VoiceXML and CCXML Developer’s Guide

Release 21.0
Document Version 1
### Document Revision History

<table>
<thead>
<tr>
<th>Release</th>
<th>Version</th>
<th>Reason for Change</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.sp3</td>
<td>0.1</td>
<td>Created document.</td>
<td>August, 22, 2007</td>
<td>Stéphane Bastien</td>
</tr>
<tr>
<td>14.sp3</td>
<td>0.2</td>
<td>Updated VoiceXML section.</td>
<td>August 30, 2007</td>
<td>Jean-François Bertrand</td>
</tr>
<tr>
<td>14.sp3</td>
<td>0.3</td>
<td>Updated document.</td>
<td>September 28, 2007</td>
<td>Vadim Ivlev</td>
</tr>
<tr>
<td>14.sp3</td>
<td>1.0</td>
<td>Included comments from review meeting.</td>
<td>October 3, 2007</td>
<td>Stéphane Bastien, Jean-François Bertrand, Vadim Ivlev</td>
</tr>
<tr>
<td>14.sp3</td>
<td>1.0</td>
<td>Added index.</td>
<td>October 4, 2007</td>
<td>Roberta Boyle</td>
</tr>
<tr>
<td>14.sp3</td>
<td>1.0</td>
<td>Edited and published document.</td>
<td>October 5, 2007</td>
<td>Patricia Renaud</td>
</tr>
<tr>
<td>15.0</td>
<td>0.1</td>
<td>Added <code>&lt;createcall&gt;</code> timeout, <code>&lt;record&gt;</code> type, <code>&lt;dialogstart&gt;</code> parameters, <code>&lt;join&gt;</code> <code>entertone</code> <code>exittone</code>, <code>&lt;createccxml&gt;</code> parameters.</td>
<td>May 21, 2008</td>
<td>Stéphane Bastien</td>
</tr>
<tr>
<td>15.0</td>
<td>0.2</td>
<td>Added SOAP client.</td>
<td>June 12, 2008</td>
<td>Stéphane Bastien</td>
</tr>
<tr>
<td>16.0</td>
<td>1</td>
<td>Removed limitation about <code>connection.redirect.failed</code>.</td>
<td>September 11, 2008</td>
<td>Jean-François Bertrand</td>
</tr>
<tr>
<td>16.0</td>
<td>1</td>
<td>Updated section 5.9.1 to make use of utterance shadow variable.</td>
<td>October 29, 2008</td>
<td>Jean-François Bertrand</td>
</tr>
<tr>
<td>16.0</td>
<td>1</td>
<td>Updated reference section with <code>draft-ietf-mediactrl-vxml-02.txt</code>.</td>
<td>October 30, 2008</td>
<td>Jean-François Bertrand</td>
</tr>
<tr>
<td>16.0</td>
<td>1</td>
<td>Introduced redundancy section and hints in <code>&lt;createcall&gt;</code>, <code>&lt;redirect&gt;</code>.</td>
<td>May 19, 2009</td>
<td>Stéphane Bastien</td>
</tr>
<tr>
<td>16.0</td>
<td>1</td>
<td>Edited changes and published document.</td>
<td>August 25, 2009</td>
<td>Andrea Fitzwilliam</td>
</tr>
<tr>
<td>17.0</td>
<td>1</td>
<td>Updated the document for Release 17.0.</td>
<td>January 18, 2010</td>
<td>Stéphane Bastien</td>
</tr>
<tr>
<td>17.0</td>
<td>1</td>
<td>Fixed obsolete reference.</td>
<td>March 8, 2010</td>
<td>Andrea Fitzwilliam</td>
</tr>
<tr>
<td>17.0</td>
<td>1</td>
<td>Edited changes and published document.</td>
<td>April 6, 2010</td>
<td>Andrea Fitzwilliam</td>
</tr>
<tr>
<td>17.0</td>
<td>2</td>
<td>Added surrounding <code>&lt;block&gt;</code> around play example in section 7.1.1 <code>Pass Parameters to Script in SIP Request URI Using RFC 5552</code> for EV 125349.</td>
<td>February 14, 2011</td>
<td>Jean-François Bertrand</td>
</tr>
<tr>
<td>17.0</td>
<td>2</td>
<td>Made minor editorial changes.</td>
<td>March 1, 2011</td>
<td>Goska Auerbach</td>
</tr>
<tr>
<td>17.0</td>
<td>2</td>
<td>Edited changes and published document.</td>
<td>March 1, 2011</td>
<td>Jessica Boyle</td>
</tr>
</tbody>
</table>
| 17.0    | 3       | Updated document:  
- perScriptLogging
- showMissedEvent example
- SISR example
- Not supported: finalsilence and inputmode | August 9, 2011 | Jean-François Bertrand          |
<p>| 17.0    | 3       | Updated document; fixed CLI level. | August 18, 2011 | Jean-François Bertrand          |
| 18.0    | 1       | Updated document; added Xtended Services Interface examples. | August 19, 2011 | Jean-François Bertrand          |</p>
<table>
<thead>
<tr>
<th>Release</th>
<th>Version</th>
<th>Reason for Change</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.0</td>
<td>1</td>
<td>Updated CCXML HTTP interface use of access control list.</td>
<td>September 24, 2011</td>
<td>Steve Liao</td>
</tr>
<tr>
<td>18.0</td>
<td>1</td>
<td>Edited changes and published document.</td>
<td>October 31, 2011</td>
<td>Andrea Fitzwilliam</td>
</tr>
<tr>
<td>18.0</td>
<td>2</td>
<td>Added section on required mime-types.</td>
<td>April 5, 2012</td>
<td>Jean-François Bertrand</td>
</tr>
<tr>
<td>18.0</td>
<td>2</td>
<td>Edited changes and published document.</td>
<td>April 11, 2012</td>
<td>Andrea Fitzwilliam</td>
</tr>
<tr>
<td>19.0</td>
<td>1</td>
<td>Updated document for Release 19.0.</td>
<td>November 1, 2012</td>
<td>Steve Liao</td>
</tr>
<tr>
<td>19.0</td>
<td>1</td>
<td>Edited changes and published document.</td>
<td>November 7, 2012</td>
<td>Patricia Renaud</td>
</tr>
<tr>
<td>20.0</td>
<td>1</td>
<td>Updated document for Release 20.0.</td>
<td>September 8, 2013</td>
<td>Steve Liao</td>
</tr>
<tr>
<td>20.0</td>
<td>1</td>
<td>Edited changes and published document.</td>
<td>October 23, 2013</td>
<td>Joan Renaud</td>
</tr>
<tr>
<td>21.0</td>
<td>1</td>
<td>Updated to reflect use of the VoiceXML virtual subscriber on the BroadWorks AS/XS to host the VoiceXML/CCXML script.</td>
<td>October 17, 2014</td>
<td>Steve Liao</td>
</tr>
<tr>
<td>21.0</td>
<td>1</td>
<td>Edited changes and updated BroadSoft legal notice.</td>
<td>October 31, 2014</td>
<td>Joan Renaud</td>
</tr>
<tr>
<td>21.0</td>
<td>1</td>
<td>Rebranded and published document.</td>
<td>December 22, 2014</td>
<td>Patricia Renaud</td>
</tr>
</tbody>
</table>
Table of Contents

INTRODUCTION ................................................................................................................................. 10

1 VoiceXML and CCXML Scripts ........................................................................................................ 11
   1.1 VoiceXML ................................................................................................................................. 11
   1.2 CCXML .................................................................................................................................... 11
   1.3 Components of VoiceXML or CCXML System ....................................................................... 12
       1.3.1 Calling Card (Example Application) ............................................................................ 13

