + Agent</td>
</tr>
<tr>
<td>Agent</td>
<td>agent</td>
<td>Supervisor</td>
<td>Full</td>
<td>Customer+ Supervisor</td>
</tr>
<tr>
<td>Customer</td>
<td>customer</td>
<td>None</td>
<td>Full</td>
<td>Agent</td>
</tr>
</tbody>
</table>

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
<?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
<?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:

<table>
<thead>
<tr>
<th>Participant</th>
<th>ID</th>
<th>Team Members</th>
<th>Mixmode</th>
<th>Hears</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervisor</td>
<td>supervisor</td>
<td>None</td>
<td>Private</td>
<td>Silence</td>
</tr>
</tbody>
</table>

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

<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:

<table>
<thead>
<tr>
<th>Participant</th>
<th>ID</th>
<th>Team Members</th>
<th>Mixmode</th>
<th>Hears</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervisor</td>
<td>supervisor</td>
<td>Agent</td>
<td>Private</td>
<td>Agent</td>
</tr>
<tr>
<td>Agent</td>
<td>agent</td>
<td>Supervisor</td>
<td>Full</td>
<td>Supervisor</td>
</tr>
</tbody>
</table>

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
<?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
<?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:

<table>
<thead>
<tr>
<th>Participant</th>
<th>ID</th>
<th>Team Members</th>
<th>Mixmode</th>
<th>Hears</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervisor</td>
<td>supervisor</td>
<td>Agent</td>
<td>Private</td>
<td>Customer + Agent</td>
</tr>
<tr>
<td>Agent</td>
<td>agent</td>
<td>Supervisor</td>
<td>Full</td>
<td>Customer+ Supervisor</td>
</tr>
<tr>
<td>Customer</td>
<td>customer</td>
<td>None</td>
<td>Full</td>
<td>Agent</td>
</tr>
</tbody>
</table>

**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. The following example shows that there are two members on the team, including the requester.

**Example 15. Shows Two Members on the Team**

```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"/>
        </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 260).
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).

Note: The values for the specific call progress tones are listed under the Type attribute.

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

dtmf

Allows the IP Media Server to send DTMF and/or RFC2833 packets provided in the digits attribute.

Parent: prompt

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>digits</td>
<td>The numbers to play back.</td>
</tr>
<tr>
<td>ontime</td>
<td>The amount of time (in milliseconds) that each digit is held.</td>
</tr>
</tbody>
</table>

Example

```xml
<?xml version="1.0" encoding="utf-8"?><MediaServerControl
version="1.0"><request><play><prompt><dtmf digits="123"
ontime="200" /></prompt></play></request> </MediaServerControl>
```
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

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
Used to subscribe for notification of specific call progress tones.
For example, the IP Media Server detects an in-band busy signal and reports this event to the application, so that false connects are eliminated.

Note: The signal element is also used in the IP Media Server’s implementation for Interactive Voice Response (IVR).

Parent: event, notification
Values: busy*, dial*, CED, CNG, PVD_PAMD, PVD, PAMD, interrupt, ring*, 400
* 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**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>Adds a teammate.</td>
</tr>
<tr>
<td>delete</td>
<td>Deletes a teammate.</td>
</tr>
<tr>
<td>query (default)</td>
<td>Returns the teammate list.</td>
</tr>
<tr>
<td>set</td>
<td>Creates a team list when followed by <code>&lt;teammate id=&quot;n&quot;&gt;</code> 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.</td>
</tr>
</tbody>
</table>

**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 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.

**Attribute of:** configure_leg

**Values:**

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

**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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>full (default)</td>
<td>Audio is mixed, but not as the loudest leg.</td>
</tr>
<tr>
<td>preferred</td>
<td>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.</td>
</tr>
<tr>
<td>parked</td>
<td>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 &lt;configure_leg&gt; request.</td>
</tr>
<tr>
<td>private</td>
<td>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.</td>
</tr>
<tr>
<td>mute_video</td>
<td>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.</td>
</tr>
<tr>
<td>mute_audio</td>
<td>Suppresses only the audio from the participant and allows the video to be received from a participant.</td>
</tr>
<tr>
<td>lecture</td>
<td>Makes that leg the lecturer for the conference. All non-lecture participants have the audio and video muted.</td>
</tr>
<tr>
<td>lecture_video</td>
<td>Makes that leg the video lecturer for the conference. All non-lecture participants have the video muted once any participant becomes the lecturer_video.</td>
</tr>
<tr>
<td>lecture_audio</td>
<td>Makes that leg the audio lecturer for the conference. All non-lecture participants have the audio muted once any participant becomes lecture_audio.</td>
</tr>
</tbody>
</table>

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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>Enables reporting of the associated event.</td>
</tr>
<tr>
<td>no</td>
<td>Disables reporting of the associated event.</td>
</tr>
</tbody>
</table>

**Attribute of:** keypress

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>Reports normal digits.</td>
</tr>
<tr>
<td>long</td>
<td>Reports long digits.</td>
</tr>
<tr>
<td>both</td>
<td>Reports both manual and long digits.</td>
</tr>
<tr>
<td>none</td>
<td>Disables asynchronous DTMF reporting.</td>
</tr>
</tbody>
</table>

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

**Attribute of:** <configure_conference>

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes (default)</td>
<td>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.</td>
</tr>
<tr>
<td>no</td>
<td>Does not play to or record the conference.</td>
</tr>
</tbody>
</table>
**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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes (default)</td>
<td>Audio is clamped when tones are detected.</td>
</tr>
<tr>
<td>no</td>
<td>Audio is not clamped.</td>
</tr>
</tbody>
</table>

**type**

Attribute of `config_leg` and `signal`.

The type of participant for this conference leg.

**Attribute of:** `<configure_leg>`

**Values:**

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

**Attribute for:** `signal` (required)

Identifies the type of signal to be reported.
### Values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>Far end busy tone (also fast busy)</td>
</tr>
<tr>
<td>dial</td>
<td>Dial tone</td>
</tr>
<tr>
<td>CED</td>
<td>Fax calling tone</td>
</tr>
<tr>
<td>CNG</td>
<td>Fax called station identifier</td>
</tr>
<tr>
<td>PVD_PAMD</td>
<td>Registering for the PVD_PAMD signaling detection registers the application for both Positive Voice Detection (PVD) and Positive Answering Machine Detection (PAMD). These do not need to be turned off. The detector runs until either PVD or PAMD is detected and then the PVD_PAMD detection is automatically disabled. An MSCML application does not need to stop the call progress signal detection prior to the completion of the call.</td>
</tr>
<tr>
<td>PVD</td>
<td>The IP Media Server sends notification regarding the PVD. Detection of the event stops the detector.</td>
</tr>
<tr>
<td>PAMD</td>
<td>The IP Media Server sends notification regarding the PAMD. Detection of the event stops the detector.</td>
</tr>
<tr>
<td>interrupt</td>
<td>Call interrupted (call waiting)</td>
</tr>
<tr>
<td>ring</td>
<td>Far end ring tone</td>
</tr>
<tr>
<td>400</td>
<td>tone used in Japan</td>
</tr>
</tbody>
</table>
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 more information about VoiceXML, see “VoiceXML Version 2.0 and Dialog Service” (page 176), and “VoiceXML Version 1.0 and Dialog Service” (page 326).

This chapter includes the following major sections:

- IVR Service
- Playing Announcements
- Collecting DTMF Digits
- Recording Audio
- Stopping an IVR Request in Progress
- MSCML MRCP Session Management
- MSCML IVR Reference
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 draft-ietf-sip-rfc2543bis-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 173) 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/> or <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 81) 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 371) for the complete MSCML grammar.

See “Sample Code and Call Flows” (page 232) for code samples and call flow diagrams.

**Call Progress Tones**

The IP Media Server supports the following call progress tones in MSCML to allow call centers to quickly classify calls as successful or not.

   - busy
   - dial
   - CED
   - CNG
   - PVD_PAMD
   - interrupt
   - ring
   - 400

**Note:** These call progress tones in MSCML can also be used in the IP Media Server’s conferencing implementations.

Third party systems can subscribe for these call progress events early in the call setup to quickly detect any of the call progress signal/tones.

The type attribute value of PVD_PAMD is used to register for both PVD and PAMD notification events.
The following is an example MSCML subscription for PVD and PAMD notification events.

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

The following is an example MSCML signal notification.

```xml
<?xml version="1.0"?>
<MediaServerControl version="1.0">
  <notification>
    <signal type="PVD"/>
  </notification>
</MediaServerControl>
```
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 153)

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

For further details concerning IVR attributes, see “IVR Attributes” (page 163).

Responses

When the announcement has finished playing, the 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 173).
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 161).

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.

```xml
<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 153).

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 extradigittimer 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 173).
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 153) 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 42 describes these timing attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>initsilence</td>
<td>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).</td>
</tr>
<tr>
<td>endsilence</td>
<td>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 IP Media Server trims the end of the recorded audio by an amount equal to the endsilence parameter.</td>
</tr>
</tbody>
</table>

Additional Attributes

Table 43 describes additional attributes associated with <playrecord>.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Whether the recording overwrites or appends.</td>
</tr>
<tr>
<td>recencoding</td>
<td>Specifies the content codec to use.</td>
</tr>
<tr>
<td>duration</td>
<td>Specifies the time in ms for the entire recording.</td>
</tr>
<tr>
<td>beep</td>
<td>Specifies whether a beep signifies the start of recording.</td>
</tr>
</tbody>
</table>

For details on all IVR attributes, see “IVR Attributes” (page 163).

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:
Recording Audio

- 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 173).

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>The IP Media Server stops the request if a fetch error occurs.</td>
</tr>
<tr>
<td>no (default)</td>
<td>Original MSCML behavior; fetch errors do not stop the request. For compatibility reasons, this is the default.</td>
</tr>
</tbody>
</table>

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 MRCP Session Management

This section describes MRCP Session Management and includes the following topics:

- Overview of MRCP Session Management
- MSCML Requests
- Create session request format
- Create session response format
- Terminate session request format
- Terminate session response format
- Sample Call Flow

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 7 outlines the MRCP Session Management process. Each stage is explained in the table following the figure.

![Figure 7. MRCP Session Management](image-url)
Table 44. Description of MRCP Session Management Topology

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIP session between a caller and the application server that invokes the application.</td>
</tr>
<tr>
<td>2</td>
<td>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 <code>&lt;mrcp_session_create&gt;</code> and <code>&lt;mrcp_session_terminate&gt;</code> requests.</td>
</tr>
<tr>
<td>3</td>
<td>RTP session with caller. This is normally full duplex, as shown.</td>
</tr>
<tr>
<td>4</td>
<td>SIP session between the IP Media Server and the ASR/TTS server, which supports MRCP V2.</td>
</tr>
<tr>
<td>5</td>
<td>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.</td>
</tr>
<tr>
<td>6</td>
<td>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 <code>&lt;mrcp_session_create&gt;</code> 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.</td>
</tr>
</tbody>
</table>

**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.

**Note:** MSCML `mrcp_session_create` works only with the conference service.

**Create session request format**

The following is the MSCML format of the `mrcp_session_create` request.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<MediaServerControl version="1.0">
  <request>
    <mrcp_session_create id="1234" resource="speechrecog"
      url="sip:mrcpserver_address" proto="TCP/MRCPv2"
      rtpdirection="sendonly"/>
  </request>
</MediaServerControl>
```

The following are the attributes of the `mrcp_create_session` request.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>No</td>
<td>An application-provided transaction identifier.</td>
</tr>
<tr>
<td>resource</td>
<td>Yes</td>
<td>The resource to be provided by the MRCP server.</td>
</tr>
<tr>
<td>url</td>
<td>Yes</td>
<td>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.</td>
</tr>
<tr>
<td>proto</td>
<td>No</td>
<td>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.</td>
</tr>
<tr>
<td>rtpdirection</td>
<td>No</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 46. Attributes of mrcp_session_response

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>No</td>
<td>An application-provided transaction identifier.</td>
</tr>
<tr>
<td>connection</td>
<td>Yes</td>
<td>The IP address of the MRCP server the client connects to.</td>
</tr>
<tr>
<td>port</td>
<td>Yes</td>
<td>The port of the MRCP server that the client connects to.</td>
</tr>
<tr>
<td>proto</td>
<td>Yes</td>
<td>The transport protocol to be used for the MRCP session. Alternatives are &quot;TCP/MRCPv2&quot; and &quot;TCP/TLS/MRCPv2&quot;.</td>
</tr>
<tr>
<td>connectiontype</td>
<td>Yes</td>
<td>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.</td>
</tr>
<tr>
<td>channel</td>
<td>Yes</td>
<td>The channel identifier returned from the MRCP server. The client uses it to communicate with the server.</td>
</tr>
</tbody>
</table>

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 47. Attributes of mrcp_terminate_session_request

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>No</td>
<td>An application-provided transaction identifier.</td>
</tr>
<tr>
<td>channel</td>
<td>Yes</td>
<td>The channel identifier returned from the MRCP server. The client uses it to communicate with the server.</td>
</tr>
</tbody>
</table>

Terminate session response format

```xml
<?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

For MRCP sample call flow, see “Call Flow for MSCML MRCP Session Management” (page 283).
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 371).

IVR Elements

Table 48 summarizes MSCML elements that support IVR.

Table 48. MSCML Elements for IVR

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Response Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>managecontent</td>
<td>src, dest, action, httpmethod, name, mimetype, fetchtimeout</td>
<td>N/A</td>
<td>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., &lt;playrecord/&gt;). There is no provision for streaming the recording to the Web server while it is being recorded.</td>
</tr>
<tr>
<td>play</td>
<td>delay, duration, id, locale, offset, prompt, promptencoding, repeat, stop_on_error</td>
<td>code, id, playduration, reason, text</td>
<td>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 &lt;prompt&gt; element in the body of the request.</td>
</tr>
<tr>
<td>playcollect</td>
<td>barge, clearedigits, delay, duration, escapekey, extradigittimer, firstdigittimer, id, interdigittimer, locale, maskdigits, maxdigits, offset, prompt, promptencoding, repeat, returnkey</td>
<td>code, digits, id, playduration, reason, text</td>
<td>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.</td>
</tr>
<tr>
<td>playrecord</td>
<td>barge, beep, clearedigits, delay, duration, endsilence, escapekey, id, insilence, locale, mode, offset, prompt, promptencoding, recencoding, recstopmask, recurl, repeat</td>
<td>code, digits, id, playduration, reason, reclength, text</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

**Note:** The play queue that can be encased within the `<prompt>` and `</prompt>` keywords can only be 16 files deep. If you send a play request with more than 16 files, only the first 16 are played and a successful return code is generated.

**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**

```xml
<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**

```xml
<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**

```xml
<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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes (default)</td>
<td>DTMF digit detection interrupts the prompt.</td>
</tr>
</tbody>
</table>

**Note:** Any digit barges in. There is no barge mask.
**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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes (default)</td>
<td>The IP Media Server plays a tone to signify the start of a recording. The beep is an 880 Hz tone 250ms in duration.</td>
</tr>
<tr>
<td>no</td>
<td>No tone is played at the start of a recording.</td>
</tr>
</tbody>
</table>

**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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>Digits are flushed.</td>
</tr>
<tr>
<td>no (default)</td>
<td>Digits are not flushed.</td>
</tr>
</tbody>
</table>

**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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>µlaw</td>
<td>G.711 µlaw encoding.</td>
</tr>
<tr>
<td>alaw</td>
<td>G.711 alaw encoding.</td>
</tr>
</tbody>
</table>

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.

**endwarning**

Number of milliseconds at the end of the recording to play warnurl.

**Attribute for:** playrecord

**Values:** a number in milliseconds.

**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=notdigit.

**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: 3000 ms.) 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:

<table>
<thead>
<tr>
<th>Value (default)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>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 &lt;?Sensitive Data?&gt; in the log files.</td>
</tr>
<tr>
<td>No</td>
<td>DTMF data is recorded and written to the IP Media Server log files.</td>
</tr>
</tbody>
</table>
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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>overwrite</td>
<td>Recording overwrites the contents of the URL.</td>
</tr>
<tr>
<td>(default)</td>
<td></td>
</tr>
<tr>
<td>append</td>
<td>Recording is appended to the contents of the URL.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulaw</td>
<td>G.711 µ-law encoding.</td>
</tr>
<tr>
<td>alaw</td>
<td>G.711 a-law encoding.</td>
</tr>
<tr>
<td>msgsm</td>
<td>MSGSM</td>
</tr>
</tbody>
</table>

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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulaw</td>
<td>G.711 µ-law encoding. (default)</td>
</tr>
<tr>
<td>alaw</td>
<td>G.711 a-law encoding.</td>
</tr>
<tr>
<td>ms_gsm</td>
<td>MSGSM</td>
</tr>
</tbody>
</table>

**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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>Enables reporting of the associated event.</td>
</tr>
<tr>
<td>no</td>
<td>Disables reporting of the associated event.</td>
</tr>
</tbody>
</table>

**Attribute for:** keypress

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>Reports normal digits.</td>
</tr>
<tr>
<td>long</td>
<td>Reports long digits.</td>
</tr>
<tr>
<td>both</td>
<td>Reports both manual and long digits.</td>
</tr>
<tr>
<td>none</td>
<td>Disables asynchronous DTMF reporting.</td>
</tr>
</tbody>
</table>

**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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>The IP Media Server stops the request if a fetch error occurs.</td>
</tr>
</tbody>
</table>
### subtype

Minor or subtype values describing how to render a variable as audio.

**Attribute for:** variable

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mdy 20021015</td>
<td>“October Fifteenth Two Thousand Two”</td>
</tr>
<tr>
<td>ymd 20021015</td>
<td>“Two Thousand Two October Fifteen”</td>
</tr>
<tr>
<td>ndn</td>
<td>North American dialing phone number phrasing (NPA-NXX-XXXX), with appropriate pauses</td>
</tr>
<tr>
<td>dmy</td>
<td>20021015 is spoken as “Fifteen October Two Thousand Two”.</td>
</tr>
<tr>
<td>t12 1700</td>
<td>“Five p.m.”</td>
</tr>
<tr>
<td>t24 1700</td>
<td>“Seventeen hundred hours”</td>
</tr>
<tr>
<td>gen</td>
<td>generic digits (for example: “one”, “five”, “zero”)</td>
</tr>
<tr>
<td>crd</td>
<td>“one”</td>
</tr>
<tr>
<td>ord</td>
<td>“first”</td>
</tr>
</tbody>
</table>

For detailed descriptions of subtype values, see “Variable Types and Subtypes” (page 96).

### type

Type is an attribute for signal and variable.

**Attribute for:** signal (required)

Identifies the type of signal to be reported.

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>Far end busy tone (also fast busy)</td>
</tr>
<tr>
<td>dial</td>
<td>Dial tone</td>
</tr>
<tr>
<td>CED</td>
<td>Fax calling tone</td>
</tr>
<tr>
<td>CNG</td>
<td>Fax called station identifier</td>
</tr>
</tbody>
</table>
Attribute for: variable (required)
Identifies the type of variable to be rendered as audio.
Values: date, digit, duration, money, month, number, silence, string, time, weekday.
For detailed descriptions of subtype values, see “Variable Types and Subtypes” (page 96).

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)

warnurl
Specifies the target URL for the warning message.
Attribute for: playrecord
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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>Command completed.</td>
</tr>
<tr>
<td>400</td>
<td>For playrecord: command not accepted due to error. The text attribute describes the cause of the error.</td>
</tr>
<tr>
<td>501</td>
<td>For playrecord: error because the URL type specified was not supported.</td>
</tr>
</tbody>
</table>

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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOF</td>
<td>The play was completed when the end of file was reached.</td>
</tr>
<tr>
<td>stopped</td>
<td>A stop command, another command, or a SIP re-INVITE with hold media stopped the play.</td>
</tr>
</tbody>
</table>

**Response for:** playcollect

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>match</td>
<td>A digit was detected.</td>
</tr>
<tr>
<td>timeout</td>
<td>No digit was received in time.</td>
</tr>
<tr>
<td>returnkey</td>
<td>The return key terminated the operation.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>escapekey</td>
<td>The escape key terminated the operation.</td>
</tr>
<tr>
<td>interrupted</td>
<td>Another request was received before the current one could finish.</td>
</tr>
<tr>
<td>stopped</td>
<td>A stop command, another command, of a SIP re-INVITE with hold media stopped the play.</td>
</tr>
</tbody>
</table>

**Response for:** playrecord

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>digit</td>
<td>A digit was detected.</td>
</tr>
<tr>
<td>endsilence</td>
<td>Trailing silence was detected.</td>
</tr>
<tr>
<td>init_silence</td>
<td>No voice (initial silence) was detected.</td>
</tr>
<tr>
<td>max_duration</td>
<td>Time for recording was complete.</td>
</tr>
<tr>
<td>stopped</td>
<td>A stop command, another command, or a SIP re-INVITE with hold media stopped the play.</td>
</tr>
<tr>
<td>escapekey</td>
<td>The escape key was entered either in the play mode or in the record mode. The application terminated the recording.</td>
</tr>
<tr>
<td>error</td>
<td>The operation failed.</td>
</tr>
</tbody>
</table>

**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 Version 2.0 and Dialog Service

This chapter explains the basics of VoiceXML (VXML) Version 2.0, lists the supported VoiceXML elements and attributes, and describes ECMAScript language functionality.

The dialog service is one of two interfaces that the IP Media Server offers for developing IVR/DTMF/Voice 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 chapter includes the following sections:

◆ About VoiceXML
  ▪ VoiceXML Interpreter
  ▪ Dialog Service Indicator and Request URI
  ▪ VoiceXML Launcher
  ▪ Default Script
  ▪ VoiceXML Concepts
  ▪ VoiceXML Application and Its Documents
  ▪ File Storage and Retrieval
  ▪ Media Content Recovery Extension
◆ VoiceXML Elements Reference
◆ VoiceXML Properties
◆ ECMAScript Functionality
◆ VoiceXML Extended Session Variables
◆ RTP Codec Selection Using the <transfer> Element
◆ Image Overlay and Text Overlay Functionality
About VoiceXML

VoiceXML is a W3C standard scripting language for playing text to speech and audio prompts, and for collecting DTMF and voice input.

The interpreter executes VoiceXML dialogs on an RTP stream. 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.

VoiceXML Interpreter

The IP Media Server includes interpreters for VoiceXML 1.0 and VoiceXML 2.0. 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 IP Media Server executes VoiceXML documents to offer IVR scripting with DTMF and voice 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;
```

**With Resource Parameter:**

```
INVITE sip:dialog@MS_IP;voicexml=http://app1.carrier.com/path/script1.vxml;resource=asr+tts
```

The VoiceXML 2.0 browser also supports an optional resource parameter. It indicates to the browser which MRCP resources are required by the VoiceXML script. It supports the values, asr, tts, and asr+tts to indicate if an ASR, or a TTS, or both an ASR and TTS resource are needed by the script respectively. Providing this parameter ensures the VoiceXML browser attempts to reserve the specified resources as the call is being established as opposed to the browser requesting the resource as needed in the middle of the dialog.