2 VoiceXML .................................................................................................................................... 14
   2.1 Standards Compliancy ............................................................................................................. 14
       2.1.1 Supported File Formats ................................................................................................. 15
   2.2 Invoke VoiceXML Script ........................................................................................................ 15
       2.2.1 Specify Script in SIP INVITE ....................................................................................... 15
       2.2.2 Default VoiceXML Script ............................................................................................. 15
       2.2.3 Pre-Release 17.0 ............................................................................................................. 16
       2.2.4 Session Variables ............................................................................................................ 16
   2.3 Platform-specific Default Values .......................................................................................... 18
   2.4 Security .................................................................................................................................. 18
       2.4.1 Pre-Release 17.0 ............................................................................................................. 19
   2.5 Logs ...................................................................................................................................... 19
       2.5.1 Pre-Release 17.0 ............................................................................................................. 20
   2.6 Internationalization ............................................................................................................... 21

3 CCXML ....................................................................................................................................... 23
   3.1 Standards Compliancy ............................................................................................................. 23
       3.1.1 Supported CCXML Tags ................................................................................................. 23
       3.1.2 SIP and TEL URI ........................................................................................................... 26
   3.2 Invoke CCXML Script ............................................................................................................ 26
       3.2.1 Specify Script in SIP INVITE ....................................................................................... 26
       3.2.2 Default CCXML Script ................................................................................................. 27
       3.2.3 Start CCXML Script When Media Server Launches .................................................... 27
       3.2.4 Session Variables ............................................................................................................ 27
   3.3 Logs ...................................................................................................................................... 28
       3.3.1 Log Configurations ......................................................................................................... 28
       3.3.2 Show Missed Events in CCXML Script ....................................................................... 29

4 Redundancy ............................................................................................................................... 30
   4.1 Use DNS SRV Lookup ........................................................................................................... 30
   4.2 Use Media Server Selection Policy on Network Server ......................................................... 32
   4.3 Load VoiceXML or CCXML Script ........................................................................................ 33
   4.4 CCXML Script Originating Call ............................................................................................. 34

5 Mime-types ................................................................................................................................. 35
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Generic Components of VoiceXML or CCXML System</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Configure Virtual VoiceXML Subscriber Hosting VoiceXML Script</td>
<td>31</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Legacy Configuration using Non-Virtual Subscriber to Host a VoiceXML Script</td>
<td>31</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Activate Route Advance for BroadWorks Media Server Device Type</td>
<td>33</td>
</tr>
</tbody>
</table>
List of Tables

Table 1  Noncompliances to VoiceXML 2.0.................................................................14
Table 2  Noncompliances to VoiceXML 2.1.................................................................14
Table 3  Noncompliances to RFC 5552 .................................................................14
Table 4  Mapping of VoiceXML to SIP Session Variables ........................................17
Table 5  Media Server Default Values ......................................................................18
Table 6  Nonlocalizable Prompts .............................................................................21
Table 7  Default Audio Prompts Located Under /usr/local/broadworks/bw_base/media/en-US ..........21
Table 8  Examples for Some Languages.................................................................22
Table 9  Supported CCXML Tags .............................................................................26
Table 10 Mapping of CCXML Event to SIP Session Variable ......................................28
Table 11 Limitations of Embedded Soap Client....................................................75
1 VoiceXML and CCXML Scripts

This guide is intended for developers who write VoiceXML or CCXML scripts that run on a BroadWorks Media Server. Note however, that this document is not an introduction to VoiceXML or CCXML (for this, see the list of references [6]). Instead, this guide provides detailed information about the BroadSoft implementation of VoiceXML and CCXML and includes solutions to common problems encountered while developing VoiceXML and CCXML scripts.

1.1 VoiceXML

VoiceXML is a standard defined by the World Wide Web Consortium (W3C) that provides a scripting language optimized for building interactive voice response (IVR) dialogs. Specifically, VoiceXML can:

- Play and record audio and video prompts
- Handle dual-tone multi-frequency (DTMF) tones
- Handle text-to-speech
- Handle speech recognition
- Handle call transfer

1.2 CCXML

CCXML is a standard defined by the World Wide Web Consortium (W3C) that provides a high-level scripting language for call control. CCXML transforms the Media Server into a customer-programmable service execution platform. Media services such as Calling Cards, Voice Mail, Video Auto-Attendant, and Meet-Me Conferencing can be built in CCXML and run on a Media Server.

CCXML is a call control environment that can:

- Originate calls
- Handle incoming calls: accept calls, redirect calls, and reject calls
- Transfer calls
- Establish and control conferences

CCXML and VoiceXML scripts can be combined to control these Media Server functions:

- Play back of audio and video prompts
- Record audio and video messages
- Detection of DTMF tones
- Conference ports
- Repeaters
- Text-to-speech
- Speech recognition

Finally, CCXML scripts can also act as Hypertext Transfer Protocol (HTTP) clients and HTTP servers, allowing CCXML scripts to interact with commercial third-party application servers such as BEA WebLogic and IBM WebSphere.
1.3 Components of VoiceXML or CCXML System

The following figure shows the components of a VoiceXML or CCXML system.

![Diagram of VoiceXML or CCXML System Components]

Figure 1 Generic Components of VoiceXML or CCXML System

The components and their functions are as follows:

- **Media Server** (also known as Media Resource Function [MRF]), which is responsible for:
  - Fetching VoiceXML and CCXML scripts from an external Web Server
  - Executing VoiceXML and CCXML scripts
  - Receiving and originating calls using the Session Initiation Protocol (SIP)
  - Playing and recording audio and video prompts
  - Communicating with Text-to-Speech and Speech Recognition servers

- **Web Server**, which is responsible for:
  - Hosting VoiceXML and CCXML scripts
  - Hosting audio and video files
  - Communicating with an external database

- **SIP Proxy**, which is responsible for:
  - Proxying SIP messages to and from external SIP devices, such as SIP phones
To illustrate how these components interact with each other, a calling card application that uses these components is described in the following section.

1.3.1 Calling Card (Example Application)

In this scenario, a customer calls a 1-800 number. The call is redirected to the Media Server and the calling card CCXML script is invoked. The customer interacts with a VoiceXML session under control of the CCXML script. The Media Server validates the customer’s PIN and the time left on the customer’s calling card against the database records. It collects the destination number digits and places the outbound call to the terminating party via the SIP Proxy.

In this application, the components have these responsibilities:

- **SIP Proxy:** Transfers incoming or outbound calls to or from the Media Server and informs it of the location and name of the CCXML and VoiceXML scripts to execute. The SIP Proxy can be a third-party SIP proxy or the BroadWorks Application Server.

- **Media Server/Media Resource Function:** Accepts the incoming calls from the SIP Proxy, fetches the CCXML and VoiceXML scripts from the hosting Web Server, fetches media files specified in the VoiceXML script, and executes the CCXML and VoiceXML scripts. It can also place outbound calls via the SIP Proxy, if instructed to do so by the CCXML script. The Media Server also plays and records audio or video prompts to the caller.

- **Application Server or Web Server:** Provides CCXML and VoiceXML scripts and associated media files to the Media Server. It also exchanges information with the CCXML and VoiceXML scripts while they are running. For example, the VoiceXML script can collect a PIN number from the caller and validate the PIN against an external database via the Web server. Note that this Application Server or Web Server is not provided by BroadSoft and is typically a third-party commercial application server, such as BEA WebLogic or IBM WebSphere.

- **Database:** This is not provided by BroadSoft and is typically a third-party commercial database (for example, Oracle or MySQL).
2 VoiceXML

2.1 Standards Compliancy

The Media Server complies with the specifications VoiceXML 2.0 [5], VoiceXML 2.1 [7], and RFC 5552 [3], with the exceptions listed in the following tables.

<table>
<thead>
<tr>
<th>VoiceXML 2.0 Section</th>
<th>Noncompliances</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3.7 Transfer Element</td>
<td>The <code>transferaudio</code>, <code>aai</code>, <code>aaiexpr</code> are ignored.</td>
</tr>
<tr>
<td>Appendix E Audio File Format</td>
<td>Raw (headerless) files are not supported.</td>
</tr>
<tr>
<td>Appendix P Built-in Grammar Types</td>
<td>No built-in grammars are supported.</td>
</tr>
<tr>
<td>2.3.6 Record Element</td>
<td>The <code>finalSilence</code> attribute is ignored.</td>
</tr>
<tr>
<td>6.3.6 Miscellaneous Properties</td>
<td>The <code>inputmodes</code> property is ignored.</td>
</tr>
</tbody>
</table>

Table 1 Noncompliances to VoiceXML 2.0

<table>
<thead>
<tr>
<th>VoiceXML 2.1 Section</th>
<th>Noncompliances</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 Recording User Utterances</td>
<td>Recording during <code>&lt;transfer&gt;</code> is not supported.</td>
</tr>
<tr>
<td>7.1 Specifying the Media Format of Utterance Recordings</td>
<td>Specifying a media format with the <code>recordutterancetype</code> property is not supported. User utterances are always recorded as WAV 8 kHz G.711 ulaw.</td>
</tr>
<tr>
<td>9 Adding Type to <code>&lt;transfer&gt;</code>, 9.1, 9.2, 9.3</td>
<td>Transfer with consultation is not supported.</td>
</tr>
</tbody>
</table>

Table 2 Noncompliances to VoiceXML 2.1

<table>
<thead>
<tr>
<th>RFC 5552 Section</th>
<th>Noncompliances</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Service Identification</td>
<td>The <code>maxage</code>, <code>maxstale</code>, <code>method</code>, <code>postbody</code> are not supported.</td>
</tr>
<tr>
<td>2.4 Session Variable Mappings</td>
<td>The <code>session.connection.redirect</code> is always empty. This information can be retrieved from <code>session.connection.protocol.sip.headers</code>.</td>
</tr>
<tr>
<td>2.6.3 MRCP Establishment</td>
<td>MRCPv2 session is established after the call is answered (that is, SIP 200 OK sent to caller)</td>
</tr>
<tr>
<td>3.2 Early Media</td>
<td>Early media is not supported.</td>
</tr>
<tr>
<td>3.4 Audio and Video Codecs</td>
<td><code>RFC 2190</code> is not supported; <code>MPEG4</code> is not supported.</td>
</tr>
<tr>
<td>4.2 SIP Mechanism</td>
<td>Carrying data in SIP 200 OK in response to a SIP BYE is not supported.</td>
</tr>
<tr>
<td>6.3 Consultation</td>
<td>This is not supported.</td>
</tr>
</tbody>
</table>

Table 3 Noncompliances to RFC 5552
2.1.1 Supported File Formats

While playing or recording, the Media Server supports the following file formats: WAV, WMA, MOV, and 3GP.

When recording, you can specify the desired file type using the “type” attribute of <record>. Supported values are:

- video/quicktime for .mov files
- video/3gpp for .3gp files
- audio/x-wav for .wav files
- audio/x-ms-wma for .wma files

2.2 Invoke VoiceXML Script

In compliance with RFC 5552 and RFC 4240, the Media Server uses “dialog” as the unique service identifier for its VoiceXML/CCXML services. VoiceXML scripts can be invoked either using the VoiceXML parameters on the request uniform resource identifier (URI) of a SIP INVITE or through the default Media Server VoiceXML script.

2.2.1 Specify Script in SIP INVITE

The SIP Request URI has the following form when specifying a VoiceXML application to run:

```
INVITE sip:dialog@meije.mtl.broadsoft.com;voicexml=http://meije/vxml/sip.vxml SIP/2.0
```

The dialog class of service is requested

The Media Server shall use http://meije/vxml/sip.vxml as the VoiceXML application to run

2.2.2 Default VoiceXML Script

A default script for a request to the dialog service can be configured when no VoiceXML or CCXML Request-URI parameter is provided. This script runs when the Media Server receives a SIP INVITE with Request-URI without a VoiceXML or a CCXML parameter. Note that this applies for CCXML as well. The following is an example of such a SIP INVITE:

```
INVITE sip:dialog@meije.mtl.broadsoft.com SIP/2.0
```

The dialog class of service is requested

Since no script is specified the Media Server shall run the default VoiceXML application

The default dialog script is configurable using the Media Server command line interface (CLI), under the Applications/MediaStreaming/Services/Dialog level.
2.2.3 Pre-Release 17.0

In older releases of the Media Server, this is configured from the Service/Dialog CLI level. For example:

```
$ bwcli
======================================================================
BroadWorks Command Line Interface
  Type HELP for more information
======================================================================
Reading initial CLI command file...
MS_CLI> service;dialog
MS_CLI/Service/Dialog> set DefaultXmlScript file:///var/www/vxml/sip.vxml
...Done.
MS_CLI/Service/Dialog> get
  DefaultXmlScript = file:///var/www/vxml/sip.vxml
  perCallLogging = Y
  ShowMissedEvents = N
  RunScriptOnStartup =
  sipProxyForStartupScript = 192.168.12.46
  sipFromForStartUpScript = sip:4504615019@linsanity.mtl.broadsoft.com
MS_CLI/Service/Dialog>
```

2.2.4 Session Variables

Session Initiation Protocol (SIP) session variables are mapped to VoiceXML session variables, based on RFC 5552 section 2.4, as defined in the following table.

<table>
<thead>
<tr>
<th>VoiceXML Session Variable</th>
<th>SIP Session Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>session.connection.local.uri</td>
<td>Evaluates to the SIP URI specified in the To: header of the initial INVITE.</td>
</tr>
<tr>
<td>session.connection.remote.uri</td>
<td>Evaluates to the SIP URI specified in the From: header of the initial INVITE.</td>
</tr>
<tr>
<td>Session.connection.redirect</td>
<td>Array populated by information contained in History-Info header, in reverse order, in the initial INVITE.</td>
</tr>
<tr>
<td>Session.connection.protocol.name</td>
<td>Evaluates to “sip”.</td>
</tr>
<tr>
<td>Session.connection.protocol.version</td>
<td>Evaluates to “2.0”.</td>
</tr>
<tr>
<td>session.connection.protocol.sip.headers</td>
<td>Associative array where each key in the array is the noncompact name of a SIP header in the initial INVITE converted to lowercase (for example, CALL-ID becomes call-id).</td>
</tr>
<tr>
<td>VoiceXML Session Variable</td>
<td>SIP Session Variable</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------</td>
</tr>
</tbody>
</table>
| session.connection.protocol.sip.requesturi | Associative array where keys and values are formed from the URI parameters on the SIP Request-URI of the initial INVITE according to the following rules:  
  - If URI parameter name includes a period (for example, obj.a.1), then a property of type object is added whose name is formed from characters to the left of the first period.  
  - If the URI parameter name contains no further periods, then a string property is added. Otherwise, it is of type object and the process of adding properties repeats.  
  - If URI parameter name includes no period, then the key is formed of the entire parameter name and its value is of type string and evaluates to the URI parameter value.  
  - If the URI parameter name is present, but its value is omitted, then the value is mapped to an empty string.  