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://appl.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://appl.carrier.com/cgi/bin/genvxml.pl%3Fsubscriberid%3D34590087

The SIP URI must be escaped, as described in “Syntax and Escaping” (page 78).

VoiceXML Launcher

If a SIP INVITE to the IP Media Server is directed to the dialog service but does not specify a file (voicexml=), then the 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.

Default Script

The IP Media Server supports the following default script which is loaded prior to the VoiceXML script supplied in the SIP URI:

/opt/snowshore/htdocs/defaults.xml

This script provides several default properties, default <catch> blocks, and prompts that application developers can modify for their own deployment.

Please use caution when modifying the script because it affects how the VoiceXML browser behaves. Refer to the W3C Voice Extensible Markup Language specification for detailed information.

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 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, NFS (Network File System), or RTSP (Real Time Streaming Protocol).

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’s use of root documents.

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 8 shows how a root document is shared in a VoiceXML application.

![Figure 8. Sharing of Root Document in VoiceXML Application](image)

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 IP Media Server context for playing audio). If a script requires only playing recorded prompts and uses DTMF recognition, you do not need an ASR and/or a TTS resource.
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.

Scripting

The IP Media Server supports ECMA tags that involve the scripting capabilities of the VoiceXML interpreter. See “ECMAScript Functionality” (page 220).

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, limit the use of ECMAScripting on the IP Media Server.

Scripting Guidelines

The following guidelines provide good practices for writing VoiceXML scripts.

- Defining grammars is the first step in the MRCP recognition process. Once you define a grammar, VXML tags such as *filled* are used to trigger the MRCP recognition processing.
- You can define grammars in the VXML script or they can be pre-compiled, stored on the MRCP server, and referenced with a URI. Pre-compiled grammar improves the performance of the application.
- Tuning the grammar is a key step in getting the best performance from a VXML script. Vendors offer commercial tools to help analyze and refine grammars.
- You can define grammar as part of the recognition command but there is a performance impact.
- Finding a grammar command in a VXML script triggers the creation of a MRCP ASR session when the VXML script is loaded and parsed.
- VoiceXML 2.0 requires that a grammar be either a voice or a DTMF grammar. A single grammar can not be both voice and DTMF. A script can have multiple grammars. VXML 2.1 will allow a single grammar to be both voice and DTMF.

DTMF Grammars Guidelines

The IP Media Server locally supports DTMF grammars. The following are some guidelines and limitations associated with this support:

<grammar>

- DTMF SRGS grammars must have the type attribute set to application/srgs+xml and a mode attribute set to dtmf.
- The IP Media Server supports the semantics/1.0 value for the tag-format attribute of DTMF SRGS grammars.
SRGS DTMF grammars can use <tag> elements for semantic interpretation. However only one <tag> element will be used for semantic interpretation for a matched grammar.

The format of the <tag> content should be as follows:

\[
\text{<tag>maincourse='pizza';</tag>}
\]

Where maincourse is the variable to be filled.

The IP Media Server does not support SRGS DTMF grammars that have external references. Specifically this means that the ‘uri’ attribute of the <ruleref> element is only supported if it starts with the character ‘#’. The ‘#’ character indicates that the ‘uri’ is a reference to a <rule> in the current SRGS grammar that is being parsed.

The ‘special’ attribute of the <ruleref> element is not supported. Equivalent behavior can be supported with the use of the <item> element’s repeat attribute.

DTMF digit content as direct descendant of the <rule> element is not supported. The equivalent grammar behavior can be accomplished by having an <item> element that contains the DTMF digit content as the direct descendant to the <rule> element.

MRCP

VoiceXML 2.0 supports Media Resource Control Protocol (MRCP) version 2.0. MRCP controls media service resources such as speech synthesizers, recognizers, and verifiers over a network. This protocol is designed to work with signaling protocol SIP which is used to establish control connections to MRCP servers. Requests are then made to the MRCP server to stream media or to perform recognition on specific media streams set to it. Media streamed both to and from the MRCP server is sent as RTP (Real Time Protocol) packets. MRCP integrates speech recognition and text to speech engines from Nuance eliminating the need for proprietary APIs. 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. Recognition and text to speech capabilities can be implemented using VXML 2.0 scripts using the Dialogic® VoiceXML 2.0 integrated with MRCP 2.0.

Session Variables

VoiceXML provides a facility, called session variables, to pass signaling information to VoiceXML scripts. The IP Media Server supports the standard VoiceXML session variables as well as extensions which capture additional information from the SIP INVITE. See “VoiceXML Extended Session Variables” (page 223) 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 VoiceXML 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 2.0 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.

Refer to the `<data>` element for detailed information.
VoiceXML Elements Reference

This section describes the VoiceXML elements (tags) supported by the IP Media Server.

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the variable being assigned to a value. Required.</td>
</tr>
<tr>
<td>expr</td>
<td>ECMAScript expression that, when evaluated, is assigned as the new value of the variable. It must be a valid ECMA value. Required.</td>
</tr>
</tbody>
</table>

<audio>
Plays pre-recorded audio, video or multimedia, and accesses the media file 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.

<table>
<thead>
<tr>
<th>Extension</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulaw</td>
<td>G.711µ-law</td>
</tr>
<tr>
<td>alaw</td>
<td>G.711 A law</td>
</tr>
<tr>
<td>msgsm</td>
<td>Microsoft® GSM</td>
</tr>
<tr>
<td>3gp or 3gpp</td>
<td>3GPP</td>
</tr>
<tr>
<td>3g2 or 3gpp2</td>
<td>3GPP2</td>
</tr>
</tbody>
</table>

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 children listed below.
Parents: <audio>, <emphasis>, <enumeration>, <paragraph>, <p>, <prompt>, <prosody>, <sentence>, <s>, <speak>, <voice>
Children:  <audio>, <catch>, <emphasis>, <mark>, <paragraph>, <p>, <phonename>, <prosody>, <say-as>, <sentence>, <s>, <sub>, <voice>

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>String literal containing the URI for the audio. Required.</td>
</tr>
</tbody>
</table>

Note: The pre-recorded audio identified by the “src” attribute is played using the IP Media Server resources unless it is a child of any of the listed parent to the TTS server for processing.

<b>block</b>

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 children listed below.

Parent:  <form>

Children:  <assign>, <audio>, <clear>, <disconnect>, <exit>, <goto>, <if>, <log>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>expr</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the form-item variable used to track whether this block can be executed. Defaults to an inaccessible internal variable.</td>
</tr>
</tbody>
</table>

<catch>

Defines an event handler within the current scope.

The value can be empty or can contain one or more of the children listed below.

Parents:  <field>, <form>, <initial>, <menu>, <record>, <subdialog>, <transfer> <vxml>
**Children:**  
<audio>, <clear>, <data>, <disconnect>, <enumerate>,  
<exit>, <foreach>, <goto>, <if>, <log>, <prompt>, <reprompt>,  
$return$, <script>, <submit>, <throw>, <value>, <var>,

**Attributes:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Event or events to catch. If the attribute is not provided, all events should be caught.</td>
</tr>
</tbody>
</table>
| 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.  
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. |
| 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 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.

**Parent:**  
<menu>

**Children:**  
<audio>, <grammar>, <say-as>, <value>

**Attributes:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtmf</td>
<td>DTMF sequence for the choice.</td>
</tr>
</tbody>
</table>
| accept    | Overrides the setting for the accept attribute in <menu>. Optional. (Defaults to exact.)  
- exact - The text contained in the <choice> element defines the exact phrase to be recognized.  
- approximate - The text contained in the <choice> element defines an approximate recognition phrase. A subset of the words in the phrase can be matched. For example, "hello world" can be matched with "hello world", "hello", or "world". |
| event     | An event to be thrown instead of going to ext. |
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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namelist</td>
<td>List of items to clear. (When not specified, the application clears all form items in the current form.) Required.</td>
</tr>
</tbody>
</table>

**<content>**

Specifies the content of an overlay used in a video stream. For examples of overlays, see “Image Overlay and Text Overlay” (page 308). For more on overlays, see “Image Overlay and Text Overlay Functionality” (page 227).

**Parents:**  <overlay>

**Children:**  <font>, <scroll>

**Attributes:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>Source URL of the image or text file. For image overlays, you must specify the source URL. For text overlays, you can specify this attribute or omit this attribute and specify inline content (that is, text strings placed in the element). One or the other must be specified. For example, to use GB18030 encoding, you must specify a local file:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;content src=&quot;file:///opt/snowshore/htdocs/vxml/demos/media/pizza.gb&quot; encoding=&quot;GB18030&quot;&gt;</code></td>
</tr>
<tr>
<td>type</td>
<td>Used with src. The MIME type of the src. Default value is plain/text, indicating text overlay. For image overlays, values are: image/png, image/jpeg.</td>
</tr>
<tr>
<td>encoding</td>
<td>The encoding type of the text. Values are: UTF8, ASCII, or GB18030. Default value is ASCII. This attribute is ignored for image overlays.</td>
</tr>
<tr>
<td>halign</td>
<td>Horizontal alignment of the text in the overlay area. Values are: left, right, and center. Default value is left. This attribute is ignored for image overlays.</td>
</tr>
<tr>
<td>wordwrap</td>
<td>Specifies whether to wrap text to the next line in the overlay area. Values are: on or off. Default value is off. This attribute is ignored for image overlays.</td>
</tr>
</tbody>
</table>
<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, fetchhint, 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.

Children: None.

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namelist</td>
<td>See above.</td>
</tr>
</tbody>
</table>

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>, <if>, <noinput>,
<nomatch>

Children: None
Attributes: None.

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
</tbody>
</table>

<enumerate>
The <enumerate> element is an automatically generated description of the choices available to the user. It specifies a template that is applied to each choice in the order they appear in the <menu> element, or in the <field> element that contains <option> elements.

Note that the <enumerate> element may only be used within the prompts and the catch elements associated with <menu> elements and with <field> elements that contain <option> elements.

Parent:  <audio>, <catch>, <error>, <field>, <help>, <if>, <menu>,
<noinput>, <nomatch>, <prompt>

Children:  <audio>, <value>
Attributes: None.

<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 children listed below where the element is either parsed text or the tag.

Parents: <field>, <form>, <initial>, <menu>, <record>, <subdialog>, <transfer>, <vxml>

Children: <assign>, <audio>, <clear>, <disconnect>, <enumerate>, <exit>, <goto>, <if>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>count</td>
<td>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 &lt;menu&gt; maintains a counter for each event that occurs while it is being visited; these counters are reset each time the &lt;menu&gt; or form item’s &lt;form&gt; is re-entered.</td>
</tr>
</tbody>
</table>

<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 children listed below.

Parent: <form>

Children: <audio>, <catch>, <enumerate>, <error>, <filled>, <help>, <if>, <link>, <noinput>, <nomatch>, <prompt>, <return>, <script>, <submit>, <throw>, <value>, <var>
### Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cond</strong></td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited</td>
</tr>
<tr>
<td><strong>expr</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>modal</strong></td>
<td>If FALSE (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.</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>Field-item variable in the dialog scope that holds the result.</td>
</tr>
<tr>
<td><strong>slot</strong></td>
<td>Name of the grammar slot used to populate the variable. If it is absent, it defaults to the variable name. 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.</td>
</tr>
<tr>
<td><strong>type</strong></td>
<td>Type of field, for example digits, currency, phone, date, Boolean, number, time. If not present, <code>&lt;grammar&gt;</code> and/or <code>&lt;dtmf&gt;</code> elements can be specified instead.</td>
</tr>
</tbody>
</table>

**Note:** An error event may be thrown if a reserved ECMAScript word is used as the name. If the type attribute references an invalid builtin type, the error.unsupported.builtin event is thrown at runtime.

### `<filled>`

Specifies an action to perform when some combination of fields are filled by user input.

The value can be empty or can contain one or more of the children listed below.

**Parents:**  
- `<field>`  
- `<form>`  
- `<record>`  
- `<subdialog>`  
- `<transfer>`

**Children:**  
- `<assign>`  
- `<audio>`  
- `<clear>`  
- `<disconnect>`  
- `<exit>`  
- `<goto>`  
- `<if>`  
- `<log>`  
- `<prompt>`  
- `<reprompt>`  
- `<return>`  
- `<script>`  
- `<submit>`  
- `<throw>`  
- `<value>`  
- `<var>`
Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>Any or all (default). If any, action is executed when any field is filled by last user input, If all, action executes when all fields are filled.</td>
</tr>
<tr>
<td>namelist</td>
<td>The input items on which to trigger. It defaults to the names of the form’s field items.</td>
</tr>
</tbody>
</table>

Note: An error:badfetch event is thrown if mode or namelist is specified for a <filled> within an input item (ex. in a <field>).

<font>

Specifies the font used in a text overlay. For an example of a text overlay, see “Image Overlay and Text Overlay” (page 308). For more on text overlay, see “Image Overlay and Text Overlay Functionality” (page 227).

This element is not used for image overlays.

Parents:  
<content>

Children: none

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the font. Specify the URL to the font file using this format: font:///usr/share/fonts/.... Or for cached fonts, specify the name defined in trcr.cfg. The default value is font:///usr/share/fonts/liberation/LiberationSans-Regular.ttf.</td>
</tr>
<tr>
<td>size</td>
<td>Font size. The default value is 14.</td>
</tr>
<tr>
<td>style</td>
<td>Used by cached fonts only, which are defined in trcr.cfg. For dynamic fonts, the style is defined in the font file. One or more attributes to apply to the text. Values are: normal, bold, italic, underlined, and outlined. The default value is normal.</td>
</tr>
</tbody>
</table>
<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 or more of the child tags listed below.

**Parent:** `<vxml>`
**Children:** `<block>, <catch>, <error>, <field>, <filled>, <grammar>, <help>, <initial>, <link>, <noinput>, <nomatch>, <object>, <property>, <record>, <subdialog>, <var>`

**Attributes:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Name of the form.</td>
</tr>
<tr>
<td>scope</td>
<td>The scope of the form’s grammar. Defaults to the dialog scope.</td>
</tr>
</tbody>
</table>

**Note:** An error.badfetch event is thrown if multiple forms have the same id (on the same page).

</form>

<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>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>`

**Attributes:** One of the following values must be specified:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>ECMAScript expression that yields the target URI.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>expritem</td>
<td>ECMAScript expression that yields the name of the target form item.</td>
</tr>
<tr>
<td>next</td>
<td>URL to which to transition.</td>
</tr>
<tr>
<td>nextitem</td>
<td>Name of the next form item in the current form.</td>
</tr>
<tr>
<td>fetchaudio</td>
<td>The URI of audio to play while waiting for the document to be fetched. Optional.</td>
</tr>
<tr>
<td>fetchtimeout</td>
<td>The length of time to wait for the next document to be fetched before throwing an error.badfetch event. Optional.</td>
</tr>
</tbody>
</table>

**<grammar>**

Defines a grammar for speech or DTMF recognition. The IP Media Server supports the XML Form of the W3C Speech Recognition Grammar Specification (SRGS).

**Parents:**  <choice>, <field>, <form>, <link>, <record>, <transfer>
Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xml:lang</code></td>
<td>The language identifier of the grammar. The list of supported values for this attribute depends on the ASR engine being used. Optional.</td>
</tr>
</tbody>
</table>
| `src` | The URI specifying the location of the grammar file. The URI can be one of the following formats:  
  - External grammar file: URL of the grammar file  
  - Built-in grammars: `builtin:grammar/type` (refer to `<field>` element for built-in types) |
| `scope` | The scope of the grammar. This attribute can be defined only if the `<grammar>` element is the child of a `<form>` or `<menu>` element. Optional. (Defaults to dialog.)  
  - `dialog` - This grammar is only active within the form.  
  - `document` - This grammar is active throughout the document. If the current page is an application root document, then the grammar is active throughout the application. |
| `type` | The MIME type of the grammar format. The IP Media Server supports the "application/srgs+xml " MIME type. |
| `mode` | The mode of the grammar. Optional. (Defaults to voice.)  
  - `voice` - voice input  
  - `dtmf` - DTMF input (replaces the `<dtmf>` element) |
| `root` | The root rule of the grammar. The default value depends on the ASR engine being used. Optional. |
| `version` | The version of the grammar. Optional. (Defaults to 1.0.) |
| `weight` | Specifies the weight of the grammar. |
| `tag-format` | The format that will be used to define tags, for slot filling. Support for this attribute depends on the ASR engine being used. |
| `xml:base` | Declares the base URI from which relative URIs in the grammar are resolved. This base declaration has precedence over the `<vxml>` base URI declaration. Optional. |
| `fetchhint` | Defines when the grammar file should be fetched. Optional.  
  - `prefetch` - grammar file may be downloaded when the page is loaded  
  - `safe` - only load the grammar file when needed |
| `fetchtimeout` | The length of time to wait for the grammar file to be fetched before throwing an error.badfetch event. Optional. |
| `maxage` | Indicates that this document is willing to use a cached copy of the grammar file only while the age of the cached copy is less than or equal to the number of seconds specified by this attribute. Optional. |
Referencing an unsupported built-in grammar will result in an error.unsupportedbuiltin being thrown.

Speech grammars require an ASR resource to be configured and available or else an error.noresource event will be thrown.

DTMF grammars are handled local to the IP Media Server and do not require an ASR resource. Refer to DTMF Grammars Guidelines on page 180 for more information.

<help>

Receives a help event, and is shorthand for <catch event="help">. The value can be empty or can contain one or more of the children listed below.

Parents:  
<field>, <form>, <initial>, <menu>, <record>, <subdialog>, <transfer>, <vxml>

Children:  
<assign>, <audio>, <clear>, <disconnect>, <exit>, <goto>, <if>,
<log>, <prompt>, <reprompt>, <return>, <script>, <submit>,
<throw>, <value>, <var>

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>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 &lt;menu&gt; maintains a counter for each event that occurs while it is being visited; these counters are reset each time the &lt;menu&gt; or form item’s &lt;form&gt; is re-entered.</td>
</tr>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
</tbody>
</table>

<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 children listed below.

Parents:  
<block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>
Children:  <assign>, <audio>, <clear>, <disconnect>, <else>, <elseif>, <exit>, <goto>, <if>, <log>, <prompt>, <reprompt>,
<return>, <script>, <submit>, <throw>, <value>, <var>

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
</tbody>
</table>

<initial>
Declar...
**Attributes:** Must specify one of next, expr, or event.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>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).</td>
</tr>
<tr>
<td>expr</td>
<td>Like next, except that the URI is dynamically determined by evaluating the given ECMAScript expression.</td>
</tr>
<tr>
<td>event</td>
<td>The event to generate when the user matches one of the link grammars.</td>
</tr>
<tr>
<td>eventexpr</td>
<td>An ECMAScript expression to be evaluated and used as the name of the event to be thrown when the user input matches one of the link grammars. Exactly one of next, expr, event, or eventexpr must be specified.</td>
</tr>
<tr>
<td>fetchtimeout</td>
<td>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.</td>
</tr>
<tr>
<td>dtmf</td>
<td>The DTMF sequence for this link. It is equivalent to a simple DTMF grammar. This attribute can be used at the same time as other grammars: the link is activated when user input matches either a link grammar or this DTMF sequence. Optional.</td>
</tr>
<tr>
<td>fetchaudio</td>
<td>The URI of audio to play while waiting for the document to be fetched. Optional.</td>
</tr>
<tr>
<td>maxage</td>
<td>Indicates that this document is willing to use a cached copy of the next document only while the age of the cached copy is less than or equal to the number of seconds specified by this attribute. Optional.</td>
</tr>
<tr>
<td>maxstale</td>
<td>Indicates that this document is willing to use a cached copy of the next document that has exceeded its expiration time by as much as the number of seconds specified by this attribute. Optional.</td>
</tr>
<tr>
<td>message</td>
<td>A message string providing additional context about the event being thrown. The message will be available as a variable within the scope of the &lt;catch&gt; element. Optional. (Can only be used if event or eventexpr is specified; only one of message and messageexpr can be specified.)</td>
</tr>
<tr>
<td>messageexpr</td>
<td>An ECMAScript expression to be evaluated and used as a message string, as documented under message, above. Optional. (Can only be used if event or eventexpr is specified; only one of message and messageexpr can be specified.)</td>
</tr>
</tbody>
</table>

**Note:** The event specified by event/eventexpr is thrown based on the scope of the active dialog (ie. <field>, <menu>, etc., where the user was when the link grammar was matched). All attribute values that are ECMAScript expressions
are evaluated based on variables from the scope of the active dialog (ie. &lt;field&gt;, &lt;menu&gt;, etc., where the user was when the link grammar was matched). An error.badfetch event is thrown if more or less than one of next, expr, event, and eventexpr are specified.

&lt;log&gt;

The &lt;log&gt; element allows the application to generate messages for debugging.

**Parents:** &lt;log&gt;, &lt;catch&gt;, &lt;error&gt;, &lt;filled&gt;, &lt;help&gt;, &lt;if&gt;, &lt;noinput&gt;, &lt;nomatch&gt;

**Children:** &lt;value&gt;

**Attributes:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>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 &lt;log&gt; 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. (Default: none.) Following is an example &lt;log&gt; tag with the trap extension: &lt;log dest=&quot;snmp&quot;&gt;Unable to contact VoiceXML application server.&lt;/log&gt;</td>
</tr>
<tr>
<td>expr</td>
<td>An ECMAScript expression of which the evaluated value will be appended to the &lt;log&gt; element content and label attribute value. Optional.</td>
</tr>
<tr>
<td>label</td>
<td>A string label that will be appended to the &lt;log&gt; element content before the value of the expr attribute is appended. Optional.</td>
</tr>
</tbody>
</table>

&lt;menu&gt;

Dialog for making a selection among choices.

The value can be empty or can contain one or more of the children listed below.

**Parents:** &lt;xmll&gt;

**Children:** &lt;audio&gt;, &lt;catch&gt;, &lt;choice&gt;, &lt;enumerate&gt;, &lt;error&gt;, &lt;help&gt;, &lt;noinput&gt;, &lt;nomatch&gt;, &lt;prompt&gt;, &lt;property&gt;, &lt;value&gt;

**Attributes:**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
</table>
| id         | Identifier of the menu. It allows the menu to be the target of a &lt;goto&gt; or a &lt;submit&gt;.
| scope      | Menu’s grammar scope. Either dialog (default) or document. |
<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:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the meta-data property.</td>
</tr>
<tr>
<td>http-equiv</td>
<td>Name of an HTTP response header. Either name or http-equiv must be specified, but not both.</td>
</tr>
<tr>
<td>content</td>
<td>Value of the meta-data property. Required.</td>
</tr>
</tbody>
</table>

<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 children listed below.

**Parents:** <field>, <form>, <initial>, <menu>, <record>, <subdialog>, <transfer>, <vxml>

**Children:** <assign>, <audio>, <clear>, <disconnect>, <enumerate>, <exit>, <goto>, <if>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>
Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>count</td>
<td>A number that allows you to issue different prompts if the user is doing something repeatedly. If omitted, it defaults to 1.</td>
</tr>
</tbody>
</table>

`<nomatch>`

Used to catch a nomatch event. Shorthand for `<catch event="nomatch">`. See “`<catch>`” (page 184).