Note that the complete Request-URI is not available as specified in RFC 5552. However, the user and host part are made available through subparameter: requesturi.user and requesturi.host  

**Example:**  
sip:dialog@mediaserver.com;voicexml=http://appserver.com/myfile.vxml;obj.x=1;obj.y;obj.z.a=3  

Then the following are created:  

- session.connection.protocol.sip.requesturi[“user”] = dialog  
- session.connection.protocol.sip.requesturi[“host”] = mediaserver.com  
- session.connection.protocol.sip.requesturi[“voicexml”] = http://appserver.com/myfile.vxml  
- session.connection.protocol.sip.requesturi[“obj”].x = 1  
- session.connection.protocol.sip.requesturi[“obj”].y = “”  
- session.connection.protocol.sip.requesturi[“obj”].z.a = 3 |
| session.connection.protocol.sip.media | Associated array where each element is an object with two properties: “type” and “direction”.  

**Example:**  
m=audio 16480 RTP/AVP 4 0  
a=rtpmap:4 G723/8000  
a=rtpmap:0 PCMU/8000  
... becomes:  

- session.connection.protocol.sip.media[0].type = “audio”  
- session.connection.protocol.sip.media[0].direction = “sendrecv” |

Table 4 Mapping of VoiceXML to SIP Session Variables
2.3 Platform-specific Default Values

The VoiceXML specification indicates that the default value for many parameters is platform-specific. The following table specifies these default values for the BroadWorks Media Server.

<table>
<thead>
<tr>
<th>Tag or Function</th>
<th>Name</th>
<th>VoiceXML 2.0 Section</th>
<th>BroadSoft Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;vxml&gt;</td>
<td>xml:lang</td>
<td>1.5.1</td>
<td>en-US</td>
</tr>
<tr>
<td>&lt;record&gt;</td>
<td>maxtime</td>
<td>2.3.6</td>
<td>30 seconds</td>
</tr>
<tr>
<td>&lt;record&gt;</td>
<td>type</td>
<td>2.3.6</td>
<td>WAV, 8 kHz, G.711 uLaw</td>
</tr>
<tr>
<td>&lt;prompt&gt;</td>
<td>bargeintype</td>
<td>4.1.5.1</td>
<td>Speech</td>
</tr>
<tr>
<td>&lt;prompt&gt;</td>
<td>timeout</td>
<td>4.1</td>
<td>10 seconds</td>
</tr>
<tr>
<td>Fetching audio, document, or script</td>
<td>*maxage</td>
<td>6.3.5</td>
<td>As per section 6.1.2, no default value provided</td>
</tr>
<tr>
<td></td>
<td>*maxstale</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5  Media Server Default Values

2.4 Security

Different strategies are used to handle the security aspect of VoiceXML applications.

Part of handling the security aspect of VoiceXML applications is to detect a VoiceXML script error that would result in infinite loops. Infinite loops are logical errors built into VoiceXML applications. Even if it is not possible to catch all infinite loops, the VoiceXML interpreter attempts to catch as many as possible. When such a situation is discovered, an _ot_internal.loop event is generated and the VoiceXML application aborts.

Another part of handling the security aspect of VoiceXML applications is to control the sandbox in which the VoiceXML application is running. Different statistics are monitored to ensure that one interpretation does not take up all available resources:

- VoiceXML transition count
- Total size of downloaded data
- Size of recorded messages
- Count of HTTP get or post
- Depth of recursion of European Computer Manufacturers Association (ECMA) script
- Depth of recursion of sub-dialog

These statistics are monitored continuously and compared against thresholds defined in MS_CLI/Applications/MediaStreaming/Service/Dialog/Security. If a VoiceXML script exceeds a threshold, alarm msVXMLSecurityThresholdExceeded is raised and the VoiceXML script is aborted. For example:

```
======================================================================
BroadWorks Command Line Interface
Type HELP for more information
======================================================================
Reading initial CLI command file...
MS_CLI> applications;mediastreaming; service;dialog;security
MS_CLI/Applications/MediaStreaming/Services/Dialog/Security> g
```
maxVxmlTransitionCount = 100
maxHttpPostCount = 25
maxHttpGetCount = 25
maxHttpGetTimeout = 6
maxDownloadedSize = 50
maxRecordedSize = 300
maxDepthOfEcmaScriptRecursion = 10
maxDepthOfSubDialogRecursion = 10

An audit mechanism is also used to catch all VoiceXML interpretations that would be leaking and thus reducing the server capacity of the Media Server to handle new sessions. This is the same audit mechanism used by other Media Server services.

2.4.1 Pre-Release 17.0

In older releases of the Media Server, security thresholds are configured from Service/Dialog/Security. For example:

```
MS_CLI> service;dialog;security
MS_CLI/Service/Dialog/Security> g
maxVxmlTransitionCount = 100
maxHttpPostCount = 25
maxHttpGetCount = 25
maxHttpGetTimeout = 6
maxDownloadedSize = 50
maxRecordedSize = 300
maxDepthOfEcmaScriptRecursion = 10
maxDepthOfSubDialogRecursion = 10
```

2.5 Logs

Logs generated by VoiceXML applications are controlled by two different input channels:

- **Vxmlfe**: Main input channel that collects all logs. Logs are written to `/var/broadworks/logs/mediaservers0/msfe<date>.txt`.
- **SessionFile**: User logs. Logs are written to `/var/broadworks/logs/scriptexec/<script name><dn><date>.log`

An input channel must be enabled to collect and output logs to its associated log file under `/var/broadworks/logs/logs`.

User logs are used to examine and troubleshoot calls that have been placed to a VoiceXML application on the Media Server. Starting in Release 17.0, user logs are controlled by the SessionFile input channel. By default, user logs are disabled. To enable them, the system administrator must use the CLI under the `Applications/MediaStreaming/Logging/InputChannels` level and set the `enabled` attribute on `SessionFile` to “true”. For example:

```
$ bwcli
BroadWorks Command Line Interface
Type HELP for more information
Reading initial CLI command file...
MS_CLI> applications;mediastreaming;logging;inputchannels
```
User logs contain the output of VoiceXML `<log>` statements, plus basic information about incoming calls as well as what occurred during the call. Each line is has the time and a tag that identifies the type of log (ALW, ERR, WARN, and so on) followed by the actual log. These logs are used by VoiceXML script developers.

User logs are stored, on the Media Server file system, under `/var/broadworks/logs/scriptexec`. User logs are named using the following convention: `<voiceXML script name>-<From: user part>-<Date-Time>-<execution count>.log`.


All logs in `user logs` are also in `ms.syslog`, which are used for BroadSoft debugging.

As with other log files, user logs are managed by the `bwAutoCleanup` cronjob. The BroadWorks operator MUST activate this task to avoid filling the disk with user logs. It is recommended that it run on a daily basis. This is done via the CLI, from the `Maintenance/Scheduler` level. By default, user logs older than two days are compressed and user logs older than 30 days are deleted.

### 2.5.1 Pre-Release 17.0

In older releases of the Media Server, user logs are enabled by the system administrator using the CLI under `Service/Dialog` level and set `perCallLogging` to “true”. For example:

```bash
$ bwcli
BroadWorks Command Line Interface
Type HELP for more information
Reading initial CLI command file...
MS_CLI> service;dialog
MS_CLI/Service/Dialog> set perCallLogging true
...Done
MS_CLI/Service/Dialog> get
DefaultXmlScript = http://meije/vxml/rec.vxml
perCallLogging = Y
ShowMissedEvents = N
RunScriptOnStartup =
```
2.6 Internationalization

The language to use for a VoiceXML script is specified in the xml:lang attribute of the <vxml> element, located at the beginning of the VoiceXML script. If no xml:lang attribute is present, the Media Server assumes that the language is U.S. English and provides a set of default audio prompts.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/local/broadworks/bw_base/media/vxmlBeep.wav</td>
<td>Default system beep</td>
</tr>
<tr>
<td>/usr/local/broadworks/bw_base/media/Ringbacktone.wav</td>
<td>Default ringback tone</td>
</tr>
</tbody>
</table>

Table 6 Nonlocalizable Prompts

<table>
<thead>
<tr>
<th>File Name</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>vxmlNoHelp.wav</td>
<td>No help available.</td>
</tr>
<tr>
<td>vxmlTimeLimitExceeded.wav</td>
<td>Your speech exceeded time limit.</td>
</tr>
<tr>
<td>vxmlUnhandledError.wav</td>
<td>An unhandled error event occurred. The interpretation cannot continue.</td>
</tr>
<tr>
<td>vxmlUnrecognizedInput.wav</td>
<td>Unrecognized input.</td>
</tr>
</tbody>
</table>

Table 7 Default Audio Prompts Located Under /usr/local/broadworks/bw_base/media/en-US

Two steps are required to add support for a new language:

1) Prompts from Table 7 must be recorded in the target language as a WAV file, 8 kHz G.711 ulaw.

2) Recorded prompts must be stored under /usr/local/broadworks/bw_base/media in a directory named after the language code (for example, fr-CA).

The language codes must follow RFC 4646 “Tags for Identifying Languages” [8]. The following are examples for some languages:

<table>
<thead>
<tr>
<th>Locale</th>
<th>Language</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-AT</td>
<td>German, Austria</td>
<td>/usr/local/broadworks/bw_base/media/de-AT</td>
</tr>
<tr>
<td>de-CH</td>
<td>German, Switzerland</td>
<td>/usr/local/broadworks/bw_base/media/de-CH</td>
</tr>
<tr>
<td>de-DE</td>
<td>German, Germany</td>
<td>/usr/local/broadworks/bw_base/media/de-DE</td>
</tr>
<tr>
<td>en-AU</td>
<td>English, Australia</td>
<td>/usr/local/broadworks/bw_base/media/en-AU</td>
</tr>
<tr>
<td>en-GB</td>
<td>English, Great Britain</td>
<td>/usr/local/broadworks/bw_base/media/en-GB</td>
</tr>
<tr>
<td>en-NZ</td>
<td>English, New Zealand</td>
<td>/usr/local/broadworks/bw_base/media/en-NZ</td>
</tr>
<tr>
<td>en-UK</td>
<td>English, United Kingdom</td>
<td>/usr/local/broadworks/bw_base/media/en-UK</td>
</tr>
<tr>
<td>en-US</td>
<td>English, United State</td>
<td>/usr/local/broadworks/bw_base/media/en-US</td>
</tr>
<tr>
<td>es-ES</td>
<td>Spanish, Spain</td>
<td>/usr/local/broadworks/bw_base/media/es-ES</td>
</tr>
<tr>
<td>es-MX</td>
<td>Spanish, Mexico</td>
<td>/usr/local/broadworks/bw_base/media/es-MX</td>
</tr>
<tr>
<td>Locale</td>
<td>Language</td>
<td>Path</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>fr-BE</td>
<td>French, Belgium</td>
<td>/usr/local/broadworks/bw_base/media/fr-BE</td>
</tr>
<tr>
<td>fr-CA</td>
<td>French, Canada</td>
<td>/usr/local/broadworks/bw_base/media/fr-CA</td>
</tr>
<tr>
<td>fr-CH</td>
<td>French, Switzerland</td>
<td>/usr/local/broadworks/bw_base/media/fr-CH</td>
</tr>
<tr>
<td>fr-FR</td>
<td>French, France</td>
<td>/usr/local/broadworks/bw_base/media/fr-FR</td>
</tr>
<tr>
<td>it-IT</td>
<td>Italian, Italy</td>
<td>/usr/local/broadworks/bw_base/media/it-IT</td>
</tr>
<tr>
<td>pt-BR</td>
<td>Portuguese, Brazil</td>
<td>/usr/local/broadworks/bw_base/media/pt-BR</td>
</tr>
<tr>
<td>pt-PT</td>
<td>Portuguese, Portugal</td>
<td>/usr/local/broadworks/bw_base/media/pt-PT</td>
</tr>
<tr>
<td>ru-RU</td>
<td>Russian, Russia</td>
<td>/usr/local/broadworks/bw_base/media/ru-RU</td>
</tr>
</tbody>
</table>

Table 8  Examples for Some Languages

There is no limit to the number of simultaneous languages that can be supported on the Media Server.
3 CCXML

3.1 Standards Compliancy

The Media Server is compliant with the CCXML 1.0 draft specification dated June 2005 [1] as well as the following elements from the January 2007 draft [2]:

- Section 7.2.1 and 7.2.2 Attribute Parameters
- Section 7.3 Events (dialog.started, dialog.exit, dialog.disconnect, dialog.transfer, dialog.terminatetransfer, error.dialog, error.dialog.nostarted, dialog.prepared, dialog.notprepared)
- Section 7.4 Dialog Object
- Section 9.2.2 Transition Variable Events
- Section 10.2.2 Connection Object
- Section 10.6.5 Event connection.disconnected
- Section 10.6.10 Event connection.signal
- Appendix D VoiceXML Integration
- Appendix K

The following elements from the January 2007 draft are not available:

- Appendix L (due to security risks)
- Section 7.5 Dialog Class
- Section 9.2.4 <move>
- Section 10.3.2 Conference Class
- Section 10.4.3 Media Endpoint Properties
- Section 10.5.10 <merge>

3.1.1 Supported CCXML Tags

The following table provides a list of supported CCXML tags with their supported parameters. If an attribute is not in this table, it is ignored by the Media Server.