The value can be empty or can contain one or more of the children listed below.

Parents: `<field>, <form>, <initial>, <menu>, <object>, <record>, <subdialog>, <transfer>, <vxml>`

Children: `<assign>, <audio>, <clear>, <disconnect>, <enumerate>, <exit>, <goto>, <if>, <log>, <prompt>, <repprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>`

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>count</td>
<td>A number that allows you to issue different prompts if the user is doing something repeatedly. If omitted, it defaults to 1.</td>
</tr>
</tbody>
</table>

`<object>`

No IP Media Server platform-specific objects are currently defined. Therefore, any use of `<object>` will result in the following error thrown:

```
error.unsupported.objectname
```

Objectname is a fixed string; it is not substituted with the name of the object.

`<option>`

Specifies an option in a `<field>`. It is a convenient way to specify a list of choices in a field without specifying a grammar. The grammar is generated automatically from the text contained in each `<option>` and the optionally specified dtmf sequence in each `<option>`.

Parent: `<field>`
Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtmf</td>
<td>A DTMF sequence to select this option. Optional.</td>
</tr>
</tbody>
</table>
| accept    | Specifies how to generate the grammar for this <option> element. Optional. (Defaults to exact.)
  | • exact - The text contained in the <option> element defines the exact phrase to be recognized. |
  | • approximate - The text contained in the <option> element defines an approximate recognition phrase. A subset of the words in the phrase can be matched. For example, "hello world" can be matched with "hello world", "hello", or "world". |
| value     | The string to assign to the field variable when this option is selected, either by speech or DTMF. Optional. (Defaults to the option text with leading and trailing white space removed, if specified. Otherwise, defaults to the value of the dtmf attribute. |

<overlay>

Specifies an overlay for a video stream. For examples of overlays, see "Image Overlay and Text Overlay" (page 308). For more on overlays, see "Image Overlay and Text Overlay Functionality" (page 227).

Parent: <prompt>

Children: <content>

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position X&lt;int&gt;Y&lt;int&gt;</td>
<td>The X coordinate and Y coordinate of the upper left-hand corner of the overlay area, given either in pixels or as a percentage of the background video frame.</td>
</tr>
<tr>
<td></td>
<td>To specify a percentage, include % character in the value.</td>
</tr>
<tr>
<td></td>
<td>The X and Y values can independently use pixels or a percentage value.</td>
</tr>
<tr>
<td></td>
<td>Example: &lt;overlay position=x20y50%&gt;.</td>
</tr>
<tr>
<td>size X&lt;int&gt;Y&lt;int&gt;</td>
<td>The height (X coordinate) and width (Y coordinate) of the overlay area, given either in pixels or as a percentage of the background video frame.</td>
</tr>
<tr>
<td></td>
<td>To specify a percentage, include % character in the value.</td>
</tr>
<tr>
<td></td>
<td>The X and Y values can independently use pixels or a percentage value.</td>
</tr>
<tr>
<td></td>
<td>Example: &lt;overlay size=x45%y20&gt;.</td>
</tr>
<tr>
<td>delay</td>
<td>Delays the start of the overlay until a specified number of seconds has passed from the start of the video stream.</td>
</tr>
<tr>
<td></td>
<td>Default value is 0.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>layer</td>
<td>The order in which overlays are displayed when multiple overlays are specified. An overlay with a higher number (such as 5) is displayed in the foreground relative to an overlay with a lower number (such as 4). When using multiple overlays, be careful that the overlays do not overlap and render the text unreadable. By default, the same layer value is used for all overlays. Default value is 0.</td>
</tr>
<tr>
<td>border-width</td>
<td>Border width of the overlay area given in pixels. Default value is 0.</td>
</tr>
<tr>
<td>border-color</td>
<td>Border color of the overlay area. Values are: red, blue, green, yellow, magenta, cyan, white, black, transparent or an unsigned integer. If a color enumeration is used, the color will be opaque. If an integer is specified, use the format 0xRRGGBBTT, where RR is a 2-digit hexadecimal value for the red component, GG is for the green component, BB is for the blue component, and TT is the value for the transparency component. Transparent can be used as a color or specified in the TT of the bitmask. Default value is black.</td>
</tr>
<tr>
<td>timeout</td>
<td>Stops displaying the overlay after a specified number of seconds has passed from the start of the video stream. A value of 0 means that no timeout will occur. Default value is 0.</td>
</tr>
<tr>
<td>bgcolor</td>
<td>Background color of the overlay area. Values are: red, blue, green, yellow, magenta, cyan, white, black, transparent or an unsigned integer. If a color enumeration is used, the color will be opaque. If an integer is specified, use the format 0xRRGGBBTT, where RR is a 2-digit hexadecimal value for the red component, GG is for the green component, BB is for the blue component, and TT is the value for the transparency component. Transparent can be used as a color or specified in the TT of the bitmask. Default value is transparent. For foreground color of the text, see &lt;font&gt;.</td>
</tr>
</tbody>
</table>

**<param>**

Specifies the values that are passed to subdialogs.
The value of this tag must be empty.

**Parent:**  
<object>, <subdialog>
## Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name to be associated with this parameter when the subdialog is invoked. Required.</td>
</tr>
<tr>
<td>expr</td>
<td>Expression that computes the value associated with name.</td>
</tr>
<tr>
<td>value</td>
<td>Associates a literal string value with name.</td>
</tr>
<tr>
<td>valuetype</td>
<td>Whether the value associated with name is data or a URI (ref). Default is data.</td>
</tr>
<tr>
<td>type</td>
<td>MIME type of the result provided by a URI if the value type is ref. Only relevant for uses of <code>&lt;param&gt;</code> in <code>&lt;object&gt;</code>.</td>
</tr>
</tbody>
</table>

### `<prompt>`

Controls the output of pre-recorded audio, video, multimedia or TTS to the user.

The value can be empty or can contain one or more of the children listed below, where the element is either parsed text or the tag.

**Parents:** `<block>`, `<catch>`, `<error>`, `<field>`, `<filled>`, `<help>`, `<if>`, `<initial>`, `<menu>`, `<noinput>`, `<nomatch>`, `<object>`, `<return>`, `<subdialog>`, `<transfer>`

**Children:** `<audio>`, `<enumerate>`, `<overlay>`, `<ssml>` `<value>`

### Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bargein</td>
<td>Whether a prompt may be interrupted. True - Allow input to interrupt (Default) False - Do not allow input to interrupt</td>
</tr>
</tbody>
</table>
| bargeintype | The bargein type. Optional. (Defaults to speech.)  
  - speech - Any user utterance can barge in on the prompt.  
  - hotword - Only user input that matches a grammar can barge in on the prompt. (Not supported in the current release.)  
  Overrides the value of the bargeintype property. |
| 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. |
| 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>, <object>, <record>, <subdialog>, <transfer>, <vxml>

**Attributes:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Property name. Required.</td>
</tr>
<tr>
<td>value</td>
<td>Property value. Required.</td>
</tr>
</tbody>
</table>

### <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 children listed below.

**Parent:**  
<form>

**Children:**  
<audio>, <catch>, <error>, <filled>, <grammar>, <help>, <noinput>, <nomatch>, <prompt>, <property>, <value>

**Attributes:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beep</td>
<td>TRUE or FALSE (the default). If TRUE, the application plays a tone just prior to recording.</td>
</tr>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
</tbody>
</table>
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtmfterm</td>
<td>TRUE or FALSE. If TRUE (default), a DTMF keypress terminates the recording. If FALSE, a DTMF keypress does not terminate recording.</td>
</tr>
<tr>
<td>expr</td>
<td>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.</td>
</tr>
<tr>
<td>finalsilence</td>
<td>Interval of silence that indicates the end of speech. (Default: 3 seconds.)</td>
</tr>
<tr>
<td>meantime</td>
<td>If the duration of the recording is less than this attribute, then the recording is assumed to be empty and a nonbook is thrown. Minimum value is 250ms (if defined as a shorter duration, it will be set to 250ms); maximum value is the value of the maxtime attribute (if defined as a longer duration, it will be set to 250ms). Optional. (Defaults to 250ms.)</td>
</tr>
<tr>
<td>maxtime</td>
<td>Maximum duration to record. (Default: 10 seconds.)</td>
</tr>
<tr>
<td>name</td>
<td>The name of a form-item variable used to track whether the &lt;initial&gt; is eligible to execute; defaults to an inaccessible internal variable.</td>
</tr>
<tr>
<td>type</td>
<td>MIME type of the recording.</td>
</tr>
</tbody>
</table>

**Note:** Voice and DTMF grammars are not active during the record. DTMF digits can only be used to terminate the recording when the record elements 'dtmfterm' attribute is set to true.

### <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 subcollege 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Returns, then sends this event.</td>
</tr>
<tr>
<td>namelist</td>
<td>Variable names to be returned to the calling dialog.</td>
</tr>
<tr>
<td>eventexpr</td>
<td>An ECMAScript expression to be evaluated and used as the name of an event to be returned to the calling dialog and thrown. Exactly one of event, eventexpr, or namelist may be specified.</td>
</tr>
<tr>
<td>message</td>
<td>A message string providing additional context about the event being thrown. The message will be available as a variable within the scope of the &lt;catch&gt; element. Optional. (Can only be used if event or eventexpr is specified; only one of message or messageexpr can be specified.)</td>
</tr>
<tr>
<td>messageexpr</td>
<td>An ECMAScript expression to be evaluated and used as the message string, as documented under message, above. Optional. (Can only be used if event or eventexpr is specified; only one of message or messageexpr can be specified.)</td>
</tr>
</tbody>
</table>

<say-as>

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 <say-as> 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.

Note: The <say-as> element is supported with IP Media Server resources and not a TTS server. It provides an alternative to TTS for minimal phrases.

Parents:  <audio>, <choice>, <emphasis>, <enumerate>, <paragraph>, <p>, <prompt>, <prosody>, <sentence>, <s>, <speak>, <voice>

Children:  <value>

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interpret-as</td>
<td>Describes the way to render the value of:</td>
</tr>
<tr>
<td></td>
<td>• phone (NANP numbers)</td>
</tr>
<tr>
<td></td>
<td>• digits (Each spoken with no pauses.)</td>
</tr>
<tr>
<td></td>
<td>• number (Spoken as a positive cardinal number. Negative numbers are not supported and will produce odd output.)</td>
</tr>
</tbody>
</table>
<script>
Allows the specification of a block of client-side ECMAScript code.

**Note:** Although the 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>, <form>, <help>, <if>, <noinput>, <nomatch>, <vxml>

**Attributes:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>URL for the resource.</td>
</tr>
<tr>
<td>chariest</td>
<td>The character encoding if an external script is used. Optional.</td>
</tr>
<tr>
<td>fetchhint</td>
<td>Defines when the script should be fetched. Optional.</td>
</tr>
</tbody>
</table>
| | • prefect - script may be downloaded when the page is loaded  
| | • safe - only load the script when needed |
| fetchtimeout | The length of time to wait for the script to be fetched before throwing an error.badfetch event. Optional. |
| maxage | Indicates that this document is willing to use a cached copy of the script only while the age of the cached copy is less than or equal to the number of seconds specified by this attribute. Optional. |
| maxstale | Indicates that this document is willing to use a cached copy of the script that has exceeded its expiration time by as much as the number of seconds specified by this attribute. Optional. |

<scroll>
Specifies the characteristics of scrolling text in a text overlay. This is an optional element for a text overlay. If the <scroll> element is not used, the text overlay will be static. For an example of a text overlay, see “Image Overlay and Text Overlay” (page 308). For more on text overlay, see “Image Overlay and Text Overlay Functionality” (page 227)

This element is not used for image overlays.

**Parents:** <content>

**Children:** none
Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>The mode of the scrolling text. Values are: content (scroll one time), continuous (scroll continuously), and none. Required attribute. Default is none.</td>
</tr>
<tr>
<td>speed</td>
<td>Speed of scrolling measured in pixels per second. Default is 25.</td>
</tr>
<tr>
<td>direction</td>
<td>Direction of scrolling. Values are: left, right, up, down. Default is left.</td>
</tr>
<tr>
<td>term-delay</td>
<td>Used with content mode. Termination delay. The amount of time in seconds to keep the overlay in the video after scrolling completes. This value must be greater than 0. Default is 5 seconds.</td>
</tr>
</tbody>
</table>

<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 children listed below.

Parent: <form>

Children: <audio>, <catch>, <error>, <filled>, <help>, <noinput>, <nomatch>, <param>, <prompt>, <property>, <value>

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped. The form is not visited.</td>
</tr>
<tr>
<td>expr</td>
<td>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.</td>
</tr>
<tr>
<td>method</td>
<td>Request method (get or post).</td>
</tr>
<tr>
<td>modal</td>
<td>Controls which grammars are active during the subcollege.</td>
</tr>
<tr>
<td>name</td>
<td>The result returned from the subdialog, an ECMAScript object whose properties are the ones defined in the namelist attribute of the &lt;return&gt; element.</td>
</tr>
</tbody>
</table>
<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Like next, except that the URI is dynamically determined by evaluating the given ECMAScript expression. One of next, expr, or fetchtimeout is required.</td>
</tr>
<tr>
<td>method</td>
<td>The request method (GET or POST). The default is GET.</td>
</tr>
<tr>
<td>namelist</td>
<td>The list of variables to submit.</td>
</tr>
<tr>
<td>next</td>
<td>The URL to which the query is submitted.</td>
</tr>
<tr>
<td>fetchtimeout</td>
<td>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.</td>
</tr>
<tr>
<td>enctype</td>
<td>The MIME encoding of the submitted data. The following types are supported:</td>
</tr>
<tr>
<td></td>
<td>• application/x-www-form-urlencoded</td>
</tr>
<tr>
<td></td>
<td>• multipart/form-data</td>
</tr>
<tr>
<td></td>
<td>Optional. (Defaults to application/x-www-form-urlencoded.)</td>
</tr>
<tr>
<td>fetchaudio</td>
<td>The URI of audio to play while waiting for the document to be fetched. Optional</td>
</tr>
<tr>
<td>fetchhint</td>
<td>Defines when the subdialog should be fetched. Optional.</td>
</tr>
<tr>
<td></td>
<td>• safe - only load the subdialog when needed</td>
</tr>
<tr>
<td>maxage</td>
<td>Indicates that this document is willing to use a cached copy of the subdialog only while the age of the cached copy is less than or equal to the number of seconds specified by this attribute. Optional.</td>
</tr>
<tr>
<td>maxstale</td>
<td>Indicates that this document is willing to use a cached copy of the subdialog that has exceeded its expiration time by as much as the number of seconds specified by this attribute. Optional.</td>
</tr>
</tbody>
</table>

### Note:
An error.semantic event is thrown if an undeclared variable name is used in the namelist. An error.badfetch event is thrown if more or less than one of next and expr are specified.

### <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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Returns, then sends this event.</td>
</tr>
<tr>
<td>eventexpr</td>
<td>An ECMAScript expression to be evaluated and used as the name of the event to be thrown. Exactly one of event or eventexpr must be specified.</td>
</tr>
<tr>
<td>message</td>
<td>A message string providing additional context about the event being thrown. For the pre-defined events thrown by the platform, the value of the message is platform-dependent. The message will be available as a variable within the scope of the &lt;catch&gt; element. Optional. (Only one of message or messageexpr can be specified.)</td>
</tr>
<tr>
<td>messageexpr</td>
<td>An ECMAScript expression to be evaluated and used as a message string, as documented under message, above. Optional. (Only one of message or messageexpr can be specified.)</td>
</tr>
</tbody>
</table>

Note: An error.badfetch event is thrown if more or less than one of event and eventexpr are specified, or if more than one of message and messageexpr are specified.

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The outcome of the transfer attempt. For possible return values, see the following table. <strong>Note:</strong> If the transfer request is unacceptable and the transfer is not initiated at all, the transfer variable will not be set and the platform will throw an error, instead.</td>
</tr>
<tr>
<td>expr</td>
<td>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.</td>
</tr>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>dest</td>
<td>The URL of the destination (phone, IP telephone address). (Default: none)</td>
</tr>
<tr>
<td>destexpr</td>
<td>An ECMAScript expression yielding the URI of the destination. The expression must produce a valid SIP URI, for example sip:<a href="mailto:19783678400@gateway.carrier.net">19783678400@gateway.carrier.net</a>. (Default: none.)</td>
</tr>
<tr>
<td>bridge</td>
<td>Determines what to do once the call is connected. A value of ‘true’ indicates the transfer is bridged. A value of ‘false’ indicates a blind transfer. A blind transfer is not currently supported by the IP Media Server. A value of false will result in the error.unsupported.transfer.blind event being thrown. A bridged transfer is currently implemented by hairpinning 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.</td>
</tr>
<tr>
<td>src</td>
<td>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)</td>
</tr>
<tr>
<td>srcexpr</td>
<td>An ECMAScript expression yielding src. (Dialogic-specific attribute)</td>
</tr>
<tr>
<td>stopdigits</td>
<td>Defines the list of single DTMF digits that ends the transfer.</td>
</tr>
<tr>
<td>longdigit</td>
<td>Enables detection of long DTMF digits or multiple, short instance of the same digit. Indicates that the transfer can be terminated only with a long DTMF digit or multiple, short instances of the same digit. YES turns DTMF detection on for long digits. NO (default) turns DTMF detection off for long digits.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>type</strong></td>
<td>Specifies the transfer behavior (from the VoiceXML application's perspective). Optional.</td>
</tr>
<tr>
<td><strong>video</strong></td>
<td>Whether video is available on the outbound call. The value can be one of suppress or allow. See “RTP Codec Selection Using the &lt;transfer&gt; Element” (page 226).</td>
</tr>
<tr>
<td><strong>connecttimeout</strong></td>
<td>The time to wait while trying to connect the call before returning, and setting the transfer variable to 'noanswer'. Minimum value allowed is 5s. This attribute only applies for bridge or consultation transfers. Optional. (Defaults to 30s.)</td>
</tr>
<tr>
<td><strong>maxtime</strong></td>
<td>The maximum time that the call is allowed to last. Minimum value is 30 seconds; maximum is one week. A value of 0 represents no limit. This attribute only applies for bridge transfers. Optional. (Defaults to 0.)</td>
</tr>
<tr>
<td><strong>transferaudio</strong></td>
<td>The URI of an audio file to be played while connecting the call. Transfer audio cannot be played if connectwhen=&quot;immediate&quot;; in this case, network ringing will be heard instead. Optional.</td>
</tr>
<tr>
<td><strong>aai</strong></td>
<td>A string containing Application-to-Application Information (AAI) data to be sent to an application on the far-end.</td>
</tr>
<tr>
<td><strong>aaiexpr</strong></td>
<td>An ECMAScript expression evaluating to the Application-to-Application Information (AAI) data string, to be sent to an application on the far-end, as documented under aai, above.</td>
</tr>
<tr>
<td><strong>requiri</strong></td>
<td>The SIP URI to be used as the Request-URI in an outbound INVITE. The value must be compliant with the grammar for a SIP URI. 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. This attribute and the requiriexpr attribute enable you to route outbound calls to a proxy. The proxy will modify the request as needed based on the URI parameters placed in the INVITE and relevant network and subscriber data.</td>
</tr>
<tr>
<td><strong>requiri expr</strong></td>
<td>An ECMAScript expression that evaluates to a SIP URI.</td>
</tr>
</tbody>
</table>

**Note:** The src, srcexpr, stoptdigits, longdigit, and video attributes are Dialogic-specific attributes.

You can use one (but not both) of the following attribute pairs:
- dest and destexpr
- requir and requiriexpr
- src and srcexpr
- aai and aaiexpr
Return values of Name variable:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>The endpoint refused the call.</td>
</tr>
<tr>
<td>noanswer</td>
<td>There was no answer within the specified time.</td>
</tr>
<tr>
<td>network busy</td>
<td>Some intermediate network refused the call.</td>
</tr>
<tr>
<td>near_end_disconnect</td>
<td>The call completed and was terminated by the caller.</td>
</tr>
<tr>
<td>far_end_disconnect</td>
<td>The call completed and was terminated by the callee.</td>
</tr>
<tr>
<td>network_disconnect</td>
<td>The call completed and was terminated by the network.</td>
</tr>
<tr>
<td>maxtime_disconnect</td>
<td>The call duration exceeded the value of the maxtime attribute and was terminated by the platform.</td>
</tr>
</tbody>
</table>

**Note:** An error event may be thrown if a reserved ECMAScript word is used as the name. An error.badfetch event is thrown if more or less than one of dest and destexpr are specified, or if more than one of aai and aaiexpr are specified.

**<value>**

Inserts the value of an ECMAScript expression into a prompt.

The value of this tag must be empty.

**Parents:** `<audio>`, `<block>`, `<catch>`, `<choice>`, `<enumerate>`, `<error>`, `<field>`, `<filled>`, `<help>`, `<if>`, `<initial>`, `<log>`, `<menu>`, `<noinput>`, `<nomatch>`, `<object>`, `<prompt>`, `<record>`, `<subdialog>`, `<transfer>`

**Attributes:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Expression to render. Required.</td>
</tr>
</tbody>
</table>

**<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>`, `<vxml>`
Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the variable. Required.</td>
</tr>
<tr>
<td>expr</td>
<td>An initial value.</td>
</tr>
</tbody>
</table>

<vxml>

The top-level element in a VoiceXML document.

The value of this tag must contain one of more of the children listed below.

Parent: None

Children: <catch>, <error> <form>, <help>, <link>, <menu>, <meta>, <noinput>, <nomatch>, <property>, <script>, <var>

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Version of VoiceXML. Set to 2.0. Required.</td>
</tr>
<tr>
<td>base</td>
<td>Base URL. The root document URL, if any.</td>
</tr>
<tr>
<td>xmlns</td>
<td>The VoiceXML specification requires the use of this attribute (set to <a href="http://www.w3.org/2001/vxml">http://www.w3.org/2001/vxml</a>) to designate the namespace for VoiceXML.</td>
</tr>
<tr>
<td>lang</td>
<td>The language identifier for this document. Used as the language for implicit grammars (i.e., universal grammars, and &lt;choice&gt;/&lt;option&gt; grammars), multi-language Nuance grammars, and &lt;prompt&gt;s. Optional.</td>
</tr>
<tr>
<td>application</td>
<td>The URI of this document's application root document. Optional.</td>
</tr>
</tbody>
</table>

Note: Omitting version or xmlns when they are required will result in an error.badfetch being thrown.

If the root document specified by application cannot be found, an error.badfetch is thrown. If the root document refers to another root document, an error.semantic event is thrown.
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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>The default critical digit timer is set to the same value as the interdigit timer (thus it has no effect).</td>
</tr>
<tr>
<td>-1</td>
<td>Immediate. Use “shortest match first”.</td>
</tr>
<tr>
<td>0</td>
<td>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.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Uses “longest match first”, and waits that amount of time when a match is found.</td>
</tr>
</tbody>
</table>
MIME Types Supported

The MIME type and optional codec parameters define the file format and encoding (audio or video) for the file. The MIME type supports both headerless (raw) and .wav file formats. Table 49 lists the supported MIME types, file formats, and audio encodings.