<table>
<thead>
<tr>
<th>CCXML Section</th>
<th>CCXML Tag</th>
<th>Supported Attributes</th>
<th>Attribute Required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.4</td>
<td>&lt;if&gt;</td>
<td>Cond</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>6.2.5</td>
<td>&lt;elseif&gt;</td>
<td>Cond</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>6.2.6</td>
<td>&lt;else&gt;</td>
<td>None</td>
<td>Not applicable</td>
<td></td>
</tr>
<tr>
<td>6.2.7</td>
<td>&lt;fetch&gt;</td>
<td>Next</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Namelist</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fetched</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>6.2.8</td>
<td>&lt;goto&gt;</td>
<td>Fetched</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>6.2.9</td>
<td>&lt;createccxml&gt;</td>
<td>next</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>namelist</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>CCXML Section</td>
<td>CCXML Tag</td>
<td>Supported Attributes</td>
<td>Attribute Required</td>
<td>Comments</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------</td>
<td>----------------------</td>
<td>--------------------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>fetchparam</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>parameters</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>method</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>sessionid</td>
<td>Optional</td>
<td></td>
<td>This is not supported.</td>
</tr>
<tr>
<td></td>
<td>timeout</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>maxage</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>maxstale</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>enctype</td>
<td>Optional</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.2.10</td>
<td>&lt;exit&gt;</td>
<td>expr</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>namelist</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>6.2.11</td>
<td>&lt;log&gt;</td>
<td>label</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>expr</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>7.2.1</td>
<td>&lt;dialogprepare&gt;</td>
<td>src</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>type</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>namelist</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameters</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>dialogid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>connectionid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>conferenceid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mediadirection</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>method</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>7.2.2</td>
<td>&lt;dialogstart&gt;</td>
<td>src</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>prepareddialogid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>namelist</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>parameters</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>dialogid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>connectionid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>conferenceid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mediadirection</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>method</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>7.2.3</td>
<td>&lt;dialogterminate&gt;</td>
<td>dialogid</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>8.2.1</td>
<td>&lt;assign&gt;</td>
<td>name</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>expr</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>8.2.2</td>
<td>&lt;script&gt;</td>
<td>src</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>fetchid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>CCXML Section</td>
<td>CCXML Tag</td>
<td>Supported Attributes</td>
<td>Attribute Required</td>
<td>Comments</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>----------------------</td>
<td>--------------------</td>
<td>----------</td>
</tr>
<tr>
<td>9.2.1</td>
<td>&lt;eventprocessor&gt;</td>
<td>statevariable</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>9.2.2</td>
<td>&lt;transition&gt;</td>
<td>state</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>event</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cond</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>9.2.3</td>
<td>&lt;send&gt;</td>
<td>target</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>targettype</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>delay</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>name</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>namelist</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>9.2.4</td>
<td>&lt;move&gt;</td>
<td></td>
<td></td>
<td>This tag is not supported.</td>
</tr>
<tr>
<td>9.2.5</td>
<td>&lt;cancel&gt;</td>
<td>sendid</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>10.5.1</td>
<td>&lt;accept&gt;</td>
<td>connectionid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>10.5.2</td>
<td>&lt;redirect&gt;</td>
<td>connectionid</td>
<td>Optional</td>
<td>The dest must start with either tel: or sip:.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dest</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>hints</td>
<td>Optional</td>
<td>Sets optional SIP headers in SIP 302 and SIP REFER.</td>
</tr>
<tr>
<td>10.5.3</td>
<td>&lt;reject&gt;</td>
<td>connectionid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reason</td>
<td>Optional</td>
<td>SIP response code to provide in response to SIP INVITE. By default, SIP 603 is returned. Example: &lt;reject reason=&quot;600&quot; /&gt;</td>
</tr>
<tr>
<td>10.5.4</td>
<td>&lt;createcall&gt;</td>
<td>dest</td>
<td>Required</td>
<td>The dest must start with either tel: or sip:.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>connectionid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>callerid</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>hints</td>
<td>Optional</td>
<td>Sets optional SIP headers in SIP INVITE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>timeout</td>
<td>Optional</td>
<td>Default is 30 seconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>joined</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>joindirection</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>10.5.5</td>
<td>&lt;createconference&gt;</td>
<td>conferenceid</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>confname</td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reservedtalkers</td>
<td>Optional</td>
<td>The valid range is 3 through 149.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reservedlisteners</td>
<td>Optional</td>
<td>The valid range is 1 through 400.</td>
</tr>
<tr>
<td>CCXML Section</td>
<td>CCXML Tag</td>
<td>Supported Attributes</td>
<td>Attribute Required</td>
<td>Comments</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------</td>
<td>----------------------</td>
<td>--------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>10.5.6</td>
<td><code>&lt;destroyconference&gt;</code></td>
<td><code>conferenceid</code></td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>10.5.7</td>
<td><code>&lt;join&gt;</code></td>
<td><code>id1</code></td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>id2</code></td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>duplex</code></td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>entertone</code></td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>exittone</code></td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>dtmfclamp</code></td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>10.5.8</td>
<td><code>&lt;unjoin&gt;</code></td>
<td><code>id1</code></td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>id2</code></td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>10.5.9</td>
<td><code>&lt;disconnect&gt;</code></td>
<td><code>connectionid</code></td>
<td>Optional</td>
<td></td>
</tr>
<tr>
<td>10.5.10</td>
<td><code>&lt;merge&gt;</code></td>
<td></td>
<td></td>
<td>This tag is not supported.</td>
</tr>
</tbody>
</table>

Table 9  Supported CCXML Tags

3.1.2 SIP and TEL URI

For outgoing calls and transfers, the Media Server accepts URIs in the form of the following:

- `tel:1234567`
- `sip:user@host`

The first form (`tel:1234567`) is preferable to the second because it leads to simpler CCXML scripts. The complexity of finding a host that can handle the phone number is left to the Media Server. In the second form (`sip:user@host`), it is up to the CCXML script to ensure that the user and host are valid.

3.2 Invoke CCXML Script

The Media Server reuses the VoiceXML "dialog" keyword to invoke CCXML scripts, as defined in `RFC 4240` and `RFC 5552`.

3.2.1 Specify Script in SIP INVITE

The SIP Request URI has the following form when specifying a CCXML script to run:

```
INVITE sip:dialog@meije.mtl.broadsoft.com;ccxml=http://meije/ccxml/script.ccxml SIP/2.0
```

The dialog class of service is requested

The Media Server shall use

http://meije/ccxml/script.ccxml as the

CCXML script to run
3.2.2 Default CCXML Script

A default script for a request to the dialog service can be configured when no VoiceXML or CCXML Request-URI parameter has been provided. For more information, see section 2.2.2 Default VoiceXML Script.

3.2.3 Start CCXML Script When Media Server Launches

An alternate method to invoke a CCXML script is to launch a script whenever the Media Server starts. This script can then listen for HTTP POST queries from external servers and perform commands as requested in the query.

The script to invoke on startup is specified using the CLI at the MS_CLI/Applications/MediaStreaming/Service/Dialog/ RunScriptOnStartup level.

In pre-Release 17.0, this was configured from MS_CLI/Service/Dialog/RunScriptOnStartup.

3.2.4 Session Variables

Session Initiation Protocol (SIP) session variables are mapped to CCXML connection.alerting.info event, based on RFC 5552 section 2.4, as defined in the following table.

<table>
<thead>
<tr>
<th>CCXML Event</th>
<th>SIP Session Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection.alerting.info.local.uri</td>
<td>Evaluates to the SIP URI specified in the To: header of the initial INVITE.</td>
</tr>
<tr>
<td>connection.alerting.info.remote.uri</td>
<td>Evaluates to the SIP URI specified in the From: header of the initial INVITE.</td>
</tr>
<tr>
<td>connection.alerting.info.redirect</td>
<td>Array populated by information contained in History-Info header, in reverse order, in the initial INVITE.</td>
</tr>
<tr>
<td>connection.alerting.info.protocol_name</td>
<td>Evaluates to &quot;sip&quot;.</td>
</tr>
<tr>
<td>connection.alerting.info.protocol_version</td>
<td>Evaluates to &quot;2.0&quot;.</td>
</tr>
<tr>
<td>connection.alerting.info.sip.headers</td>
<td>Associative array where each key in the array is the noncompact name of a SIP header in the initial INVITE converted to lowercase (for example, CALL-ID becomes call-id).</td>
</tr>
</tbody>
</table>
CCXML Event | SIP Session Variable
--- | ---
connection.alerting.info.sip.requesturi | Associative array where keys and values are formed from the URI parameters on the SIP Request-URI of the initial INVITE according to the following rules:
- If URI parameter name includes a period (for example, obj.a.1), then a property of type object is added whose name is formed from characters to the left of the first period.
- If the URI parameter name contains no further periods, then a string property is added. Otherwise, it is of type object and the process of adding properties repeats.
- If URI parameter name includes no period, then the key is formed of the entire parameter name and its value is of type string and evaluates to the URI parameter value.
- If the URI parameter name is present, but its value is omitted, then the value is mapped to an empty string.