Table 49. Supported MIME Types, Formats, and Audio Encodings

<table>
<thead>
<tr>
<th>MIME Type</th>
<th>File Format</th>
<th>Audio Encoding</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio/basic</td>
<td>raw</td>
<td>G.711 µ-law</td>
<td></td>
</tr>
<tr>
<td>audio/x-alaw-basic</td>
<td>raw</td>
<td>G.711 A law</td>
<td></td>
</tr>
<tr>
<td>audio/vnd.wave</td>
<td>WAVE</td>
<td></td>
<td>IANA standard</td>
</tr>
<tr>
<td>audio/wav</td>
<td>WAVE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>audio/x-wav</td>
<td>WAVE</td>
<td></td>
<td>Proprietary</td>
</tr>
<tr>
<td>audio/3gpp</td>
<td>3GPP</td>
<td>AMR-NB</td>
<td></td>
</tr>
<tr>
<td>audio/3gpp2</td>
<td>3GPP2</td>
<td>AMR-NB</td>
<td></td>
</tr>
<tr>
<td>video/3gpp</td>
<td>3GPP</td>
<td>AMR-NB</td>
<td>Valid for audio-only file</td>
</tr>
<tr>
<td>video/3gpp2</td>
<td>3GPP2</td>
<td>AMR-NB</td>
<td>Valid for audio-only file</td>
</tr>
</tbody>
</table>

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.

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 (see http://www.iana.org/assignments/wave-avi-codec-registry), but other commonly used forms are also supported.

Table 50 provides a summary of the supported codec parameters and values and their corresponding audio encodings.

Table 50. Audio Codec Parameter Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Audio Encoding</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>G.711 µ-law</td>
<td>IANA standard</td>
</tr>
<tr>
<td>6</td>
<td>G.711 A law</td>
<td>IANA standard</td>
</tr>
<tr>
<td>31</td>
<td>Microsoft® GSM</td>
<td>IANA standard</td>
</tr>
<tr>
<td>ulaw</td>
<td>G.711 µ-law</td>
<td></td>
</tr>
<tr>
<td>alaw</td>
<td>G.711 A law</td>
<td></td>
</tr>
</tbody>
</table>
Table 51 lists the supported MIME types, file formats, and video encodings.

Table 51. Supported MIME Types, Formats, and Video Encodings

<table>
<thead>
<tr>
<th>MIME Type</th>
<th>File Format</th>
<th>Encoding</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>video/x-wav</td>
<td>WAVE</td>
<td>Proprietary</td>
<td></td>
</tr>
<tr>
<td>video/3gpp</td>
<td>3GPP</td>
<td>Audio: AMR-NB.</td>
<td>Supported for play and record.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Video: H.263+,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>H.263++, or H.264.</td>
<td></td>
</tr>
<tr>
<td>video/3gpp2</td>
<td>3GPP2</td>
<td>Audio: AMR-NB.</td>
<td>Supported for play only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Video: H.263+,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>H.263++, or H.264.</td>
<td></td>
</tr>
</tbody>
</table>

The video/x-wav MIME type produces a recording using the same codec as the one negotiated during call setup.

The following code snippet is an example of how to record to a video file using the video/3gpp MIME type. You can replace this type with the preferred MIME type.

```xml
<record name="vmail_recording" beep="true" dtmfterm="true"
    type=video/3gpp" maxtime="20s"
    ...
</record>
```
ECMAScript Functionality

While the IP 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 IP Media Server and allows the IP Media Server to handle a large volume of simultaneous sessions.

Note: The ECMAScript Language Specification can be downloaded for free.

Table 52 summarizes the categories and features supported by the IP Media Server’s VoiceXML Interpreter.

### Table 52. ECMAScript Functionality

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature/Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Handling</td>
<td>Array join, length, reverse, sort</td>
</tr>
<tr>
<td>Assignments</td>
<td>Assign (=)</td>
</tr>
<tr>
<td></td>
<td>Compound Assign (OP=)</td>
</tr>
<tr>
<td>Booleans</td>
<td>Boolean</td>
</tr>
<tr>
<td>Comments</td>
<td>/<em>...</em>/ or //</td>
</tr>
<tr>
<td>Constants/Literals</td>
<td>NaN, null, true, false, Infinity, undefined</td>
</tr>
<tr>
<td>Control flow</td>
<td>Break, continue, for, for...in, if...else, return, while</td>
</tr>
</tbody>
</table>
Table 52. ECMAScript Functionality (continued)

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature/Keyword</th>
</tr>
</thead>
</table>
| Dates and Time       | Date
getDateTime, getDay, getFullYear, getHours,
getMilliseconds, getMinutes, getMonth,
getSeconds, getTime, getTimezoneOffset,
getYear,
getUTCDate, getUTCDay, getUTCFullYear,
getUTCHours, getUTCMilliseconds,
getUTCMilliseconds, getUTCMonth,
getUTCSeconds,
setDate, setFullYear, setHours,
setMilliseconds, setMinutes, setMonth,
setSeconds, setTime, setYear,
setUTCDate, setUTCFullYear, setUTCHours,
setUTCMilliseconds, setUTCMonth,
setUTCSeconds, toGMTString, toLocaleString, toUTCString,
parse, UTC           |
| 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             |
**Table 52. ECMAScript Functionality (continued)**

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature/Keyword</th>
</tr>
</thead>
</table>
| 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 (++) |
| Objects   | Array  
Boolean  
Date  
Function  
Global  
Math  
Number  
Object  
String |
| Strings   | String  
charAt, charCodeAt, fromCharCode  
indexOf, lastIndexOf  
split  
toLowerCase, toUpperCase  
length |
VoiceXML Extended Session Variables

The Dialogic® VoiceXML browser supports the following VoiceXML 2.0 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.
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%3d1;dogs=nice;user=phone
SIP/2.0
From: <sip:968037431@10.102.4.45:5060>;tag=2d04660a-13c4-40ed433f-9ec19fd-7061@10.102.4.45
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!yi8MK2SsMu8Uyake2qO0gMi8i!y.1-1d9515c
Via: SIP/2.0/UDP 10.102.4.45:5060;received=10.102.4.45;branch=z9hG4bK-40ed433f-9ec19fd-847
Max-Forwards: 69
User-Agent: RADVision ViaIP GW Vers. 2.0
Call-Info: <Media:Video>;purpose=info
Contact: <sip:968037431@10.102.4.45:5060>
Content-Length:0
```

The IP 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://10.102.4.9:9020/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=961234567&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://10.102.4.9:9020/ivr/sip_init.vxml?dnis=961234567&hasvideo=1;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
RTP Codec Selection Using the <transfer> Element

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. Application-level control to override this standard behavior is available. 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 53 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>
<thead>
<tr>
<th>Video Attribute Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow Value</td>
</tr>
<tr>
<td>Inbound Leg Negotiated SDP</td>
</tr>
<tr>
<td>Audio+Video</td>
</tr>
</tbody>
</table>

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.
Image Overlay and Text Overlay Functionality

This section provides information on text overlay and image overlay functionality:

- **Overlay Overview**
- **Characteristics of Image Overlay**
- **Characteristics of Text Overlay**
- **Text Overlay Restrictions**
- **Installing Font Files**
- **Dynamic Fonts**
- **Cached Fonts**
- **Overlay Demos**

**Overlay Overview**

The IP Media Server provides text and image overlay functionality that can be used to create interactive menus or to display corporate taglines and logos, among other things. Up to 16 simultaneous overlays per RTP stream are supported. These 16 overlays can be text overlays, image overlays, or a combination of text and image overlays.

Text overlay functionality is supported for play only; it is not supported for record.

Image overlay functionality is implemented in VoiceXML 2.0 using the `<overlay>` and `<content>` elements. Text overlay functionality is implemented in VoiceXML 2.0 using the `<overlay>`, `<content>`, `<font>`, and `<scroll>` elements. Default values are provided for various characteristics of an overlay, such as font, font size, and font color.

For reference information, see the `<overlay>`, `<content>`, `<font>`, and `<scroll>` element descriptions in this chapter. For examples of scripts, see “Image Overlay and Text Overlay” (page 308) in Chapter 7, “Sample Code and Call Flows”.

The creation of an overlay requires a definition of the overlay, such as its size and position, and a definition of the content to be displayed in the overlay.

The image overlay content definition specifies the source file to be rendered and the image type.

The text overlay content definition provides the text string to be rendered and a font definition. The IP Media Server offers cached font and dynamic font rendering. For more information, see “Cached Fonts” (page 229) and “Dynamic Fonts” (page 229).

**Characteristics of Image Overlay**

The IP Media Server supports the following file types for image overlay:

- **JPEG** (MIME type: image/jpeg)
- **PNG** (MIME type: image/png)
When you define the size of the overlay window (or area), the image is resized to fit this window and the aspect ratio of the original image is maintained. The image is typically centered in the window (content attributes halign is ignored).

The `<overlay>` and `<content>` elements are used for image overlay. The `<font>` and `<scroll>` elements are not used for image overlay.

**Characteristics of Text Overlay**

The following characteristics of a text overlay are configurable:

- Fonts, font sizes, and font colors for compatibility with varying video output formats. See `<font>`.
- The degree of transparency of text foreground (the text itself). See `<font fgcolor>`.
- The degree of transparency of text background (the space in between the letters). See `<overlay bgcolor>`.
- The size, position, and border of the text window in which the text input is displayed. See `<overlay>`.
- Scrolling text, its direction, speed, and mode of display. See `<scroll>`.
- Character-encoding, word wrap, and text alignment. See `<content>`.

**Text Overlay Restrictions**

The following restrictions are in effect for text overlay:

- For scrolling text, when scroll mode is continuous, there is no delay or space between the beginning of content and the end of content. To avoid this restriction, you can manually add spaces at the end of a text string.
- For scrolling text, the entire text overlay window scrolls. This means that the window is not fixed; instead, it scrolls with the text. To avoid this restriction, you can define two overlays: a smaller one to contain the text, and a larger one for the window. See “Scrolling Text Overlay” (page 308) for an example.
- For cached fonts, the font style must match the font file defined in trcr.cfg. For example, if the font file specifies a bold font, you must set the style to bold. If it specifies italics, you must set the style to italics.

**Installing Font Files**

Fonts must be installed on the IP Media Server system that will execute the VoiceXML script. A distributed environment, such as one using Font Server, is not supported.

The IP Media Server requires that the Red Hat Linux Liberation font package be installed during the operating system install. Other font packages such as Hindi and Chinese are provided by Red Hat and may be installed as needed using the `rpm-Uvh <font package name>` command. Font packages are placed in `/usr/share/fonts/` directories. For more information on installation, see the Software Quick Start Guide.

To use fonts not provided by Red Hat, you must manually copy the font file such as gz or rpm to the directory of choice. After the font file is on your system, you can specify it using the `<font>` element. See Dynamic Fonts and Cached Fonts for more information.
Dynamic Fonts

Dynamic font rendering allows rendering in any size using a font file that the application specifies at runtime. Dynamic font rendering is more flexible, but requires more processing at runtime and may add to play latency.

To use UTF-8 or GB18030 character encodings in a text overlay, you must use dynamic fonts. Cached fonts support the ASCII character set only.

To use dynamic fonts:

1. Ensure that the font files for the fonts to be used in the text overlay are already installed on your IP Media Server system. See “Installing Font Files” (page 228).

2. In the <content> element, specify a local file for src attribute. Specify values for other attributes as needed. For example:

   ```xml
   <content src="file:///opt/snowshore/htdocs/vxml/demos/media/pizza.gb" encoding="GB18030" halign="center" />
   ```

3. In the <font> element, specify the font file and its location for name attribute. The font file must be local to the IP Media Server system. Specify values for other attributes as needed. For example:

   ```xml
   <font name="font:///usr/share/fonts/chinese/TrueType/uming.ttf" size="30" fgcolor="black" />
   ```

Cached Fonts

Cached font rendering uses the fonts defined in the trcr.cfg configuration file to provide fast rendering. These cached fonts require less processing at runtime, but require fonts and sizes to be identified and stored in memory when the IP Media Server is started. Play latency is typically reduced but more memory is consumed.

The character set supported by cached fonts is limited to ASCII characters. For other character encodings such as UTF-8 or GB18030, you must use dynamic fonts; see “Dynamic Fonts” (page 229).

To use cached fonts:

1. Ensure that the font files for the fonts to be used in the text overlay are already installed on your IP Media Server system. See “Installing Font Files” (page 228).

2. Define the fonts in the [fonts] section of trcr.cfg, located in opt/snowshore/etc. See “trcr.cfg Description” (page 229). After making changes in trcr.cfg, you must restart the IP Media Server.

3. In the <content> element, specify a local file for src attribute or inline ASCII text. Specify values for other attributes as needed.

4. In the <font> element, specify a font name defined in trcr.cfg. Specify values for other attributes as needed. For example:

   ```xml
   <font name="SampleFont" size="24" fgcolor="blue" />
   ```

trcr.cfg Description

The trcr.cfg configuration file contains two sections: [global] and [fonts].
The [global] section of trcr.cfg specifies overall text and image rendering behavior and contains the following parameters:

<table>
<thead>
<tr>
<th>[global] section parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>debuglogmask</td>
<td>Set of logging bits that can be activated to obtain detailed tracing information of render operations. Default: 0x00000003 (trace any detected WARNING or ERROR condition). This variable should not require any modification for standard operations.</td>
</tr>
<tr>
<td>SharedMem</td>
<td>Sets the size of the shared memory area that is used to pass rendered overlay content between the rendering process and the media server. This value is set at download time based on the number of transcoding instances in the license. A default value of 99 KB is allocated for each transcoding device. The value of 99 KB is determined by allotting for one overlay covering a QCIF resolution image: 176 pixels wide * 144 pixels high * 4 bytes per pixel for rendering = 101376 bytes / 1024 = 99 kilobytes. If necessary, you can change this value using the KBsPerOverlayResource variable in /opt/snowshore/etc/snowshore.cfg. The amount of memory is allocated using this formula: the sum of transcoding devices * KBsPerOverlayResource. You will need to increase the KBsPerOverlayResource value if you receive the following error in mserv.log: trcr request failed (error=out of shared memory).</td>
</tr>
</tbody>
</table>

The [fonts] section of trcr.cfg contains font definitions used for cached font rendering. It specifies pre-defined fonts and contains the following parameters:

<table>
<thead>
<tr>
<th>[fonts] section parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>A short name to identify the font.</td>
</tr>
<tr>
<td>Font File Name</td>
<td>The name of the true type font file describing the font.</td>
</tr>
<tr>
<td>Pre-rendered From To</td>
<td>A range of font sizes to be pre-rendered.</td>
</tr>
<tr>
<td>Font sizes increment</td>
<td>The incremental value in the range of font sizes.</td>
</tr>
</tbody>
</table>
Sample trcr.cfg

The following example shows font definitions in the [fonts] section of trcr.cfg.

<table>
<thead>
<tr>
<th>Name</th>
<th>Font File Name</th>
<th>Pre-rendered</th>
<th>font sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>font MonoBold</td>
<td>&quot;/usr/share/fonts/liberation/LiberationMono-Bold.ttf&quot;</td>
<td>10 24</td>
<td>2</td>
</tr>
<tr>
<td>font MonoItal</td>
<td>&quot;/usr/share/fonts/liberation/LiberationMono-Italic.ttf&quot;</td>
<td>10 24</td>
<td>2</td>
</tr>
<tr>
<td>font Mono</td>
<td>&quot;/usr/share/fonts/liberation/LiberationMono-Regular.ttf&quot;</td>
<td>10 24</td>
<td>2</td>
</tr>
<tr>
<td>font SansBold</td>
<td>&quot;/usr/share/fonts/liberation/LiberationSans-Bold.ttf&quot;</td>
<td>10 24</td>
<td>2</td>
</tr>
<tr>
<td>font SansItal</td>
<td>&quot;/usr/share/fonts/liberation/LiberationSans-Italic.ttf&quot;</td>
<td>10 24</td>
<td>2</td>
</tr>
<tr>
<td>font Sans</td>
<td>&quot;/usr/share/fonts/liberation/LiberationSans-Regular.ttf&quot;</td>
<td>10 24</td>
<td>2</td>
</tr>
</tbody>
</table>

Overlay Demos

Overlay demos (sample applications) are provided in /opt/snowshore/htdocs/vxml/demos/tovl. For more information, see the demo readme file included in this directory.
7 - Sample Code and Call Flows

This chapter provides examples and call flow diagrams for a variety of scenarios.

This chapter includes the following sections:

- Announcements Sample Code and Call Flows
- Conferences Sample Code and Call Flows
- IVR with MSCML Sample Code and Call Flows
- VoiceXML Sample Code and Call Flows
Announcements Sample Code and Call Flows

Play an Announcement as Early Media

The numbers in the left-hand column reference the Call Flow diagram in Figure 9 (page 235).

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=1005580
780
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)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>SIP/2.0 183 Session Progress</td>
</tr>
<tr>
<td></td>
<td>Contact: sip:192.168.12.155:5060</td>
</tr>
<tr>
<td></td>
<td>Via: SIP/2.0/UDP 192.168.1.150:6100</td>
</tr>
<tr>
<td></td>
<td>To:</td>
</tr>
<tr>
<td></td>
<td><a href="">sip:anncc@192.168.12.155:5060</a>;tag=1005580 780</td>
</tr>
<tr>
<td></td>
<td>From: <a href="">sip:test0@192.168.12.153:5060</a></td>
</tr>
<tr>
<td></td>
<td>Call-ID: 27125@192.168.1.150</td>
</tr>
<tr>
<td></td>
<td>CSeq: 2 INVITE</td>
</tr>
<tr>
<td></td>
<td>Content-Type: application/sdp</td>
</tr>
<tr>
<td></td>
<td>Require: 100rel</td>
</tr>
<tr>
<td></td>
<td>RSeq: 22752</td>
</tr>
<tr>
<td></td>
<td>Content-Length: 153</td>
</tr>
<tr>
<td></td>
<td>v=0</td>
</tr>
<tr>
<td></td>
<td>o=SnowShoreUaV1 27736 410 IN IP4 192.168.12.155</td>
</tr>
<tr>
<td></td>
<td>s=SnowShore Sdp</td>
</tr>
<tr>
<td></td>
<td>t=0 0</td>
</tr>
<tr>
<td></td>
<td>m=audio 4362/1 RTP/AVP 0</td>
</tr>
<tr>
<td></td>
<td>c=IN IP4 192.168.12.156</td>
</tr>
<tr>
<td></td>
<td>a=sendrecv</td>
</tr>
<tr>
<td></td>
<td>a=ptime:20</td>
</tr>
<tr>
<td>4</td>
<td>SIP/2.0 200 OK</td>
</tr>
<tr>
<td></td>
<td>Via: Sip/2.0/UDP 192.168.1.150:6100</td>
</tr>
<tr>
<td></td>
<td>To:</td>
</tr>
<tr>
<td></td>
<td><a href="">sip:anncc@192.168.12.155:5060</a>;tag=1005580 780</td>
</tr>
<tr>
<td></td>
<td>From: <a href="">sip:test0@192.168.12.153:5060</a></td>
</tr>
<tr>
<td></td>
<td>Call-ID: 27125@192.168.1.150</td>
</tr>
<tr>
<td></td>
<td>CSeq: 3 PRACK</td>
</tr>
<tr>
<td></td>
<td>Content-Length: 0</td>
</tr>
<tr>
<td>5</td>
<td>SIP/2.0 487 Request Terminated</td>
</tr>
<tr>
<td></td>
<td>Contact: sip:192.168.12.155:5060</td>
</tr>
<tr>
<td></td>
<td>Via: SIP/2.0/UDP 192.168.12.150:6100</td>
</tr>
<tr>
<td></td>
<td>To:</td>
</tr>
<tr>
<td></td>
<td>sip:anncc@192.168.12.155:5060&gt;;tag=10055807 80</td>
</tr>
<tr>
<td></td>
<td>From: sip:4444@192.168.12.153:5060</td>
</tr>
<tr>
<td></td>
<td>Call-ID: 27125@192.168.1.150</td>
</tr>
<tr>
<td></td>
<td>CSeq: 2 INVITE</td>
</tr>
<tr>
<td></td>
<td>Content-Length: 0</td>
</tr>
<tr>
<td>6</td>
<td>ACK sip:anncc@192.168.12.155:5060 SIP/2.0</td>
</tr>
<tr>
<td></td>
<td>Via: SIP/2.0/UDP 192.168.12.150:6100</td>
</tr>
<tr>
<td></td>
<td>To:</td>
</tr>
<tr>
<td></td>
<td>sip:anncc@192.168.12.155:5060&gt;;tag=10055807 80</td>
</tr>
<tr>
<td></td>
<td>From: sip:4444@192.168.12.153:5060</td>
</tr>
<tr>
<td></td>
<td>Call-ID: 27125@192.168.1.150</td>
</tr>
<tr>
<td></td>
<td>CSeq: 2 ACK</td>
</tr>
<tr>
<td></td>
<td>Content-Length: 0</td>
</tr>
</tbody>
</table>
Call Flow for an Early Media Announcement

The following illustrates a call flow diagram for early media announcement.

Figure 9. Call Flow: Announcement, Early Media
## Playing an Announcement as Normal Media

The numbers in the left-hand column reference the diagram Figure 10 (page 238).

### Example 21. Playing an Announcement as Normal Media

```
1 sip:annnc@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:annnc@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:annnc@192.168.12.155:5060>;tag=1005580654
   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)

<table>
<thead>
<tr>
<th>Step</th>
<th>Message</th>
</tr>
</thead>
</table>
| 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:annnc@192.168.12.155:5060>;tag=10055806 54  
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:annnc@192.168.12.155:5060 SIP/2.0  
Via: SIP/2.0/UDP 192.168.1.150:6100  
To:  
  <sip:annnc@192.168.12.155:5060>;tag=10055806 54  
From: <sip:test0@192.168.12.153:5060>  
Call-ID: 27103@192.168.1.150  
CSeq: 2 ACK  
Content-Length: 0 |
| 5    | BYE sip:annnc@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=1005580 654  
From: <sip:annnc@192.168.12.155:5060>  
Call-ID: 27103@192.168.1.150  
CSeq: 3 BYE  
Supported: timer  
Supported: 100rel  
Content-Length: 0 |
| 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=1005580 654  
From: <sip:annnc@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

The following illustrates a call flow diagram for normal media announcement.

Figure 10. 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=

[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

The following illustrates a call flow diagram for stopping media using hold.

Figure 11. Call Flow: Stopping Media Using Hold
Repeating the Audio

The following illustrates sample code for repeating audio indefinitely.