Note that the complete Request-URI is not available as specified in RFC 5552. However, the user and host part are made available through sub-parameter: requesturi.user and requesturi.host

Example:
sip:dialog@mediaserver.com;ccxml=http://appserver.com/myfile.ccxml;obj.x=1;obj.y;obj.z.a=3

Then the following are created:
connection.alerting.info.sip.requesturi[“user”] = dialog
connection.alerting.info.sip.requesturi[“host”] = mediaserver.com
connection.alerting.info.sip.requesturi[“ccxml”] = http://appserver.com/myfile.ccxml
connection.alerting.info.sip.requesturi[“obj”].x = 1
connection.alerting.info.sip.requesturi[“obj”].y = “”
connection.alerting.info.sip.requesturi[“obj”].z.a = 3

connection.alerting.info.sip.media | Associated array where each element is an object with two properties: “type” and “direction”.

Example:
m=audio 16480 RTP/AVF 4 0
a=rtpmap:4 G723/8000
a=rtpmap:0 PCMU/8000
... becomes:
connection.alerting.info.sip.media[0].type = “audio”
connection.alerting.info.sip.media[0].direction = “sendrecv”

Table 10  Mapping of CCXML Event to SIP Session Variable

3.3 Logs

3.3.1 Log Configurations

Logging for CCXML scripts is the same as logging for VoiceXML scripts, which is described in section 2.5 Logs.
3.3.2 Show Missed Events in CCXML Script

To view missed events in a CCXML script, use the `showMissedEvents` CLI attribute. The `showMissedEvents` attribute is available in the 
Applications/MediaStreaming/Services/Dialog CLI level. For example, it can be set as follows:

```
MS_CLI/Applications/MediaStreaming/Services/Dialog> set showmissedevents true
...Done
MS_CLI/Applications/MediaStreaming/Services/Dialog> get
defaultXmlScript =
showMissedEvents = true
runScriptOnStartup = file:///var/www/vxml/originateAndPlay/parent.ccxml
sipProxyForStartupScript = sip:5146992505@lefroy-rhel54
ccxmlHttpServer =
ccxmlHttpServerHost = 192.168.8.22
ccxmlHttpServerPort = 5080
```

With this enabled, missed events are logged. For example, an invalid `<accept/>` in an application that initiates a call results in the following:

```
15:30:24.306 {child.ccxml-unknown-2010.12.01-15.30.22--1} <Interpretation tracking>
[CCXML Session Interpreter] "Missed event" : {
  name: "error.connection.wrongstate",
  eventid: "ccxmlEvent_12",
  eventsourcetype: "ccxml",
  eventtarget: "child.ccxml-unknown-2010.12.01-15.30.22--1",
  delayed: "0s",
  connectionid: "ccxmlConn_9",
  reason: "Connection was not in a state suitable for <accept>.",
  tagname: "accept"
}
```

3.3.2.1 Pre-Release 17.0

In older releases of the Media Server, the show missed events option is enabled by the system administrator using the CLI under Service/Dialog level and set `showMissedEvents` to "true". For example:

```
$ bwcli
--------------------------------------------------------------------------------
BroadWorks Command Line Interface
Type HELP for more information
--------------------------------------------------------------------------------
Reading initial CLI command file...

MS_CLI> service;dialog
MS_CLI/Service/Dialog> set ShowMissedEvents true
...Done

MS_CLI/Service/Dialog> get
DefaultXmlScript = http://meije/vxml/rec.vxml
perCallLogging = Y
ShowMissedEvents = Y
RunScriptOnStartup =
SipProxyForStartupScript = 192.168.12.46
SipFromForStartUpScript = sip:4504615019@linsanity.mtl.broadsoft.
CCXMLHttpServer =
CCXMLHttpServerHost =
CCXMLHttpServerPort =
```

©2014 BROADSOFT, INC. PAGE 29 OF 78
4 Redundancy

Redundancy of Media Servers is provided by deploying Media Servers in a pool using an N+M scheme (M must be greater to or equal to 1). When a Media Server in a pool fails, new calls are redirected by the Application Server (AS) to other Media Servers in the pool. Existing VoiceXML and CCXML scripts and calls running on the server that failed are lost.

Note that planned outages such as hardware platform maintenance, applying patches, or upgrades do not cause script or call failures, because the Media Server is put in a Locked state prior to shutting it down. While in Locked state, the Media Server refuses new calls, the Application Server redirects these calls to other servers in the pool, and existing scripts and calls handled by the Media Server are allowed to complete as usual.

The options for configuring the Application Server and Network Server (NS) to route advance when Media Server failure occurs are described in the following section.

4.1 Use DNS SRV Lookup

Redundancy of Media Servers can be provided by using domain name system (DNS) service locator (SRV) lookup on the Application Server. To configure this redundancy scheme, follow these steps:

1) On the DNS, create an SRV record fully qualified domain name (FQDN) that includes all Media Servers in the pool. They should all have equal weighting.

2) Create a new VoiceXML virtual subscriber on the BroadWorks Application Server. This subscriber is called “the subscriber hosting the script”.

3) Assign a phone number to this subscriber. For deployments in IMS networks, the VoiceXML virtual subscriber can also be configured with a public service identity (PSI).

4) Create a new “Identity/Device Profile”. Only device types that are static registration capable are displayed. One such device is the BroadWorks Media Server device type.

5) In the SIP: contact field, enter the SRV record FQDN.

For a VoiceXML script:
dialog@fully.qualified.domain.name;voicexml=http://webserver/myscript.vxml

For a CCXML script:
dialog@fully.qualified.domain.name;ccxml=http://webserver/myscript.ccxml

When a caller dials the phone number of the subscriber hosting the script, the Application Server resolves the FQDN using a DNS SRV query, and then sends a SIP INVITE to all IP addresses found in the SRV response, one IP address at a time. The Application Server keeps on trying IP addresses as long as the Application Server gets a SIP 486 Busy response or the SIP INVITE request times out. The Application Server stops route advancing when it gets a SIP 200 OK response or a non-SIP 486 final response.
The following figure shows the configuration steps described earlier.

Figure 2 Configure Virtual VoiceXML Subscriber Hosting VoiceXML Script

Note that the Line/Port field must be filled with a unique value. It is not used with VoiceXML scripts, but it must match the From: field in SIP INVITEs originated by a CCXML script.

The legacy method of creating a non-virtual subscriber to host the VoiceXML script is still supported for non-IMS deployments, although the preferred method is via a virtual VoiceXML subscriber. The device types presented for non-virtual subscribers are not limited to static registration capable devices. Care must be taken to select a device type that is static registration-capable. It is recommended to select the BroadWorks Media Server device type.

The following figure shows the configuration steps for a non-virtual subscriber hosting the VoiceXML script.

Figure 3 Legacy Configuration using Non-Virtual Subscriber to Host a VoiceXML Script
4.2 Use Media Server Selection Policy on Network Server

An alternate method to provide redundancy involves triggering the Media Server Selection policy on the BroadWorks Network Server. This policy is typically used when Media Servers are distributed geographically at the service provider point-of-presence (POP). This policy returns an ordered list of Media Servers, from the closest to the caller to the farthest from the caller. The Application Server then tries to send a SIP INVITE to Media Servers in the order specified by the Network Server.