Example 23. Repeating Audio Indefinitely

```xml
<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>
```
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: 10315791205158192.168.1.126
CSeq: 1327320033 INVITE
Content-Length: 201
Content-Type: application/sdp
Contact: <sip:192.168.1.126:5060;transport=udp>
Via: SIP/2.0/UDP 192.168.12.109
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 192.168.12.109
To: sip:conf=1234@192.168.12.153;tag=1031665657
From: sip:threepcc@192.168.1.126;tag=1as8ut
Call-ID: 10315791205158192.168.1.126
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
a=rtpmap:0 pcmu/8000/1
a=rtpmap:8 pcma/8000/1
```
## Call Flow for a Simple Conference (Normal Media)

<table>
<thead>
<tr>
<th>Participant1</th>
<th>Participant2</th>
<th>Participant3</th>
<th>Control Agent</th>
<th>Media Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIP INVITE To Public Conf URI</td>
<td>SIP INVITE To MS Conf URI</td>
<td>SIP INVITE To MS Conf URI</td>
<td>SIP INVITE To MS Conf URI</td>
<td>SIP INVITE To MS Conf URI</td>
</tr>
<tr>
<td>200 OK</td>
<td>200 OK</td>
<td>200 OK</td>
<td>200 OK</td>
<td>200 OK</td>
</tr>
<tr>
<td>ACK</td>
<td>ACK</td>
<td>ACK</td>
<td>ACK</td>
<td>ACK</td>
</tr>
<tr>
<td>RTP Session with Participant 1</td>
<td>RTP Session with Participant 2</td>
<td>RTP Session with Participant 3</td>
<td>RTP Session with Participant 3</td>
<td>RTP Session with Participant 3</td>
</tr>
<tr>
<td>SIP INVITE To Public Conf URI</td>
<td>SIP INVITE To Public Conf URI</td>
<td>SIP INVITE To Public Conf URI</td>
<td>SIP INVITE To Public Conf URI</td>
<td>SIP INVITE To Public Conf URI</td>
</tr>
<tr>
<td>200 OK</td>
<td>200 OK</td>
<td>200 OK</td>
<td>200 OK</td>
<td>200 OK</td>
</tr>
<tr>
<td>ACK</td>
<td>ACK</td>
<td>ACK</td>
<td>ACK</td>
<td>ACK</td>
</tr>
<tr>
<td>BYE</td>
<td>BYE</td>
<td>BYE</td>
<td>BYE</td>
<td>BYE</td>
</tr>
<tr>
<td>200 OK</td>
<td>200 OK</td>
<td>200 OK</td>
<td>200 OK</td>
<td>200 OK</td>
</tr>
</tbody>
</table>

---

**Figure 12. 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
m=audio 18000 RTP/AVP 0 101
a=rtpmap:0 pcmu/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
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"></response>
</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 13. 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

```plaintext
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 192.168.12.126:5060
To: sip:conf=1234@192.168.12.153;tag=031666539
From: sip:threepcc@192.168.1.126;tag=jtopyd
Call-ID: 1031580002825@192.168.1.126
CSeq: 2076031605 INVITE
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
<?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:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.1.126;tag=1031666918
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 192.168.12.153:5060
<?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 192.168.12.153:5060
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@192.168.1.126;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:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.1.126;tag=1031666918
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@192.168.1.126;tag=1031666918
From: sip:confbridge@192.168.1.126;tag=oi32zw
Call-ID: 1031580381757@192.168.1.126
CSeq: 1066407034 INFO
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
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"
</response>
</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
<?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'
</playcollect>
</request>
</MediaServerControl>
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.12.153:5060
From: sip:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.12.153;tag=1031666918
Response to PlayCollect
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@192.168.1.126;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 192.168.1.126:5060
From: sip:confbridge@192.168.1.126;tag=oi32zw
To: sip:conf=1234@192.168.1.126;tag=1031666918
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/tellyo
      urname.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
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:confbridge@192.168.1.126;tag=oi32zw
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
<?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:confbridge@192.168.1.126;tag=oi32zw
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
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>
```
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
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>

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
Conferences Sample Code and Call Flows

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

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

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

<?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@192.168.12.158:5060;
tag=1119358673
From: sip:control@192.168.12.158:5564;tag=1
Call-ID: 1119359020
CSeq: 1 INVITE
Contact: sip:192.168.12.158:5060
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:192.168.12.158:5060 SIP/2.0
From: sip:control@192.168.12.158:5564;tag=1
To: sip:conf=1234@192.168.12.158:5060;
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"/></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"/></teammate></configure_team></configure_leg>
```
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"/>
  </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

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"/>
  </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"/>
    </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
```
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 192.168.12.150:5060
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 192.168.12.158:5564;tag=1
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 Sample Code and Call Flows

Playing a Simple Announcement

Play Payload

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

Expected Response

```xml
<?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
<?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
<?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
<?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
<?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
<?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
<?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
<?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
<?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
<?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
<?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 14. Call Flow: IVR with MSCML
**Call Flow for PIN Collection, IVR with MSCML**

![Call Flow Diagram]

**Figure 15. 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 15**.

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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: IP Media Server's SDP |
| 4      | Response to SIP INVITE [1]  
         Method: SIP INVITE  
         Body: 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 |
<p>| 7      | Bi-directional SDP established between caller and IP Media Server |</p>
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
</table>
| [8]  | Application server directs IP Media Server to play a prompt and collect returned digits  
Method: SIP INFO  
Body: ```xml  
<?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 IP Media Server to <playcollect> request  
Method: SIP INFO  
Body: Empty |
| [10] | Return collected DTMF digits  
Method: SIP INFO  
Body: ```xml  
<?xml version="1.0"?>  
<MediaServerControl version="1.0">  
  <response request="playcollect" code="200" text="OK" reason="match" digits="1234" playduration="5230"/>  
</MediaServerControl>```  |
Method: SIP INFO  
Body: Empty |
Call Flow for Recording a Message, IVR with MSCML

Figure 16. 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 16.

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
</tr>
</thead>
</table>
| [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 |
Method: SIP INVITE  
Body: IP Media Server's SDP |
Method: SIP INVITE  
Body: IP Media Server’s SDP |
Method: SIP ACK  
Body: None |
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  
<?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="0123456789*#"/>  
</request>  
</MediaServerControl>  
``` |
| [9]    | Response from IP Media Server to <playrecord> request  
Method: SIP INFO  
Body: Empty |
| [10]   | Recording is written to the external storage location specified in the url parameter of the <record> request. |
Call Flow for MSCML MRCP Session Management

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 assumes 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 following is an example of the MSCML payload:

```xml
<?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>
```

[11] Response from IP Media Server to application server when recording is complete
Method: SIP INFO
Body:
```xml
<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
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:32ADCB234338@speechrecog
  a=cmid:1
  m=audio 48260 RTP/AVP 0 101
  a=rtpmap:0 pcmu/8000
```
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:a8b4b76e66710
CSeq:314162 ACK
Content-Length:0

2 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.

3 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">
4 The MRCP server responds that recognition is in progress.

MRCP/2.0 48 32614 200 IN-PROGRESS
Channel-Identifier:32AECB23433801@speechrecog

5 The application requests the media server to play a sequence of prompts by sending a request like the following.

```xml
<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>
```

6 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

7 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
<?xml version="1.0" encoding="UTF-8"?>
<MediaServerControl version="1.0">
  <request>
    <stop id="5678"/>
  </request>
</MediaServerControl>
```

8 The IP Media Server sends a response to the `<stop>` request.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<MediaServerControl version="1.0">
```
9 The IP Media Server sends a response for the stopped `<play>` request similar to the following:

```xml
<response id="5678" request="stop" code="200"
  text="OK"/>
</MediaServerControl>
```

10 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
<?xml version="1.0"?>
<result grammar="session:request1@form-level.store">
  <interpretation>
    <instance name="Person">
      <Person>
        <Name> Alex Michaels</Name>
      </Person>
    </instance>
  </interpretation>
</result>
```

11 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
```

12 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
<?xml version="1.0"?>
<result grammar="session:request1@form-level.store">
  <interpretation>
    <instance name="Person">
```

```
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.
VoiceXML Sample Code and Call Flows

Playing an Announcement

The sample code 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
<?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
<?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>
  </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"/>
    </if>
    <else/>
    <clear namelist="pin confirm"/>
  </filled>
</field>
</form>
</vxml>
Call Flow for VoiceXML

The following illustrates a call flow diagram for VoiceXML.

![Call Flow Diagram for VoiceXML](image)

**Figure 17. 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
<?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 between the caller and -->
<!-- the callee. -->
<xml 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>
  </transfer>

  <filled>
    <if cond="makethecall == 'busy'">
      <prompt>
        <audiosrc="http://10.10.10.111/snowshore/prompts/gmvoices/2.ris" />
      </prompt>
    </if>
    <elseif cond="makethecall == 'noanswer'"/>
    <prompt>
      <audio src="http://10.10.10.111/snowshore/prompts/gmvoices/3.ris" />
    </prompt>
  </filled>

  <filled>
    <if cond="makethecall == 'near_end_disconnect'">
      <prompt>
        <audio src="http://10.10.10.111/snowshore/prompts/gmvoices/4.ris" />
      </prompt>
    </if>
    <elseif cond="makethecall == 'far_end_disconnect'"/>
    <prompt>
      <audio src="http://10.10.10.111/snowshore/prompts/gmvoices/5.ris" />
    </prompt>
  </filled>

  <if cond="makethecall == 'network_disconnect'">
    <prompt>
      <audio src="http://10.10.10.111/snowshore/prompts/gmvoices/6.ris" />
    </prompt>
  </if>
</form>
```
</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>
</form>
</block>
<audio src="http://10.10.10.111/snowshore/prompts/gmvoices/star.neu" />
</block>
</form>
</vxml>
**T.30 Fax Detection**

The T.30 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:** The T.30 fax detection feature is implemented only in the VoiceXML browser.

**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 following the figure.

---

**Figure 18. T.30 Fax Call Transfer**
The following table contains a detailed description of the call flow for the T.30 Fax Call Transfer scenario.

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Sending Fax machine initiates an analog call (dialed number is 3010) to the Analog-to-IP Adapter.</td>
</tr>
<tr>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>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.</td>
</tr>
<tr>
<td>4</td>
<td>The CNG tone is detected by the IP Media Server and a notification event (CNG) is sent to the VXML script. In response to this event, the VXML script performs a Fax call <code>&lt;transfer&gt;</code> specifying <code>bridge=&quot;forward&quot;</code> 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 IP Media Server to hairpin together the Receiving Fax and the Sending Fax RTP streams.</td>
</tr>
<tr>
<td>5</td>
<td>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.</td>
</tr>
<tr>
<td>6</td>
<td>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.</td>
</tr>
<tr>
<td>7</td>
<td>The VXML script terminates when the Sending Fax RTP stream closes which closes the Receiving Fax stream.</td>
</tr>
</tbody>
</table>
CNG Tone Detection and Event Notification

The VXML 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 Implementation

The **com.snowshore.signal** property 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>
</catch>
```

**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>
</catch>
```

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
<?xml version="2.0" encoding="UTF-8"?>

<vxml version="2.0">

  <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>
    </field>

    <prompt bargein="false">
    </prompt>

    <help count="1" cond="dtmf">
      since the condition is set to false, this handler will be skipped
    </help>

    <catch event="com.snowshore.signal">
      <if cond="_message == 'CNG'">
        <!—Perform Fax Call Transfer when CNG tone is detected -->
      </if>
    </catch>

  </form>

</vxml>
```
  </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>
T.38 Fax Detection, Termination, and Initiation

The fax termination server is incorporated into the IP Media Server using the Dialogic® Brooktrout® T.38 host fax termination software. Refer to the IP Media Server Installation and Operations Guide for instructions on configuring the fax software.

Detection and Termination

This feature enables the 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 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.

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

Once 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 54. T.38 Fax Parameters |
|-------------------------------|------------------|------------------|-------------|
| Parameter                     | Description      | Default in decimal | Range      |
| SR140APP_PORT                 | First port for the T.38 fax server | 9000             | 9000-40000  |
Fax Call Termination Call Flow

Figure 18 outlines the call flow between the initiating (sending) fax server and the terminating (receiving) fax server. Each step is explained following the figure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default in decimal</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR140APP_PROXY_PORT</td>
<td>Host name for proxy fax SIP Server</td>
<td>5060</td>
<td></td>
</tr>
<tr>
<td>SR140APP_PROXY_IP</td>
<td>Required field for sending fax</td>
<td>No default</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 19. T.38 Fax Call Transfer**
The following is a detailed description of the call flow for the T.38 Fax Call Transfer scenario.

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Sending Fax server initiates an analog call (dialed number is 3010) to the Analog-to-IP Adapter.</td>
</tr>
<tr>
<td>2</td>
<td>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).</td>
</tr>
<tr>
<td>3</td>
<td>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.</td>
</tr>
<tr>
<td>4</td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td>The CNG tone is detected by the Media Server 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 <code>&lt;record&gt;</code> (specifying dest_enc=&quot;tiff&quot;). This results in a RE-INVITE request sent to 192.168.12.203 with the new RTP port for the fax data.</td>
</tr>
<tr>
<td>6</td>
<td>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.</td>
</tr>
<tr>
<td>7</td>
<td>The VXML script sends the fax tiff file to a file server.</td>
</tr>
</tbody>
</table>

**CNG Tone Detection and Event Notification**

The VXML 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 Media Server application enables and disables the CNG tone detection through the Media Server Protocol (MSP) command's Call Progress Analysis parameters. In addition, the Media Server application provides event notification back to the VXML browser / application.

**VoiceXML Implementation**

The `com.snowshore.signal` property 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:

```xml
<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 VoiceXML 2.0 example:

```xml
<catch event="com.snowshore.signal">
  <if cond="_message == 'cng'">
    <!--Perform Fax Call Transfer when CNG tone is detected -->
  </if>
</catch>
```

For VoiceXML 1.0 syntax, replace 'cng' with 'CNG' in the "_message" variable in the above example.

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:

```xml
<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 VoiceXML 2.0 syntax to enable CNG tone detection and catch the event.

For VoiceXML 1.0 syntax, replace 'cng' with 'CNG' in the "_message" variable:

```xml
<if cond="_message == 'CNG'">
  <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'">
          <!-- perform fax call transfer when CNG is detected -->
        </if>
      </catch>
    </field1>
    <!-- This field waits for the CNG tone -->
  </form>
</if>
```
....
</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 server 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 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 2.0 FAX Call Record script. The script demonstrates use of FAX Call Record and HTTP post of the file.

```xml
<vxml version="2.0">
<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>
```
Using VCR and Gain Controls

The following properties are used for VCR and gain controls in the VoiceXML 2.0 browser:

- `com.snowshore.vcr_skip`: Contains the amount of time to skip ahead or behind when one of the 'vcr_fwd' or 'vcr_rew' grammars are matched. This value is a Time Designation type defined in section 6.5 of the VoiceXML 2.0 specification and has a default value of '5s' if the property is not set.

- `com.snowshore.gain_adj`: Contains the amount of gain to adjust the audio playback when 'gain_up' or 'gain_down' grammars are matched. This value is in dB and has a default value of '3' if the property is not set.

- `com.snowshore.vcr_pause`: Contains the amount of time to pause a prompt when the 'vcr_pause' grammar is matched. This value is a Time Designation type defined in section 6.5 of the VoiceXML 2.0 specification and has a default value of '3s' if the property is not set.

- `com.snowshore.vcr_speed_adj`: Contains the approximate percentage change in speed that the IP Media Server will attempt to play back the audio prompt when one of the 'vcr_faster' or 'vcr_slower' grammars are matched. The actual percentage change in speed depends on the content of the audio prompt and the system environment; the IP Media Server places emphasis on the audio quality produced.

For an example script, see "Using VCR and Gain Controls in VoiceXML 2.0" (page 305).

**Note:** VCR controls for video files are supported through the Real Time Streaming Protocol (RTSP) only. VCR controls for all other video calls will cause the video file to be played back from the beginning; only the gain adjustment has effect and is applied to the audio level at the restart of the play.

Table 55 describes the VCR and gain controls that are supported through the RTSP in VoiceXML 2.0. For an example script, see "Using VCR Controls for RTSP in VoiceXML 2.0" (page 306).

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>RTSP Support</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>com.snowshore.vcr_skip</code></td>
<td>fast forward and rewind</td>
<td>Yes</td>
</tr>
<tr>
<td><code>com.snowshore.gain_adj</code></td>
<td>gain adjustment</td>
<td>No</td>
</tr>
<tr>
<td><code>com.snowshore.vcr_pause</code></td>
<td>pause</td>
<td>Yes</td>
</tr>
<tr>
<td><code>com.snowshore.speed_adj</code></td>
<td>speed adjustment</td>
<td>No</td>
</tr>
</tbody>
</table>

The VoiceXML 2.0 browser supports built-in grammars to support VCR and gain controls:

```xml
<grammar mode="dtmf" src="builtin:dtmf/vcr"/>
<grammar mode="dtmf" src="builtin:dtmf/gain"/>
```
The following is an example script using VCR and gain control in VoiceXML 2.0.

Example 45. Using VCR and Gain Controls in VoiceXML 2.0

```xml
<?xml version="1.0"?><vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">

<property name="com.snowshore.gain_adj" value="10"/>
<property name="com.snowshore.vcr_skip" value="5s"/>
<property name="com.snowshore.vcr_pause" value="3s"/>
<property name="com.snowshore.vcr_speed_adj" value="5"/>

<form id="g">
  <field name="u">
    <link event="modify_audio_control">
      <grammar src="builtin:dtmf/gain?down=7;up=8;reset=9"/>
    </link>
    <link event="modify_audio_control">
      <grammar src="builtin:dtmf/vcr?rew=1;fwd=2;pause=3;slower=4;faster=5"/>
    </link>
    <prompt bargein="true" vcr="true">
      <audio src="file:///opt/showshore/prompts/generic/en_US/num_changed.ulaw"/>
      <audio src="http://localhost/showshore/prompts/number.wav"/>
      <audio src="http://localhost/showshore/prompts/alpha.wav"/>
    </prompt>
    <catch event="modify_audio_control">
      <clear/>
      <reprompt/>
    </catch>
  </field>
</form>
</vxml>
```
The following is an example script using VCR controls for the RTSP in VoiceXML 2.0.

**Example 46. Using VCR Controls for RTSP in VoiceXML 2.0**

```xml
<?xml version="1.0"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">

    <property name='com.snowshore.vcr_skip' value='6s'/>
    <property name='com.snowshore.vcr_pause' value='5s'/>

    <form id="g">
        <field type="boolean" name="u">
            <link event="modify_video_control">
                <grammar src="builtin:dtmf/vcr?rew=0;fwd=6;pause=3"/>
            </link>
            <prompt bargein='true' vcr='true'>
                <audio src="rtsp://rtspserver.dialogic.com/sample.3gp"/>
            </prompt>

            <catch event="modify_video_control">
                <clear />
                <reprompt />
            </catch>
        </field>
    </form>
</vxml>
The following is an example script using VCR controls in VoiceXML 1.0.

Example 47. Using VCR Controls in VoiceXML 1.0

```xml
<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">
            <!-- The message audio -->
            </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>
            </prompt>
            filled>
                <!-- Fetch next thing to do -->
                <submit next="msgfetch?sid=12345&msgid=42&digit=$field1"/>
            </filled>
        </field>
    </form>
```
Image Overlay and Text Overlay

This section provides VoiceXML 2.0 text and image overlay sample code.

Simple Text Overlay

This script illustrates a simple text overlay using VoiceXML 2.0. Modify the script, such as audio source file and font name, to suit your implementation.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  <form id="form1">
    <block>
      <prompt bargein="false">
        <audio src="file:///opt/snowshore/htdocs/3GPP/sample1.3gp"/>
        <overlay position="x0y240" size="x346y20">
          <content>
            <font name="/opt/snowshore/fonts/FreeSans.ttf" size="14" fgcolor="yellow"/>
            "Hello, world!"
          </content>
        </overlay>
      </prompt>
    </block>
  </form>
</vxml>
```

Scrolling Text Overlay

This script illustrates a scrolling text overlay using VoiceXML 2.0. Modify the script, such as audio source file and font name, to suit your implementation.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  <form id="form1">
    <block>
      <prompt bargein="false">
        <audio src="file:///opt/snowshore/htdocs/3GPP/sample2.3gp"/>
        <!--First overlay defines the outer border and does not use scrolling--> 
        <overlay position="x0%y20%" size="x100%y10%" border-width="2" border-color="yellow"bgcolor="#0xFFFF007F">
          <content> " " </content>
        </overlay>
        <!--Second overlay is inside first overlay and will scroll--> 
        <overlay position="x2%y22%" size="x96%y6%" layer="1">
          <content>
            <font name="/opt/snowshore/fonts/FreeSans.ttf" size="14" fgcolor="black"/>
            <scroll mode="continuous" speed="40" direction="left"/>
            "The quick brown fox jumps over the lazy dog."
          </content>
        </overlay>
      </prompt>
    </block>
  </form>
</vxml>
```
Image Overlay

This script illustrates a simple image overlay using VoiceXML 2.0. Modify the script, such as image source file and file type, to suit your implementation.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  <form id="form1">
    <block>
      <prompt bargein="false">
        <audio src="file:///opt/snowshore/htdocs/3GPP/sample2.3gp"/>
        <overlay position="x0y60" size="x180y80">
          <content type="image/jpeg" src="file:///opt/snowshore/htdocs/ubuntu.jpg" />
        </overlay>
      </prompt>
    </block>
  </form>
</vxml>
```
Late Media Sample Code

The sample code below is for the Late Media feature.

Example 48. Late Media

```
INVITE sip:conf=1@192.168.16.150 SIP/2.0
Via: SIP/2.0/UDP 10.128.41.4:5060;branch=z9hG4bKB710DA6E24074155BD75A591F68C4101;rport
From: "Your Full Name" <sip:Username@10.128.41.4:5060>;tag=A5C31AD26902A99B6B818E7C9A97079B
To: <sip:conf=1@192.168.16.150>
Contact: <sip:Username@10.128.41.4:5060;transport=udp>
Call-ID: 9B1E82814DDCA7DCF06764DC14AEF68DE010.128.41.4
User-Agent: Kapanga Softphone Desktop Windows 1.00/2182d+128820401_00FF88241B88_7595AB69C3CA5
Supported: timer, replaces
CSeq: 1 INVITE
Max-Forwards: 70
Session-Expires: 1800;refresher=uac
Allow: INVITE, INFO, PRACK, ACK, BYE, CANCEL, OPTIONS, NOTIFY, REGISTER, SUBSCRIBE, REFER, PUBLISH, UPDATE, MESSAGE
Content-Length: 0

SIP/2.0 100 Trying
From: "Your Full Name" <sip:Username@10.128.41.4:5060>;tag=A5C31AD26902A99B6B818E7C9A97079B
To: <sip:conf=1@192.168.16.150>
Call-ID: 9B1E82814DDCA7DCF06764DC14AEF68DE010.128.41.4
CSeq: 1 INVITE
Via: SIP/2.0/UDP 10.128.41.4:5060;rport=5060;branch=z9hG4bKB710DA6E24074155BD75A591F68C4101
Supported: 100rel,timer
Require: timer
Contact: <sip:192.168.16.150:5060>
Content-Length: 0

SIP/2.0 200 OK
From: "Your Full Name" <sip:Username@10.128.41.4:5060>;tag=A5C31AD26902A99B6B818E7C9A97079B
To: <sip:conf=1@192.168.16.150>;tag=495b7d8-9610a8c0-13c4-50022-157e-18461c0-157e
Call-ID: 9B1E82814DDCA7DCF06764DC14AEF68DE010.128.41.4
CSeq: 1 INVITE
Allow: INVITE, CANCEL, ACK, BYE, OPTIONS, INFO, PRACK
Via: SIP/2.0/UDP 10.128.41.4:5060;ack=5060;branch=z9hG4bKB710DA6E24074155BD75A591F68C4101
Supported: 100rel,timer
Require: timer
Contact: <sip:192.168.16.150:5060>
```
Session-Expires: 180; refresher=uac
Content-Type: application/sdp
Content-Length: 517

v=0
o=snowshoreSDP 2890844526 2890844526 IN IP4 192.168.16.150
s=snowshoreSDP
i=IN IP4 192.168.16.150
t=0
m=audio 6000/1 RTP/AVP 18 2 8 96 0 101
a=rtmap:18 G729/8000
a=fmtp:18 annexb=no
a=rtmap:2 G726-32/8000
a=rtmap:8 PCMA/8000
a=rtmap:96 AMR/8000
a=fmtp:96 octet-align=0; mode-set=7
a=rtmap:0 PCMUS/8000
a=rtmap:101 telephone-event/8000/1
a=sendrecv
a=ptime:20
m=video 6002/1 RTP/AVP 34
a=rtmap:34 H263/90000
a=sendrecv
a=fmtp:34 QCIF=2

ACK sip:192.168.16.150:5060 SIP/2.0
Via: SIP/2.0/UDP
    10.128.41.4:5060;branch=z9hG4bKB710DA6E24074155BD75A591F6
8C4101
From: "Your Full Name"
    <sip:Username@10.128.41.4:5060>;tag=A5C31AD26902A99B6B818
    E7C9A97097B
To: <sip:conf=1@192.168.16.150>;tag=495b7d8-9610a8c0-13c4-50022-1574e-488461c0-1574e
Contact: <sip:Username@10.128.41.4:5060;transport=udp>
Call-ID: 9B1E828144DBA7DCF06764DC148E68BE010.128.41.4
User-Agent: Kapanga Softphone Desktop Windows
    1.00/2182d+1288207401_00FF88241B88_705AB69C3CA5
CSeq: 1 ACK
Max-Forwards: 70
Content-Type: application/sdp
Content-Length: 284
v=0
o=Username 1297963048 1297963099 IN IP4 10.128.41.4
s=Kapanga [1297963048]
c=IN IP4 10.128.41.4
t=0
m=audio 5100 RTP/AVP 18 0 101
a=rtmap:18 G729/8000
a=fmtp:18 annexb=no
a=sendrecv
a=maxptime:20
a=ptime:20
a=rtmap:0 PCMUS/8000
a=rtmap:101 telephone-event/8000/1
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 includes the following major sections:

- Sound Library
- Generic Audio Files
Sound Library

The 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
- Numbers
- Letters
- Time and Money
- Press Keys
- Quantities
- Miscellaneous Words

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 56. Prompt Library: Standard Phrases

<table>
<thead>
<tr>
<th>Filename</th>
<th>Spoken as</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg001</td>
<td>one second of silence</td>
</tr>
<tr>
<td>msg002</td>
<td>600Hz beep tone</td>
</tr>
</tbody>
</table>
Table 56. Prompt Library: Standard Phrases (continued)

<table>
<thead>
<tr>
<th>Filename</th>
<th>Spoken as</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg003</td>
<td>Goodbye.</td>
</tr>
<tr>
<td>msg004</td>
<td>Hello.</td>
</tr>
<tr>
<td>msg005</td>
<td>Good morning.</td>
</tr>
<tr>
<td>msg006</td>
<td>Good afternoon.</td>
</tr>
<tr>
<td>msg007</td>
<td>Good evening.</td>
</tr>
<tr>
<td>msg008</td>
<td>If this is correct...</td>
</tr>
<tr>
<td>msg009</td>
<td>To transfer to another extension...</td>
</tr>
<tr>
<td>msg010</td>
<td>To return to the main menu...</td>
</tr>
<tr>
<td>msg011</td>
<td>To return to the previous menu...</td>
</tr>
<tr>
<td>msg012</td>
<td>To delete this message...</td>
</tr>
<tr>
<td>msg013</td>
<td>To delete all messages...</td>
</tr>
<tr>
<td>msg014</td>
<td>To exit the system...</td>
</tr>
<tr>
<td>msg015</td>
<td>To end this call...</td>
</tr>
<tr>
<td>msg016</td>
<td>To cancel delivery of this message...</td>
</tr>
<tr>
<td>msg017</td>
<td>To send your message now...</td>
</tr>
<tr>
<td>msg018</td>
<td>Please hold while your call is being transferred.</td>
</tr>
<tr>
<td>msg019</td>
<td>Please leave your message after the tone.</td>
</tr>
<tr>
<td>msg020</td>
<td>Please begin recording after the tone.</td>
</tr>
<tr>
<td>msg021</td>
<td>Thank you for calling.</td>
</tr>
<tr>
<td>msg022</td>
<td>Thank you for calling, goodbye.</td>
</tr>
<tr>
<td>msg023</td>
<td>Thank you.</td>
</tr>
<tr>
<td>msg024</td>
<td>Please enter your mailbox number.</td>
</tr>
<tr>
<td>msg025</td>
<td>Please enter your passcode.</td>
</tr>
<tr>
<td>msg026</td>
<td>I'm sorry, that passcode is invalid. Please reenter your passcode.</td>
</tr>
<tr>
<td>msg027</td>
<td>I'm sorry, that mailbox number is invalid. Please reenter your mailbox number.</td>
</tr>
<tr>
<td>msg028</td>
<td>I'm sorry, that is an invalid entry. Please try again.</td>
</tr>
<tr>
<td>msg029</td>
<td>You have no messages.</td>
</tr>
<tr>
<td>msg030</td>
<td>You have no more messages.</td>
</tr>
<tr>
<td>msg031</td>
<td>You have...</td>
</tr>
<tr>
<td>msg032</td>
<td>...new message.</td>
</tr>
</tbody>
</table>
Table 56. Prompt Library: Standard Phrases (continued)

<table>
<thead>
<tr>
<th>Filename</th>
<th>Spoken as</th>
</tr>
</thead>
<tbody>
<tr>
<td>msg033</td>
<td>...new messages.</td>
</tr>
<tr>
<td>msg034</td>
<td>...saved message.</td>
</tr>
<tr>
<td>msg035</td>
<td>...saved messages.</td>
</tr>
<tr>
<td>msg036</td>
<td>Your mailbox is currently full.</td>
</tr>
<tr>
<td>msg037</td>
<td>Message deleted.</td>
</tr>
<tr>
<td>msg038</td>
<td>Message saved.</td>
</tr>
<tr>
<td>msg039</td>
<td>Please hold.</td>
</tr>
<tr>
<td>msg040</td>
<td>Please hold for assistance.</td>
</tr>
<tr>
<td>msg041</td>
<td>I'm sorry...</td>
</tr>
<tr>
<td>msg042</td>
<td>You entered...</td>
</tr>
<tr>
<td>msg043</td>
<td>End of messages.</td>
</tr>
<tr>
<td>msg044</td>
<td>We cannot identify that entry. Please try again.</td>
</tr>
<tr>
<td>msg045</td>
<td>Please hold while I transfer your call to the operator.</td>
</tr>
<tr>
<td>msg046</td>
<td>We're sorry you are having difficulty. Please try your call again later.</td>
</tr>
<tr>
<td>msg047</td>
<td>If this is correct, press 1. If not, press 2.</td>
</tr>
<tr>
<td>msg048</td>
<td>Please enter your passcode now.</td>
</tr>
<tr>
<td>msg049</td>
<td>If you're calling from a touchtone phone, press 1 now. Otherwise, please stay on the line to speak with an operator.</td>
</tr>
<tr>
<td>msg050</td>
<td>Please enter your business phone number beginning with the area code now.</td>
</tr>
<tr>
<td>msg051</td>
<td>You have selected...</td>
</tr>
<tr>
<td>msg052</td>
<td>You have reached...</td>
</tr>
<tr>
<td>msg053</td>
<td>To make another selection...</td>
</tr>
<tr>
<td>msg054</td>
<td>To try again...</td>
</tr>
<tr>
<td>msg055</td>
<td>Extension...</td>
</tr>
<tr>
<td>msg056</td>
<td>The telephone number you entered is...</td>
</tr>
<tr>
<td>msg057</td>
<td>The fax number you entered is...</td>
</tr>
<tr>
<td>msg058</td>
<td>Please enter the account number.</td>
</tr>
<tr>
<td>msg059</td>
<td>Please enter your personal identification number.</td>
</tr>
<tr>
<td>msg060</td>
<td>Your personal identification number is...</td>
</tr>
</tbody>
</table>
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 57. Prompt Library: Cardinal Numbers

<table>
<thead>
<tr>
<th>Filename</th>
<th>Numbers</th>
<th>Extension</th>
<th>Spoken As</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.neu ...</td>
<td>through 9.neu</td>
<td>.neu</td>
<td>Zero through Nine with neutral, rising, or falling intonation</td>
</tr>
<tr>
<td>0.ris ...</td>
<td>through 9.ris</td>
<td>.ris</td>
<td></td>
</tr>
<tr>
<td>0.dwn ...</td>
<td>through 9.dwn</td>
<td>.dwn</td>
<td></td>
</tr>
</tbody>
</table>
Table 57. Prompt Library: Cardinal Numbers (continued)

<table>
<thead>
<tr>
<th>Filename</th>
<th>Numbers</th>
<th>Extension</th>
<th>Spoken As</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.num, 11.num, 12.num... through 100.num</td>
<td>10 through 100</td>
<td>.num</td>
<td>Ten, Eleven, Twelve,... One hundred’</td>
</tr>
<tr>
<td>100.neu ... through 1000.neu</td>
<td>100 through 1000 in increments of 100</td>
<td>.neu</td>
<td>One hundred through One thousand with neutral, rising, or falling intonation.</td>
</tr>
<tr>
<td>100.ris ... through 1000.ris</td>
<td></td>
<td>.ris</td>
<td></td>
</tr>
<tr>
<td>100.dwn ...through 1000.dwn</td>
<td></td>
<td>.dwn</td>
<td></td>
</tr>
<tr>
<td>thousand.num</td>
<td>thousand, million, billion, and trillion</td>
<td>.num</td>
<td>Thousand, Million, Billion, Trillion</td>
</tr>
<tr>
<td>million.num</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>billion.num</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trillion.num</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dates and Ordinal Numbers
The following files consists of dates and ordinal numbers.

Table 58. Prompt Library: Dates and Ordinal Numbers

<table>
<thead>
<tr>
<th>Filename</th>
<th>Days/Months</th>
<th>Spoken As</th>
</tr>
</thead>
<tbody>
<tr>
<td>mon.dat</td>
<td>Monday through Sunday</td>
<td>Day of the week, Monday through Sunday</td>
</tr>
<tr>
<td>tue.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>wed.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thu.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fri.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sat.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sun.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1st.dat, 2nd.dat, 3rd.dat... through 31st.dat</td>
<td>1st, 2nd, 3rd... through 31st</td>
<td>Spoken as ordinal number First, Second, Fourth, Thirty-first</td>
</tr>
<tr>
<td>jan.dat</td>
<td>January through December</td>
<td>Name of month January, February, etc.</td>
</tr>
<tr>
<td>feb.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mar.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>apr.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>may.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jun.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jul.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aug.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sep.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oct.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nov.dat</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dec.dat</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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:

```plaintext
/opt/snowshore/prompts/gmvoices/en_US/susan/
```

and in

```plaintext
/opt/snowshore/prompts/gmvoices/zh_CN/sara/
```

<table>
<thead>
<tr>
<th>Filename</th>
<th>Days/Months</th>
<th>Spoken As</th>
</tr>
</thead>
<tbody>
<tr>
<td>1975.dat ... through 2015.dat</td>
<td>1975 through 2015</td>
<td>Spoken as year. Nineteen seventy five, ’Two thousand and fifteen</td>
</tr>
</tbody>
</table>
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 59. Prompt Library: Time and Money Phrases

<table>
<thead>
<tr>
<th>Filename</th>
<th>Spoken as Time Zones:</th>
</tr>
</thead>
<tbody>
<tr>
<td>est.tim</td>
<td>Eastern Standard Time</td>
</tr>
<tr>
<td>cst.tim</td>
<td>Central Standard Time</td>
</tr>
<tr>
<td>mst.tim</td>
<td>Mountain Standard Time</td>
</tr>
<tr>
<td>pst.tim</td>
<td>Pacific Standard Time</td>
</tr>
<tr>
<td>am.tim</td>
<td>AM</td>
</tr>
<tr>
<td>at.tim</td>
<td>at (time)</td>
</tr>
<tr>
<td>mid.tim</td>
<td>Midnight</td>
</tr>
<tr>
<td>noon.tim</td>
<td>Noon</td>
</tr>
<tr>
<td>oclock.tim</td>
<td>o’clock</td>
</tr>
<tr>
<td>pm.tim</td>
<td>PM</td>
</tr>
<tr>
<td>and.mon</td>
<td>and (money)</td>
</tr>
<tr>
<td>cent.mon</td>
<td>Cent</td>
</tr>
<tr>
<td>cents.mon</td>
<td>Cents</td>
</tr>
<tr>
<td>dollar.mon</td>
<td>Dollar</td>
</tr>
<tr>
<td>dollars.mon</td>
<td>Dollars</td>
</tr>
</tbody>
</table>

### 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/`
- `/opt/snowshore/prompts/gmvoices/zh_CN/sara/`

Table 60. Prompt Library: Press Key Phrases

<table>
<thead>
<tr>
<th>Filename</th>
<th>Spoken as Press Key Phrases</th>
</tr>
</thead>
<tbody>
<tr>
<td>press</td>
<td>Press</td>
</tr>
<tr>
<td>prs0.dwn / prs0.neu</td>
<td>Press 0</td>
</tr>
<tr>
<td>prs1.dwn / prs1.neu</td>
<td>Press 1</td>
</tr>
<tr>
<td>prs2.dwn / prs2.neu</td>
<td>Press 2</td>
</tr>
<tr>
<td>prs3.dwn / prs3.neu</td>
<td>Press 3</td>
</tr>
<tr>
<td>prs4.dwn / prs4.neu</td>
<td>Press 4</td>
</tr>
<tr>
<td>prs5.dwn / prs5.neu</td>
<td>Press 5</td>
</tr>
<tr>
<td>prs6.dwn / prs6.neu</td>
<td>Press 6</td>
</tr>
<tr>
<td>prs7.dwn / prs7.neu</td>
<td>Press 7</td>
</tr>
<tr>
<td>prs8.dwn / prs8.neu</td>
<td>Press 8</td>
</tr>
<tr>
<td>prs9.dwn / prs9.neu</td>
<td>Press 9</td>
</tr>
<tr>
<td>asterisk.dn /asterisk.neu</td>
<td>Asterisk</td>
</tr>
<tr>
<td>pound.dwn / pound.neu</td>
<td>Pound</td>
</tr>
<tr>
<td>star.dwn / star.neu</td>
<td>Star</td>
</tr>
</tbody>
</table>
# Quantities

Quantity audio files are located in:

```
/opt/snowshore/prompts/gmvoices/en_US/susan/
```

and in

```
/opt/snowshore/prompts/gmvoices/zh_CN/sara/
```

### Table 61. Prompt Library: Quantities

<table>
<thead>
<tr>
<th>Filename</th>
<th>Spoken as</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>year.neu</td>
<td>year</td>
<td>neutral</td>
</tr>
<tr>
<td>years.neu</td>
<td>years</td>
<td>neutral</td>
</tr>
<tr>
<td>month.neu</td>
<td>month</td>
<td>neutral</td>
</tr>
<tr>
<td>months.neu</td>
<td>months</td>
<td>neutral</td>
</tr>
<tr>
<td>day.neu</td>
<td>day</td>
<td>neutral</td>
</tr>
<tr>
<td>days.neu</td>
<td>days</td>
<td>neutral</td>
</tr>
<tr>
<td>hour.neu</td>
<td>hour</td>
<td>neutral</td>
</tr>
<tr>
<td>hours.neu</td>
<td>hours</td>
<td>neutral</td>
</tr>
<tr>
<td>minute.neu</td>
<td>minute</td>
<td>neutral</td>
</tr>
<tr>
<td>minutes.neu</td>
<td>minutes</td>
<td>neutral</td>
</tr>
<tr>
<td>second.neu</td>
<td>second</td>
<td>neutral</td>
</tr>
<tr>
<td>seconds.neu</td>
<td>seconds</td>
<td>neutral</td>
</tr>
<tr>
<td>dollar.neu</td>
<td>dollar</td>
<td>neutral</td>
</tr>
<tr>
<td>dollars.neu</td>
<td>dollars</td>
<td>neutral</td>
</tr>
<tr>
<td>cent.neu</td>
<td>cent</td>
<td>neutral</td>
</tr>
<tr>
<td>cents.neu</td>
<td>cents</td>
<td>neutral</td>
</tr>
<tr>
<td>minus.dwn</td>
<td>minus</td>
<td>neutral</td>
</tr>
<tr>
<td>year.dwn</td>
<td>year</td>
<td>final intonation</td>
</tr>
<tr>
<td>years.dwn</td>
<td>years</td>
<td>final intonation</td>
</tr>
<tr>
<td>month.dwn</td>
<td>month</td>
<td>final intonation</td>
</tr>
<tr>
<td>months.dwn</td>
<td>months</td>
<td>final intonation</td>
</tr>
<tr>
<td>day.dwn</td>
<td>day</td>
<td>final intonation</td>
</tr>
<tr>
<td>days.dwn</td>
<td>days</td>
<td>final intonation</td>
</tr>
<tr>
<td>hour.dwn</td>
<td>hour</td>
<td>final intonation</td>
</tr>
<tr>
<td>hours.dwn</td>
<td>hours</td>
<td>final intonation</td>
</tr>
</tbody>
</table>
Table 61. Prompt Library: Quantities (continued)

<table>
<thead>
<tr>
<th>Filename</th>
<th>Spoken as</th>
</tr>
</thead>
<tbody>
<tr>
<td>minute.dwn</td>
<td>minute final intonation</td>
</tr>
<tr>
<td>minutes.dwn</td>
<td>minutes final intonation</td>
</tr>
<tr>
<td>second.dwn</td>
<td>second final intonation</td>
</tr>
<tr>
<td>seconds.dwn</td>
<td>seconds final intonation</td>
</tr>
<tr>
<td>dollar.dwn</td>
<td>dollar final intonation</td>
</tr>
<tr>
<td>dollars.dwn</td>
<td>dollars final intonation</td>
</tr>
<tr>
<td>cent.dwn</td>
<td>cent final intonation</td>
</tr>
<tr>
<td>cents.dwn</td>
<td>cents final intonation</td>
</tr>
<tr>
<td>minus.dwn</td>
<td>minus final intonation</td>
</tr>
</tbody>
</table>

Miscellaneous Words

Miscellaneous words are located in:

```
/opt/snowshore/prompts/gmvoices/en_US/susan/
```

and in

```
/opt/snowshore/prompts/gmvoices/zh_CN/sara/
```

**Note:** Many of these files do not have filename extensions.

Table 62. Prompt Library: Miscellaneous Words

<table>
<thead>
<tr>
<th>Filename</th>
<th>Spoken as</th>
</tr>
</thead>
<tbody>
<tr>
<td>act</td>
<td>Activate</td>
</tr>
<tr>
<td>acted</td>
<td>Activated</td>
</tr>
<tr>
<td>dash</td>
<td>Dash</td>
</tr>
<tr>
<td>deact</td>
<td>Deactivate</td>
</tr>
<tr>
<td>deacted</td>
<td>Deactivated</td>
</tr>
<tr>
<td>for</td>
<td>For</td>
</tr>
<tr>
<td>no</td>
<td>No</td>
</tr>
<tr>
<td>none</td>
<td>None</td>
</tr>
<tr>
<td>o.dwn</td>
<td>O</td>
</tr>
<tr>
<td>o.neu</td>
<td></td>
</tr>
<tr>
<td>o.ris</td>
<td></td>
</tr>
</tbody>
</table>
Table 62. Prompt Library: Miscellaneous Words (continued)

<table>
<thead>
<tr>
<th>Filename</th>
<th>Spoken as</th>
</tr>
</thead>
<tbody>
<tr>
<td>off.dwn</td>
<td>Off</td>
</tr>
<tr>
<td>off.neu</td>
<td></td>
</tr>
<tr>
<td>on.dwn</td>
<td>On</td>
</tr>
<tr>
<td>on.neu</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>Or</td>
</tr>
<tr>
<td>percent</td>
<td>Percent</td>
</tr>
<tr>
<td>point</td>
<td>Point</td>
</tr>
<tr>
<td>sent</td>
<td>Sent</td>
</tr>
</tbody>
</table>
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 63. Generic Prompt Phrases

<table>
<thead>
<tr>
<th>Filename</th>
<th>Spoken as</th>
</tr>
</thead>
<tbody>
<tr>
<td>ac_changed.ulaw</td>
<td>“The area code for the number you are dialing has changed to...”</td>
</tr>
<tr>
<td>ac_changed.wav</td>
<td></td>
</tr>
<tr>
<td>circuit_busy.ulaw</td>
<td>“We’re sorry. All circuits are busy now.”</td>
</tr>
<tr>
<td>circuit_busy.wav</td>
<td></td>
</tr>
<tr>
<td>contact_provider.ulaw</td>
<td>“Please contact your service provider.”</td>
</tr>
<tr>
<td>contact_provider.wav</td>
<td></td>
</tr>
<tr>
<td>dial_again.ulaw</td>
<td>“If you’d like to make a call, please hang up and dial again.”</td>
</tr>
<tr>
<td>dial_again.wav</td>
<td></td>
</tr>
<tr>
<td>dial_operator.ulaw</td>
<td>“If you need help, please hang up and dial your operator.”</td>
</tr>
<tr>
<td>dial_operator.wav</td>
<td></td>
</tr>
<tr>
<td>disconnected.ulaw</td>
<td>“…has been disconnected.”</td>
</tr>
<tr>
<td>disconnected.wav</td>
<td></td>
</tr>
<tr>
<td>make_note.ulaw</td>
<td>“Please make a note of it.”</td>
</tr>
<tr>
<td>make_note.wav</td>
<td></td>
</tr>
<tr>
<td>new_number.ulaw</td>
<td>“The new number is...”</td>
</tr>
<tr>
<td>new_number.wav</td>
<td></td>
</tr>
<tr>
<td>no_permission.ulaw</td>
<td>“You no longer have permission to utilize the system.”</td>
</tr>
<tr>
<td>no_permission.wav</td>
<td></td>
</tr>
<tr>
<td>num_changed.ulaw</td>
<td>“The number you are calling has changed.”</td>
</tr>
<tr>
<td>num_changed.wav</td>
<td></td>
</tr>
<tr>
<td>num_dialed.ulaw</td>
<td>“The number you have dialed...”</td>
</tr>
<tr>
<td>num_dialed.wav</td>
<td></td>
</tr>
<tr>
<td>num_invalid.ulaw</td>
<td>“…is not a valid number.”</td>
</tr>
<tr>
<td>num_invalid.wav</td>
<td></td>
</tr>
<tr>
<td>please_check.ulaw</td>
<td>“Please check the number and dial again.”</td>
</tr>
<tr>
<td>please_check.wav</td>
<td></td>
</tr>
<tr>
<td>service_outage.ulaw</td>
<td>“Please contact your service provider for information relating to this service outage.”</td>
</tr>
<tr>
<td>service_outage.wav</td>
<td></td>
</tr>
</tbody>
</table>
Table 63. Generic Prompt Phrases (continued)

<table>
<thead>
<tr>
<th>Filename</th>
<th>Spoken as (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>try_again.ulaw</td>
<td>“Please hang up and try your call again later.”</td>
</tr>
<tr>
<td>try_again.wav</td>
<td></td>
</tr>
</tbody>
</table>
This appendix explains the basics of VoiceXML (VXML) Version 1.0, lists the supported VoiceXML elements and attributes, and explains ECMAScript language functionality.

**Note:** For information about VoiceXML 2.0, see “VoiceXML Version 2.0 and Dialog Service” (page 176).

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 Interpreter
- VoiceXML Launcher
- Dialog Service Indicator and Request URI
- VoiceXML Concepts
- VoiceXML Application and Its Documents
- Dialogs
- Grammar and Scripting
- Session Variables
- File Storage and Retrieval
- Media Content Recovery Extension
- VoiceXML Elements Reference
- VoiceXML Attributes Reference
- 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 IP 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 IP Media Server and allows the 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 “VoiceXML Version 2.0 and Dialog Service” (page 176).

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 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://
    appl.carrier.com/cgi/bin/
    genvxml.pl%3Fsubscriberid%3D34590087
```

The SIP URI must be escaped, as described in “Syntax and Escaping” (page 78).

**VoiceXML Launcher**

If a SIP INVITE to the IP Media Server is directed to the dialog service but does not specify a file (voicexml=), then the 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_name attribute_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 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 20 shows how a root document is shared in a VoiceXML application.

![Diagram of root document sharing in VoiceXML application](image)

**Figure 20. 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 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 IP Media Server supports DTMF grammars and also supports ECMA tags that involve the scripting capabilities of the VoiceXML interpreter. See “ECMAScript Functionality” (page 365).
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 368) 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 IP Media Server. For a more detailed description of VoiceXML attributes, see “VoiceXML Attributes Reference” (page 353).

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the variable being assigned to a value. Required.</td>
</tr>
<tr>
<td>expr</td>
<td>ECMAScript expression that, when evaluated, is assigned as the new value of the variable. It must be a valid ECMA value. Required.</td>
</tr>
</tbody>
</table>

<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.

<table>
<thead>
<tr>
<th>Extension</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>ulaw</td>
<td>G.711 µ-law</td>
</tr>
<tr>
<td>alaw</td>
<td>G.711 A law</td>
</tr>
<tr>
<td>msgsm</td>
<td>Microsoft® GSM</td>
</tr>
</tbody>
</table>

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:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>String literal containing the URI for the audio. Required.</td>
</tr>
</tbody>
</table>

**<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:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>expr</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the form-item variable used to track whether this block can be executed. Defaults to an inaccessible internal variable.</td>
</tr>
</tbody>
</table>

**<break>**

Adds a pause to the audio content.

The value of this tag must be empty.

**Parents:** `<audio>`, `<choice>`, `<prompt>`

**Attributes:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>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.</td>
</tr>
<tr>
<td>msecs</td>
<td>Text. The number of milliseconds to pause.</td>
</tr>
</tbody>
</table>

**<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:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Event or events to catch. <strong>Required.</strong></td>
</tr>
<tr>
<td>count</td>
<td>The occurrence of the event. (Default: 1.) The count allows you to handle different occurrences of the same event differently. Each form item and <code>&lt;menu&gt;</code> maintains a counter for each event that occurs while it is being visited; these counters are reset each time the <code>&lt;menu&gt;</code> or form item’s <code>&lt;form&gt;</code> is re-entered.</td>
</tr>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. (Default: TRUE). When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
</tbody>
</table>

**<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 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:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtmf</td>
<td>DTMF sequence for the choice.</td>
</tr>
<tr>
<td>event</td>
<td>An event to be thrown instead of going to ext.</td>
</tr>
<tr>
<td>expr</td>
<td>Expression to evaluate instead of going to next.</td>
</tr>
<tr>
<td>next</td>
<td>URI of next dialog or document.</td>
</tr>
</tbody>
</table>

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:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namelist</td>
<td>List of items to clear. (When not specified, the application clears all form items in the current form.) <strong>Required.</strong></td>
</tr>
</tbody>
</table>

**<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>namelist</td>
<td>See above.</td>
</tr>
</tbody>
</table>

**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:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>URI for the source for an external grammar.</td>
</tr>
<tr>
<td>scope</td>
<td>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.</td>
</tr>
<tr>
<td>type</td>
<td>x-dtmf or regex</td>
</tr>
</tbody>
</table>

**<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:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
</tbody>
</table>

<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:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>count</td>
<td>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 &lt;menu&gt; maintains a counter for each event that occurs while it is being visited; these counters are reset each time the &lt;menu&gt; or form item's &lt;form&gt; is re-entered.</td>
</tr>
</tbody>
</table>

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>expr</td>
<td>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.</td>
</tr>
<tr>
<td>modal</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>Field-item variable in the dialog scope that holds the result.</td>
</tr>
<tr>
<td>slot</td>
<td>Name of the grammar slot used to populate the variable. If it is absent, it defaults to the variable name.</td>
</tr>
<tr>
<td>type</td>
<td>Type of field, for example digits, currency, phone, date, Boolean, number, time. If not present, &lt;grammar&gt; and/or &lt;dtmf&gt; elements can be specified instead.</td>
</tr>
</tbody>
</table>

<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>`.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mode</td>
<td>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.</td>
</tr>
<tr>
<td>namelist</td>
<td>The names of the form’s field items.</td>
</tr>
</tbody>
</table>

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Name of the form.</td>
</tr>
<tr>
<td>scope</td>
<td>Default scope of the form’s grammars.</td>
</tr>
</tbody>
</table>

/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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>ECMAScript expression that yields the target URI.</td>
</tr>
<tr>
<td>expritem</td>
<td>ECMAScript expression that yields the name of the target form item.</td>
</tr>
<tr>
<td>next</td>
<td>URL to which to transition.</td>
</tr>
<tr>
<td>nextitem</td>
<td>Name of the next form item in the current form.</td>
</tr>
</tbody>
</table>
<grammar>
This element is identical to <dtmf>, because the IP Media Server supports only DTMF grammars.

<help>
Receives a help event, and is shorthand for <catch event="help">. On the 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>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 &lt;menu&gt; maintains a counter for each event that occurs while it is being visited; these counters are reset each time the &lt;menu&gt; or form item's &lt;form&gt; is re-entered.</td>
</tr>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
</tbody>
</table>

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
</tbody>
</table>

**<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>expr</td>
<td>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.</td>
</tr>
<tr>
<td>name</td>
<td>The name of a form-item variable used to track whether the <code>&lt;initial&gt;</code> is eligible to execute; defaults to an inaccessible internal variable.</td>
</tr>
</tbody>
</table>

**<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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>next</td>
<td>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).</td>
</tr>
<tr>
<td>expr</td>
<td>Like next, except that the URI is dynamically determined by evaluating the given ECMAScript expression.</td>
</tr>
<tr>
<td>event</td>
<td>The event to generate when the user matches one of the link grammars.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>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 &lt;log&gt; 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 &lt;log&gt; tag with the trap extension: &lt;log dest=&quot;snmp&quot;&gt;Unable to contact VoiceXML application server.&lt;/log&gt;</td>
</tr>
</tbody>
</table>

**<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:  

Child Elements:  

Attributes:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Identifier of the menu. It allows the menu to be the target of a &lt;goto&gt; or a &lt;submit&gt;.</td>
</tr>
<tr>
<td>scope</td>
<td>Menu's grammar scope. Either dialog (the default) or document.</td>
</tr>
<tr>
<td>dtmf</td>
<td>TRUE or FALSE. When TRUE, any choices that do not have explicit DTMF elements are given implicit ones: 1, 2, etc.</td>
</tr>
</tbody>
</table>
<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the meta-data property.</td>
</tr>
<tr>
<td>http-equiv</td>
<td>Name of an HTTP response header. Either name or http-equiv must be specified, but not both.</td>
</tr>
<tr>
<td>content</td>
<td>Value of the meta-data property. Required.</td>
</tr>
</tbody>
</table>

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>count</td>
<td>A number that allows you to issue different prompts if the user is doing something repeatedly. If omitted, it defaults to 1.</td>
</tr>
</tbody>
</table>

<nomatch>
Used to catch a nomatch event. Shorthand for <catch event="nomatch">. See “<catch>” (page 333).

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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>count</td>
<td>A number that allows you to issue different prompts if the user is doing something repeatedly. If omitted, it defaults to 1.</td>
</tr>
</tbody>
</table>

<param>

Specifies the values that are passed to subdialogs.

The value of this tag must be empty.

Parent:  <subdialog>

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name to be associated with this parameter when the subdialog is invoked. Required.</td>
</tr>
<tr>
<td>expr</td>
<td>Expression that computes the value associated with name.</td>
</tr>
<tr>
<td>value</td>
<td>Associates a literal string value with name.</td>
</tr>
<tr>
<td>valuetype</td>
<td>Whether the value associated with name is data or a URI (ref).</td>
</tr>
<tr>
<td>type</td>
<td>MIME type of the result provided by a URI if the value type is ref; only relevant for uses of &lt;param&gt; in &lt;object&gt;.</td>
</tr>
</tbody>
</table>

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bargein</td>
<td>Whether a prompt may be interrupted. (Default: TRUE.)</td>
</tr>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
</tbody>
</table>
<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Property name. <strong>Required.</strong></td>
</tr>
<tr>
<td>value</td>
<td>Property value. <strong>Required.</strong></td>
</tr>
</tbody>
</table>

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beep</td>
<td>TRUE or FALSE (the default). If TRUE, the application plays a tone just prior to recording.</td>
</tr>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>dtmfterm</td>
<td>TRUE or FALSE. If TRUE (default), a DTMF keypress terminates the recording.</td>
</tr>
</tbody>
</table>
<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Returns, then sends this event.</td>
</tr>
<tr>
<td>namelist</td>
<td>Variable names to be returned to the calling dialog.</td>
</tr>
</tbody>
</table>

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>URL for the resource.</td>
</tr>
</tbody>
</table>

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>expr</td>
<td>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.</td>
</tr>
<tr>
<td>method</td>
<td>Request method (get or post).</td>
</tr>
<tr>
<td>modal</td>
<td>Controls which grammars are active during the subdialog.</td>
</tr>
</tbody>
</table>
VoiceXML Elements Reference

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.

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Like next, except that the URI is dynamically determined by evaluating the given ECMAScript expression. One of next, expr, or fetchtimeout is required.</td>
</tr>
<tr>
<td>method</td>
<td>The request method (GET or POST). The default is GET.</td>
</tr>
<tr>
<td>namelist</td>
<td>The list of variables to submit.</td>
</tr>
<tr>
<td>next</td>
<td>The URL to which the query is submitted.</td>
</tr>
<tr>
<td>fetchtimeout</td>
<td>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.</td>
</tr>
</tbody>
</table>

<throw>

Sends an event to be received by <catch>.

The value of this tag must be empty.

Attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>Returns, then sends this event.</td>
</tr>
</tbody>
</table>
<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>The outcome of the transfer attempt. For possible return values, see the following table.</td>
</tr>
<tr>
<td>expr</td>
<td>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.</td>
</tr>
<tr>
<td>cond</td>
<td>Boolean value of TRUE or FALSE. Defaults to TRUE. When FALSE, the tag is ignored or skipped; the form is not visited.</td>
</tr>
<tr>
<td>dest</td>
<td>The URL of the destination (phone, IP telephone address).</td>
</tr>
<tr>
<td>destexpr</td>
<td>An ECMAScript expression yielding the URI of the destination.</td>
</tr>
<tr>
<td>bridge</td>
<td>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.</td>
</tr>
<tr>
<td>connect</td>
<td>Time to wait while trying to connect the timeout call before returning the noanswer condition. Default is platform specific.</td>
</tr>
<tr>
<td>maxtime</td>
<td>Time that the call is allowed to last, or 0 if it can last arbitrarily long. Only applies if bridge is TRUE. (Default: 0.)</td>
</tr>
<tr>
<td>transfer</td>
<td>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.</td>
</tr>
<tr>
<td>reqUri</td>
<td>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.</td>
</tr>
<tr>
<td>requiri expr</td>
<td>An ECMAScript expression that evaluates to a SIP URI.</td>
</tr>
<tr>
<td>src</td>
<td>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)</td>
</tr>
<tr>
<td>srcexpr</td>
<td>An ECMAScript expression yielding src. (Dialogic-specific attribute).</td>
</tr>
<tr>
<td>stopdigits</td>
<td>Defines the list of single DTMF digits that ends the transfer.</td>
</tr>
<tr>
<td>longdigit</td>
<td>Enables detection of long DTMF digits or multiple, short instance of the same digit.</td>
</tr>
<tr>
<td>video</td>
<td>Whether video is available on the outbound call. The value can be one of suppress or allow. See &quot;RTP Codec Selection Using the &lt;transfer&gt; Element&quot; (page 226).</td>
</tr>
</tbody>
</table>
**Note:** The src, srcexpr, stopdigits, longdigit, and video attributes are Dialogic-specific attributes.

Return values of Name variable:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>The endpoint refused the call.</td>
</tr>
<tr>
<td>noanswer</td>
<td>There was no answer within the specified time.</td>
</tr>
<tr>
<td>network busy</td>
<td>Some intermediate network refused the call.</td>
</tr>
<tr>
<td>near_end_disconnect</td>
<td>The call completed and was terminated by the caller.</td>
</tr>
<tr>
<td>far_end_disconnect</td>
<td>The call completed and was terminated by the callee.</td>
</tr>
<tr>
<td>network_disconnect</td>
<td>The call completed and was terminated by the network.</td>
</tr>
</tbody>
</table>

**<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:**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>Expression to render.</td>
</tr>
<tr>
<td></td>
<td><strong>Required.</strong></td>
</tr>
<tr>
<td>class</td>
<td>Data type of the value.</td>
</tr>
<tr>
<td>recsrc</td>
<td>URI used to determine text to prompt conversion.</td>
</tr>
<tr>
<td></td>
<td>Defaults to the value of com.snowshore.recsrc if</td>
</tr>
<tr>
<td></td>
<td>specified, otherwise it is</td>
</tr>
<tr>
<td></td>
<td>&quot;<a href="http://localhost/snowshore/phrase.cgi">http://localhost/snowshore/phrase.cgi</a>?&quot;.</td>
</tr>
</tbody>
</table>

**<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name of the variable. <strong>Required.</strong></td>
</tr>
<tr>
<td>expr</td>
<td>An initial value.</td>
</tr>
</tbody>
</table>

<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Version of VoiceXML. <strong>Required.</strong></td>
</tr>
<tr>
<td>base</td>
<td>Base URL. The root document URL, if any.</td>
</tr>
</tbody>
</table>
VoiceXML Attributes Reference

This section describes the VoiceXML attributes supported by the 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Allow input to interrupt.</td>
</tr>
<tr>
<td>(default)</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>Do not allow input to interrupt.</td>
</tr>
</tbody>
</table>

bargeintype
Type of interrupt (bargein) the IP Media Server performs in response to DTMF inputs.
Attribute for:  <prompt>
Values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The default value of bargein is true.</td>
</tr>
<tr>
<td>false</td>
<td>The default value of bargein is false.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Plays a tone when recording starts.</td>
</tr>
<tr>
<td>false</td>
<td>Does not play tone when recording starts.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Suspends document interpretation until the transferred call terminates, at which time document processing resumes.</td>
</tr>
<tr>
<td>false (default)</td>
<td>Suspends document interpretation but terminates immediately after the transferred call ends.</td>
</tr>
</tbody>
</table>

**class**

The data type of the value.

**Attribute for:** `<sayas>` `<value>`

Values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>phone</td>
<td>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.</td>
</tr>
<tr>
<td>digits</td>
<td>Each digit is spoken with no pauses.</td>
</tr>
<tr>
<td>number</td>
<td>The value is spoken as a positive cardinal number. Negative numbers are not supported and will produce odd output.</td>
</tr>
</tbody>
</table>
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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The form item is executed.</td>
</tr>
<tr>
<td>(default)</td>
<td></td>
</tr>
<tr>
<td>false</td>
<td>The form item is ignored or skipped; the form is not visited.</td>
</tr>
</tbody>
</table>

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>`

- choiceText /CDATA (touch-tone digit for this choice).
- menuDialog for making a selection among choices. See “<menu>” (page 342).

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Assigns touch-tone digits.</td>
</tr>
<tr>
<td>false</td>
<td>Does not assign touch-tone digits.</td>
</tr>
</tbody>
</table>

**dtmfterm**
Whether to allow touch-tone interruption.

**Attribute for:** `<record>`

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true (default)</td>
<td>Touch-tone terminates recording.</td>
</tr>
<tr>
<td>false</td>
<td>Touch-tone does not terminate recording.</td>
</tr>
</tbody>
</table>

**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).
expitem
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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>Turns DTMF detection on for long digits.</td>
</tr>
<tr>
<td>no</td>
<td>Turns DTMF detection off for long digits.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get</td>
<td>HTTP GET method.</td>
</tr>
<tr>
<td>post</td>
<td>HTTP POST method.</td>
</tr>
</tbody>
</table>

**modal**

Whether to enable outside grammars.

**Attribute for:** `<field>`, `<subdialog>`

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Disable higher-level grammars in the scope.</td>
</tr>
<tr>
<td>false</td>
<td>Enable higher-level grammars in the scope.</td>
</tr>
</tbody>
</table>

**mode**

Specifies mode for performing action when fields are filled in a form.

**Attribute for:** `<filled>`

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Action executes when all fields are filled.</td>
</tr>
<tr>
<td>any</td>
<td>Action is executed when any field is filled by user input.</td>
</tr>
</tbody>
</table>

**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.
**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.

**Attributes for:** `<choice>`, `<goto>`, `<link>`

**nextitem**

First item to visit in the next dialog.

**Attributes for:** `<goto>`

**Value:** Text (NMTOKEN)

**recsrc**

Method to use when concatenating audio.

**Attributes 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.

**Attributes for:** `<form>`, `<grammar>`, `<menu>`

**Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialog</td>
<td>Scope is the current dialog.</td>
</tr>
<tr>
<td>document</td>
<td>Scope is the current document.</td>
</tr>
</tbody>
</table>

**size**

Relative pause duration.

**Attributes for:** `<break>`
Values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>0 ms</td>
</tr>
<tr>
<td>small</td>
<td>50 ms</td>
</tr>
<tr>
<td>medium</td>
<td>200 ms</td>
</tr>
<tr>
<td>(default)</td>
<td></td>
</tr>
<tr>
<td>large</td>
<td>500 ms</td>
</tr>
</tbody>
</table>

**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>, <say-as>

Values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtmf</td>
<td>x-dtmf or regex.</td>
</tr>
<tr>
<td>field</td>
<td>A built-in type (Boolean, digits, phone, date, currency, number, time).</td>
</tr>
<tr>
<td>grammar</td>
<td>Content type of the grammar.</td>
</tr>
<tr>
<td>object</td>
<td>Content type of the object data.</td>
</tr>
<tr>
<td>param</td>
<td>MIME type of the result provided by a URI if the value type is ref; only relevant for uses of &lt;param&gt; in &lt;object&gt;.</td>
</tr>
<tr>
<td>record</td>
<td>MIME type of the recording. See &quot;MIME Recording Encoding Types&quot; (page 363).</td>
</tr>
<tr>
<td>sayas</td>
<td>Data type to use in interpreting the value for the following: telephone, date, digits, number.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Value is data.</td>
</tr>
<tr>
<td>ref</td>
<td>Value is a URL pointing to data.</td>
</tr>
</tbody>
</table>
version
Version of VoiceXML. Required.
Attribute for: <vxml>
Value: 1.0.

VCR
Whether VCR controls are active
Attribute for: <prompt>
Values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>VCR controls are enabled.</td>
</tr>
</tbody>
</table>

video
Whether video media is offered on outgoing calls.
Attribute for: <transfer>
Values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>Video media is offered on outgoing calls.</td>
</tr>
<tr>
<td>suppress</td>
<td>Video media is not offered on outgoing calls.</td>
</tr>
</tbody>
</table>

See “RTP Codec Selection Using the <transfer> Element” (page 226) 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:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>The default critical digit timer is set to the same value as the interdigit timer (thus it has no effect).</td>
</tr>
<tr>
<td>-1</td>
<td>Immediate. Use “shortest match first”.</td>
</tr>
<tr>
<td>0</td>
<td>Infinite. Use &quot;longest match first&quot;, but waits infinitely once a match is found. Thus, matched input must end with a return digit or restart digit.</td>
</tr>
<tr>
<td>&gt;0</td>
<td>Uses “longest match first”, and waits that amount of time when a match is found.</td>
</tr>
</tbody>
</table>

**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 64 lists the MIME type, file format, and audio encoding.

**Table 64. Supported MIME Types, Formats, and Audio Encodings**

<table>
<thead>
<tr>
<th>MIME Type</th>
<th>File Format</th>
<th>Audio Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio / basic</td>
<td>raw</td>
<td>G.711 µ-law</td>
</tr>
<tr>
<td>audio/ x-alaw-basic</td>
<td>raw</td>
<td>G.711 A law</td>
</tr>
</tbody>
</table>

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 65 lists the accepted WAVE format specifiers.
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 66 provides a summary of the supported codec parameters and values and their corresponding audio encodings.

### Table 66. Codec Parameter Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Audio Encoding</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>G.711 µ-law</td>
<td>IANA standard</td>
</tr>
<tr>
<td>6</td>
<td>G.711 A law</td>
<td>IANA standard</td>
</tr>
<tr>
<td>31</td>
<td>Microsoft® GSM</td>
<td>IANA standard</td>
</tr>
<tr>
<td>ulaw</td>
<td>G.711 µ-law</td>
<td></td>
</tr>
<tr>
<td>alaw</td>
<td>G.711 A law</td>
<td></td>
</tr>
<tr>
<td>msgsm</td>
<td>Microsoft® GSM</td>
<td></td>
</tr>
</tbody>
</table>

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 67 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>
<thead>
<tr>
<th>Category</th>
<th>Feature/Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Handling</td>
<td>Array join, length, reverse, sort</td>
</tr>
<tr>
<td>Assignments</td>
<td>Assign (=) Compound Assign (OP=)</td>
</tr>
<tr>
<td>Booleans</td>
<td>Boolean</td>
</tr>
<tr>
<td>Comments</td>
<td>/<em>...</em>/ or //</td>
</tr>
<tr>
<td>Constants/Literals</td>
<td>NaN null true, false Infinity undefined</td>
</tr>
<tr>
<td>Control flow</td>
<td>Break continue for for...in if...else return while</td>
</tr>
<tr>
<td>Dates and Time</td>
<td>Date getDate, getDay, getFullYear, getHours, getMilliseconds, getMinutes, getMonth, getSeconds, getTime, getTimezoneOffset, getYear, getUTCDate, getUTCDay, getUTCFullYear, getUTCHours, getUTCMilliseconds, getUTCMinutes, getUTCMonth, getUTCSecs, setDate, setFullYear, setHours, setMilliseconds, setMinutes, setMonth, setSeconds, setTime, setYear, setUTCDate, setUTCFullYear, setUTCHours, setUTCMilliseconds, setUTCMinutes, setUTCMonth, setUTCSecs, toGMTString, toLocaleString, toUTCString, parse, UTC</td>
</tr>
<tr>
<td>Declarations</td>
<td>Function new this var with</td>
</tr>
</tbody>
</table>
### Table 67. ECMAScript Functionality (continued)

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature/Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function Creation</td>
<td>Function arguments, length</td>
</tr>
<tr>
<td>Global Methods</td>
<td>Global escape, unescape eval isFinite, isNaN parseInt, parseFloat</td>
</tr>
<tr>
<td>Math</td>
<td>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</td>
</tr>
<tr>
<td>Numbers</td>
<td>Number MAX_VALUE, MIN_VALUE NaN NEGATIVE_INFINITY, POSITIVE_INFINITY</td>
</tr>
<tr>
<td>Object Creation</td>
<td>Object new constructor, prototype, instanceof, toString, valueOf</td>
</tr>
<tr>
<td>Operators</td>
<td>Addition (+), Subtraction (-) Modulus arithmetic (%) Multiplication (*), Division (/) Negation (-) Equality (==), Inequality (!=) Less Than (&lt;), Less Than or Equal To (&lt;=) Greater Than (&gt;) Greater Than or Equal To (&gt;=) Logical And(&amp;&amp;), Or (</td>
</tr>
<tr>
<td>Objects</td>
<td>Array Boolean Date Function Global Math Number Object String</td>
</tr>
</tbody>
</table>
Table 67. ECMAScript Functionality (continued)

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature/Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strings</td>
<td>String</td>
</tr>
<tr>
<td></td>
<td>charAt, charCodeAt, fromCharCode</td>
</tr>
<tr>
<td></td>
<td>indexOf, lastIndexOf</td>
</tr>
<tr>
<td></td>
<td>split</td>
</tr>
<tr>
<td></td>
<td>toLowerCase, toUpperCase</td>
</tr>
<tr>
<td></td>
<td>length</td>
</tr>
</tbody>
</table>
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.
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%3d1;dogs=nice;user=phone
SIP/2.0
From: <sip:968037431@10.102.4.45>;tag=2d04660a-13c4-40ed433f-9ec19fd-1786
To: <sip:961234567@10.102.4.9>
Call-ID: 6a7b774-2d04660a-13c4-40ed433f-9ec19fd-706101.102.4.45
CSeq: 1 INVITE
Via: SIP/2.0/UDP 10.102.4.134:5060;branch=z9hG4bKi8MKi8i!yi8MK2SsMu8Uyake2q00UqMi8iy.1-ld9515c
Via: SIP/2.0/UDP 10.102.4.45:5060;received=10.102.4.45;branch=z9hG4bK-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://10.102.4.9:9020/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=961234567&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://10.102.4.9:9020/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
Example 49. MSCML Schema

```xml
<?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"/>
            </xs:choice>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
type="conferenceNotificationType"/>
<xs:element name="keypress"
type="keypressNotificationType"/>
<xs:element name="signal"
type="signalNotificationType"/>
</xs:choice>
</xs:complexType>
</xs:element>

</xs:choice>
</xs:complexType>
</xs:element>

</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:extension>
</xs:complexContent>
</xs:complexType>
<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" use="required"/>
</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:extension>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="managecontentRequestType">
  <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: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:complexType>
<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" maxOccurs="unbounded"/>
      </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:sequence>
</xs:complexType>
<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>
<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:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</xs:complexType>
<xs:enumeration value="ndn"/>
<xs:enumeration value="crd"/>
<xs:enumeration value="ord"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
<xs:complexType name="yesnoType">
<xs:restriction base="xs:NMTOKEN">
<xs:enumeration value="yes"/>
<xs:enumeration value="no"/>
<xs:enumeration value="true"/>
<xs:enumeration value="false"/>
</xs:restriction>
</xs:simpleType>
</xs:complexType>
<xs:complexType 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:complexType>
</xs:schema>
D - Dial Pulse Detection

This appendix describes the Dial Pulse Detection feature and includes the following sections:

◆ Overview
◆ Configuration
Overview

In addition to the existing checks for Dual Tone Multi Frequency (DTMF) and Call Progress Analysis (CPA) tones, the IP Media Server can be configured to detect Dial Pulses.

Dial Pulse Detection 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 383.)
- The IP Media Server detects Dial Pulse and DTMF within a single stream. As a result, when the IP Media Server detects each type of signaling, it checks for the completion of any previously detected digits of the alternate type.

Consistency

In order for the IP Media Server 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.

```xml
<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 IP Media Server 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>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default (in Decimal)</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP_enabled</td>
<td>DP Enabled. Determines if the Dial Pulse detection is enabled or disabled system-wide</td>
<td>N</td>
<td>Y or N</td>
</tr>
<tr>
<td>DP_detectAfterDtmf</td>
<td>DP Detect After Dtmf. Determines if Dial Pulses should be detected on a call leg after DTMF has been detected on that leg.</td>
<td>N</td>
<td>Y or N</td>
</tr>
<tr>
<td>DP_low_Thresh</td>
<td>Low Threshold Level. Sample level required to be below in order to declare sample as a Low Signal</td>
<td>5,000</td>
<td>0 to 32768</td>
</tr>
<tr>
<td>DP_lowThreshCnt</td>
<td>Low Threshold Count. Number of consecutive Low samples to be considered a post Dial Pulse Low (or trough)</td>
<td>17</td>
<td>0 to 32768</td>
</tr>
<tr>
<td>DP_peakThresh</td>
<td>Peak Threshold. Sample value required for signal to be declared a dial pulse peak</td>
<td>14,000</td>
<td>20 to 32768</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Default (in Decimal)</td>
<td>Range</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DP_maxTransCnt</td>
<td>Max Transition Count 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</td>
<td>30,000</td>
<td>0 to 32768</td>
</tr>
<tr>
<td>DP_settleByCnt</td>
<td>Settle By Count Number of consecutive samples that the signal must be below the threshold to declare the signal as a dial pulse.</td>
<td>400</td>
<td>0 to 65535</td>
</tr>
<tr>
<td>DG_InterDigitQuietCnt</td>
<td>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.</td>
<td>30</td>
<td>20 to 32768</td>
</tr>
</tbody>
</table>
The following figure illustrates these parameters and the description follows.

*Figure 21. Dial Pulse Detection Parameters*

**Description of Dial Pulse Detection 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, the IP Media Server 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.
Index

Symbols
# key 149
* key 149
= 178, 328
? 178, 328

Numerics
100rel 77
2833 events 102
3GPP 32, 33, 218, 219
3GPP2 32, 33, 218, 219

A
ACK method (SIP) 54
active talker events 29, 106, 112, 115
activetalkers (MSCML) 136
advanced conferences
   active talker events 115
   audio 121
   control leg 111, 116
   creating 111
   defaults 107
   DTMF 119
   joining participants 113, 249
   modifying conference 112, 248
   modifying participants 114
   MSCML 125
   parking participants 116, 253
   playing to 111, 116
   recording 111, 116
   removing participants 114
Advanced conferencing attributes
   reserveconf media 107
   reserved talkers 107
anncc 27, 82, 89, 91
announcement parameters 91
delay 95
duration 96
early 95
play 35, 94
announcement sequences 36, 89
announcement service indicator 27
   See also anncc
announcement variables
digits 96
money 97
months 97
numbers 97
silence 98
strings 98
time 98
weekdays 98
announcements 35, 89
default locations 36
in conferences 140
Request-URI 90
sequences 89, 92
simple 35, 89
storage 35
types of 89
application (VoiceXML) 353
application servers 26, 47
assign element (VoiceXML) 183, 332
attribute lines (SDP) 75
Attributes
   endsilence 152
   initsilence 152
   playrecord 151, 152
   stop_on_error 151
attributes
   stop_on_error 153
audio 32
   DTMF digits 34, 138
duration of play 96
encoding 168
file retrieval 89
in conferences 116, 121, 139
mixing 139
playing 91
recording 151, 161
supported CODECs 74
voice encoding 32
audio element (VoiceXML) 183, 332
audio files
cardinal numbers 316
Chinese Mandarin 313
dates 317
default location 36, 89
in Sound Library 35
letters of the alphabet 318
ordinal numbers 317
press key phrases 320
standard phrases 313
storage 35

B
barge (MSCML) 148, 151, 163
bargein (VoiceXML) 353
bargeintype (VoiceXML) 353
base (VoiceXML) 353
baserurl (MSCML) 164
beep (VoiceXML) 353
block element (VoiceXML) 184, 333
break element (VoiceXML) 333
bridge (VoiceXML) 354

C
cached fonts 229
call control 26
Call-ID header (SIP) 62
channels 36
choice element (VoiceXML) 185, 189, 334
class (VoiceXML) 354
clear element (VoiceXML) 186, 334
cleardigits (MSCML) 164
CLI
configuration for conferences 99
code (MSCML) 174
CODECs 71, 74
cond (VoiceXML) 355
cnf= 27, 99
conference control leg 111, 116
conference participants
advanced conferences 113
audio mixing 139
joining to advanced conference 113
joining to simple conference 101
modifying 114
playing audio to 121
removing 114
simple conferences 102
conference service indicator 99
conferences 36
advanced 106
audio 121
control leg 106
creating advanced 111
creating simple 101
defaults for advanced 107
defaults for simple 102
DTMF digits 119, 138
ending simple 101
examples 101
joining participants 101, 113
mixing 36
modifying advanced 112, 114
parking participants 116
playing announcements in 140
recording 116, 161
removing participants 114
simple 37, 101
conferencing attributes
activetalkers (MSCML) 136
configure_team (MSCML) 136
dtmfclamp (MSCML) 137, 138
events (MSCML) 137
id (MSCML) 138
keypress (MSCML) 137
mixmode 139
mixmode (MSCML) 139
notification (MSCML) 137
report (MSCML) 140
reserveconfmedia 140
reserveconfmedia (MSCML) 140
reservedtalkers 141
signal 137
subscribe 137
subscribe (MSCML) 137
toneclamp (MSCML) 141
type 141
type (MSCML) 141
configure_conference 106
configure_conference (MSCML) 106, 111, 125
configure_leg 106, 130
configure_leg (MSCML) 106, 113, 127, 128
configure_team 130
configure_team (MSCML) 136
conformance
HTTP 33
NFS 33
connection information (SDP) 74
Contact header (SIP) 62
content attribute (VoiceXML) 355
content element (VoiceXML) 187
content types (SIP) 47
Content-Length header (SIP) 62
Content-Type header (SIP) 63
control agent 26
control leg (advanced conferences) 106, 111, 116
count attribute (VoiceXML) 355
creating
advanced conferences 111
simple conferences 101
CSeq header (SIP) 63

D
data element (VoiceXML) 188, 335
default application 27
default location
  simple announcements 90
  SnowShore prompts 89
delay (announcement parameter) 95
delay (MSCML) 164
dest attribute (VoiceXML) 355
dialog 27, 29
dialog service indicator 177, 327
Digit Buffering 148
digit buffering (MSCML) 148
digit collection (MSCML) 148
digits 173
  barging (MSCML) 148, 151
  buffering 148
  collection 119, 161, 166
  ending collection 170
  flushing 164
  maximum collected 168
digits (MSCML) 173
digits (variables) 96
direction (SDP) 75
disconnect element (VoiceXML) 189, 336
double keypress 120
DTMF 34, 138, 148, 161
  barging 163
  detecting in conferences 119
  escape key 165
  flushing digits 164
  return key 170
  VoiceXML grammar 180, 329
DTMF attribute (VoiceXML) 356
dtmf element (VoiceXML) 336
dtmfclamp (MSCML) 108, 113, 128, 136, 137, 138
dtmfterm attribute (VoiceXML) 356
duration 164
duration (MSCML) 164
duration= (announcement parameter) 96
dynamic fonts 229

E
early= (announcement parameter) 69, 95
ECMAScripting 220, 327
else element (VoiceXML) 189, 336
elseif element (VoiceXML) 189, 337
encoding (MSCML) 165
ending

F
field element (VoiceXML) 190, 338
filled element (VoiceXML) 191, 338
finalsilence (VoiceXML) 357
firstdigittimer (MSCML) 166
font element (VoiceXML) 192
font files 228
form element (VoiceXML) 193, 339
From header (SIP) 63

G
G.711 74
gain control 304
goto element (VoiceXML) 193, 339
grammar element (VoiceXML) 194, 340

H
hairpinning 294
headers
  SDP 71
help element (VoiceXML) 196, 340
Hold Behavior 81
Hold media 74, 81, 145
Hold SDP 82
HTTP 33, 35, 89
http-equiv (VoiceXML) 357
I

id (MSCML) 138, 173
id (VoiceXML) 357
image overlay 41, 227
INFO method (SIP) 58, 144
and MSCML 28
initsilence (MSCML) 166
interdigittimer (MSCML) 166
INVITE method (SIP) 52
and MSCML 28
IVR
digit collection 161
escape key 165
functions 144
return key 170
ivr 27, 28, 82, 144
IVR attributes
barge 163
baseurl 164
beep 164
cleardigits 164
delay 164
duration 164
encoding 165
endsilence 165, 172
escapekey 165
extradigittimer 166
firstdigittimer 166
id 166
initsilence 166, 167, 170
interdigittimer 166
maxdigits 168
mode 168
offset 168
promptencoding 168
recencoding 169
recstopmask 169
recurl 169
repeat 139, 169
returnkey 170
stop_on_error 170
subtype 171
type 171
url 172
IVR responses
code 174
digits 174
error_info 173
id 173
playduration 174
reason 174
reclength 175
text 175
IVR service indicator 143

J

joining participants
advanced conferences 113
simple conferences 101

K

keypress (MSCML) 137

L

locale (MSCML) 167

M

Maskdigits 120
maskdigits 167
maxdigits (MSCML) 168
maxtime (VoiceXML) 357
Media Content Recovery 182, 188, 330, 331, 335
media information (SDP) 74
Media Server Control Markup Language 28
method (VoiceXML) 357
methods (SIP) 52
mid-call requests 58, 144
MIME 49, 106, 218, 219
mixed audio/video 32
mixing (in conferences) 36, 139
mixmode (MSCML) 108, 113, 116, 139
modal (VoiceXML) 358
mode (VoiceXML) 358
modifying advanced conferences 112, 248
money (PDML) 97
months (PCML) 97
MSCML 28, 99, 164, 173
activetalkers 136
advanced conferences 37
barge 163
baseurl 164
cleardigits 164
code 174
conferencing attributes 136, 137, 138
conferencing dtmfclamp 138
conferencing events 137
conferencing id 138
conferencing keypress 137
conferencing mixmode 139
conferencing notification 137
conferencing report 140
conferencing requests 125
conferencing reserveconfmedia 140
conferencing subscribe 137
conferencing toneclamp 141
conferencing type 141
configure_team 136
delay 164
digit collection 148
dtmfclamp 137, 138
encoding 165
endsilence 165
escapekey 165
events 137
extradigittimer 166
firstdigittimer 166
id 138, 173
initsilence 166
interdigittimer 166
keypress 137
maxdigits 168
MIME type 106
mixmode 139
multipart syntax 80
notification 137
prompt 162
promptencoding 168
recencoding 169
reclength 175
recstopmask 169
report 140
reserveconfmedia 140
reservedtalkers 141
responses 29, 147, 149, 154
responses for playrecord 152
sample code 256
signal 137
subscribe 137
subtype 171
syntax 80
text 175
toneclamp 141
type 141, 171
value 172
MSCML directives for IVR functions
play 145
playcollect 145
playrecord 145
stop 145
MSCML locale 167
msecs (VoiceXML) 358
msVXMLCriticalError 199, 342
msVXMLLastCriticalError 199, 342
multimedia files 32

O
overlay 41
overlay element (VoiceXML) 202

P
packet time (SDP) 75
PDF 22
play (MSCML) 116, 121
  sample code 271
Play and/or Collect 148
play= 35
play= (announcement parameter) 35, 94
playcollect (MSCML) 116, 119
  sample code 256, 273
playrecord (MSCML) 116, 117, 151
  sample code 256, 273
Playrecord attributes
  recurl 151
playrecord attributes
  beep 152
  duration 152
  mode 152
  recencoding 152
port 5060 76, 89
prompt (MSCML) 162
promptencoding (MSCML) 168
provisional responses 77
ptime (SDP) 75

Q
question mark 178, 328

R
Real Time Streaming Protocol (RTSP) 34, 42
  VCR controls 304
  VCR controls example 306
recencoding (MSCML) 169
reclength (MSCML) 175
recording 151, 165, 166
  encoding 169
  length of 175
  mode 168
  MSCML request for 161
  terminating 169
  URL for 169
Record-Route header (SIP) 64
recsrc (VoiceXML) 359
recstopmask (MSCML) 169
recvonly (SDP) 75
re-INVITE (SIP) 53

N
name (VoiceXML) 358
namelist (VoiceXML) 358
next (VoiceXML) 359
nextitem (VoiceXML) 359
NFS 35, 89
notification (MSCML) 137
numbers (variables) 97
Index

repeat= (announcement parameter) 95
report (MSCML) 140
report values
  both 120
  long 120
  none 119
  standard 120
Requests (SIP) 50
Request-URI (SIP) 50, 90, 169
  parameters 94
reserveconfmedia (MSCML) 107, 111, 116, 140
reservedtalkers 107, 141
reservedtalkers (MSCML) 111, 141
responses
  for MSCML 173
  SIP 67
Return codes (SIP) 67
returnkey 149
returnkey (MSCML) 170
RFC conformance 45
root document 179, 328
Route header (SIP) 64

S
scope (VoiceXML) 359
scroll element (VoiceXML) 208
SDP 27, 52, 54, 71
  attribute lines 75
  connection information 74
  direction 75
  headers 71
  Hold media 74, 81, 145
  media description headers 73
  media information 74
  optional headers 75
  packet time 75
  session description headers 71
  syntax errors 71
  time description headers 72
  TTL 74
sendonly (SDP) 75
sendrecv (SDP) 75
sequences 36, 92
  See also announcement sequences
service indicators 27, 48
  annc 27
  announcements 27, 89
  conf 99
  conf= 27
  conference 99
  default 27
  dialog 27
  ivr 27, 145
  session 27
Session Description Protocol 71

See also SDP
session timer 53, 64
Session-Expires header (SIP) 64
signal 137
signal (MSCML) 137
silence 165
  and recording 165, 166
  as announcement variables 98
simple announcements 35, 89
  default location 90
  overview 91
Simple conference attributes
dtmfclamp 102
mixmode 102
type 102
simple conferences 37, 101
  creating 101
  default parameters for 102
  ending 101
  joining participants to 101
  upgrading 101
SIP 26
  announcement parameters 94
  default port 76
  event notification 29, 115
  message body 47
  ports 89
  provisional responses 77
  Requests 50
  Request-URI 50, 169
  responses 67
  session description 54
  syntax 47
  transaction 47
SIP headers 60
  Call-ID 62, 115
  compact form 61
  Contact 62
  Content-Length 62
  Content-Type 63
  CSeq 63
  From 63
  Record-Route 64
  Route 64
  Session-Expires 64
  Supported 65
  To 65
  Unsupported 65
  Via 65
SIP methods 52
  ACK 54
  INFO 28, 58, 106, 116, 126, 144
  INVITE 28, 52
SIP Return Codes 67
size (VoiceXML) 359
slot (VoiceXML) 360
src (VoiceXML) 360

Application Developer's Guide 391
Index

Star and Pound Keys 149
stop (MSCML) 154
  sample code 273
stopdigits (VoiceXML) 360
strings (PDML) 98
subdialog element (VoiceXML) 209, 347
subscribe 115
subscribe (MSCML) 115, 126, 137
subscribe element 107
subtype (MSCML) 171
Supported header (SIP) 65
syntax overview
  SDP 71
  SIP 47

T
text (MSCML) 175
text overlay 41, 187, 227
throw element (VoiceXML) 210, 348
time (PDML) 98
timeout (VoiceXML) 360, 361
timing attributes 152
  extradigittimer 149
  firstdigittimer 149
  interdigittimer 149
To header (SIP) 65
toneclamp (MSCML) 108, 113, 128, 141
tones 34
transfer element (VoiceXML) 212, 349
trcr.cfg 192, 229, 231
TTL 74
type (MSCML) 108, 113, 128, 141, 171
type (VoiceXML) 361

U
Unsupported header (SIP) 65
URIs
  recurl attribute 169

V
value (MSCML) 172
value (VoiceXML) 361, 362
value element (VoiceXML) 215, 351
valuetype (VoiceXML) 361
var element (VoiceXML) 215, 351
variable subtypes (PDML) 96
variable types (PDML) 96
VCR (VoiceXML) 362
VCR controls 304
version (VoiceXML) 362
Via header (SIP) 65
video 32
video transcoder 39
  auto setting rules 39
  restrictions 40
  settings 39
video transcoding 39
VoiceXML 29
  cond 355
  elements 183, 332
  grammars supported 180, 329
  sample code 289
VoiceXML attributes
  application 353
  bargein 353
  bargeintype 353
  base 353
  beep 353
  bridge 354
  class 354
  content 355
  count 355
  dest 355
dtmf 356
  dtmfterm 356
event 356
  expr 356
  expritem 357
  finalsilence 357
  http-equiv 357
  id 357
  maxtime 357
  method 357
  modal 358
  mode 358
  msecs 358
  name 358
  namelist 358
  next 359
  nextitem 359
  recsrc 359
  scope 359
  size 359
  slot 360
  src 360
  stopdigits 360
  timeout 360, 361
  type 361
  value 361, 362
  valuetype 361
  VCR 361
  version 362
VoiceXML elements
  assign 183, 332
  audio 183, 332
  block 184, 333
  break 333
  choice 185, 189, 334
  clear 186, 334
  content 187
disconnect 189, 336
dtmf 336
derse 189, 336
derse 189, 337
error 190, 337
exit 190, 337
field 190, 338
filled 191, 338
font 192
form 193, 339
goto 193, 339
grammar 194, 340
help 196, 340
overlay 202
scroll 208
subdialog 209, 347
throw 210, 348
transfer 212, 349
value 215, 351
var 215, 351
vxml 216, 352
VoiceXML interpreter 177, 327
vxml element (VoiceXML) 216, 352

W

weekdays (variables) 98