The following steps show how to configure redundancy that uses the Media Server Selection policy:

1) Create the Media Server Selection policy on the Network Server cluster.
2) Make sure that the Static Registration Capable box and the Route Advance box are checked for the device type that is to be used by the VoiceXML virtual subscriber (see Figure 4). The BroadWorks Media Server device type is one such device that can be configured for this purpose.
3) Create a new VoiceXML virtual subscriber on the BroadWorks Application Server. This subscriber is called “the subscriber hosting the script”.
4) Assign a phone number to this subscriber.
5) Create a new “Identity/Device Profile” and select the previously mentioned device type.
6) In the SIP: contact field, enter the following:
   For a VoiceXML script:
   \texttt{dialog@ns.cluster.fqdn;voicexml=http://webserver/myscript.vxml}
   For a CCXML script:
   \texttt{dialog@ns.cluster.fqdn;ccxml=http://webserver/myscript.ccxml}

When a caller dials the phone number of the subscriber hosting the script, the Application Server sends a SIP INVITE to the Network Server (NS) cluster. The Network Server provides a SIP 302 response with an ordered list of Media Servers. The Application Server then sends a SIP INVITE to all IP addresses specified by the Network Server, one IP address at a time. The Application Server keeps on trying IP addresses as long as the Application Server gets a SIP 486 Busy response or the SIP INVITE request times out. The Application Server stops route advancing when it gets a SIP 200 OK response or a non-486 response.

Note that such a setup prevents a regular redirection (302 without Diversion) of terminating calls by the CCXML script. When configured with the Route Advance device option, the Application Server/Execution Server interprets the 302 response as a route advancing method instead of a redirection.
Figure 4 shows how to enable Route Advance for the BroadWorks Media Server device type.

The legacy method of creating a non-virtual subscriber to host the VoiceXML script is still supported for non-IMS deployments, although the preferred method is via a virtual VoiceXML subscriber.

The following section describes redundancy issues after a SIP INVITE has reached the Media Server.

4.3 Load VoiceXML or CCXML Script

After a SIP INVITE has reached a live Media Server and the Media Server decides that it has the capacity to accept the call (that is, it is not overloaded), the Media Server tries to load the VoiceXML or CCXML script either from the file system (for SIP requests of the type `sip:dialog@ms;voicexml=file:///directory/myscript.vxml`) or from a Web Server (for SIP requests of the type `sip:dialog@ms;voicexml=http://webserver/myscript.vxml`).

The behavior of the Media Server depends on the access type and the script type:

- For a CCXML script stored on the file system, if the file is present and can be loaded, the control of the call is passed to the CCXML script. If the file cannot be loaded or does not contain a valid CCXML script, the Media Server returns a SIP 400 Invalid request response.
- For a CCXML script stored on a Web Server, the Media Server resolves the Web Server address using a DNS A-record lookup. Then the Media Server performs an HTTP GET on the first IP address returned.
  - If the script can be downloaded and is valid, the control of the call is passed to the CCXML script.
If the Web Server returns an error or the downloaded file does not contain a valid CCXML script, the Media Server returns a SIP 400 Invalid request response.

If the Web Server does not provide a response and multiple IP addresses were returned in the DNS A-record lookup, the next IP address is tried. When all IP addresses are exhausted, the Media Server returns a SIP 400 Invalid request response.

For a VoiceXML script stored on the file system, the Media Server provides a SIP 200 OK response. When the Media Server gets the SIP ACK, it tries to load the script from the file system. If the script is successfully loaded and is valid, the script is executed. Otherwise, the Media Server terminates the call by sending a SIP BYE to the Application Server.

For a VoiceXML script stored on a Web Server, the Media Server resolves the Web Server address using a DNS A-record lookup. Then the Media Server performs an HTTP GET on the first IP address returned.

- If the script is successfully downloaded and is valid, the script is executed.
- If the Web Server returns an error or the downloaded file does not contain a valid VoiceXML script, the Media Server terminates the call by sending a SIP BYE to the Application Server.
- If the Web Server does not provide a response and multiple IP addresses were returned in the DNS A-record lookup, the next IP address is tried. When all IP addresses are exhausted, the Media Server terminates the call by sending a SIP BYE to the Application Server.

The following section describes how a CCXML script originates a call in a redundant configuration.

### 4.4 CCXML Script Originating Call

When a CCXML script originates a call, the Media Server resolves the host portion of the SIP request URI (specified in parameter dest of <createcall>) by performing an A-record or AAAA-record lookup. The Media Server then tries all IP-addresses returned one after the other until it gets a response. The SIP response, whether a 200 OK, 3XX, 4XX, 5XX, or 6XX, is passed back to the CCXML script for further processing.

Therefore, if the A-/AAAA-record lookup resolves in the IP addresses of BroadWorks Application Server primary and secondary servers, the Media Server tries the primary first. If the Media Server does not get a response, it tries the secondary.
The Media Server VoiceXML interpreter uses mime-types to determine how it should handle a file retrieved from a Web Server. Therefore, the following mime-types should be returned by the Web Server hosting VoiceXML/CCXML applications, grammar files, and media files.

**Mime-types used by BroadWorks**

<table>
<thead>
<tr>
<th>Mime-type</th>
<th>MimeType</th>
</tr>
</thead>
<tbody>
<tr>
<td>application/srgs+xml</td>
<td>grxml</td>
</tr>
<tr>
<td>application/voicexml+xml</td>
<td>vxml</td>
</tr>
<tr>
<td>application/ccxml+xml</td>
<td>ccxml</td>
</tr>
<tr>
<td>audio/x-wav</td>
<td>wav</td>
</tr>
<tr>
<td>audio/x-ms-wma</td>
<td>wma</td>
</tr>
<tr>
<td>audio/mpeg</td>
<td>mp3</td>
</tr>
<tr>
<td>video/quicktime</td>
<td>mov</td>
</tr>
<tr>
<td>video/3gpp</td>
<td>3gp</td>
</tr>
</tbody>
</table>

These are typically configured in `/etc/mime.types`. 
6 Licensing

The VoiceXML interpreter, CCXML interpreter, and MRCPv2 client are built into the BroadWorks Media Server, but these services must be activated using a license.

For VoiceXML, the license control is based on a per-port usage. The `numVoiceXMLPorts` parameter limits the number of simultaneous VoiceXML sessions that can be executed on the Media Server.

CCXML is also licensed on a per-port basis. The `numCCXMLPorts` parameter limits the number of simultaneous CCXML sessions that can be executed on the Media Server.

Finally, for ASR and TTS resources invoked in the scripts to work, an MRCPv2 license is required. MRCPv2 is licensed as a system-wide feature on the Media Server. The `MRCPv2` parameter controls whether or not the MRCPv2 client is active. If the MRCPv2 client is inactive, VoiceXML scripts using ASR and TTS resources receive an “error.noresource” exception.

The following shows a sample license file with VoiceXML, CCXML, and MRCPv2 enabled.

```xml
<?xml version="1.0" encoding="UTF-8"?><com.broadsoft.apm.managedservice.BWLicense bWVersion="14.sp2"><licensedHostIDs><hostArray><string value="001422728062"></string></hostArray></licensedHostIDs><licensedServicesArray><mediaServer.ServiceLicense serviceName="MediaServerLicenseFile" numPorts="100" numG729Ports="30" numAMRPorts="30" numVoiceXMLPorts="50" MRCPv2="y" /></licensedServicesArray></com.broadsoft.apm.managedservice.BWLicense>
```

The license file must be installed in the `/usr/local/broadworks/bw_base/conf` directory.
7 VoiceXML and CCXML Cookbook

This section provides a list of tasks and problems that VoiceXML and CCXML developers commonly encounter. For each task or problem, a description of the task or problem is presented, followed by its solution and code example.

This presentation style, which can be quickly reviewed and used by programmers, is inspired from popular books such as “Perl Cookbook” and “Secure Programming Cookbook”.

7.1 Start VoiceXML and CCXML Scripts

7.1.1 Pass Parameters to Script in SIP Request URI Using RFC 5552

Problem
Some VoiceXML and CCXML scripts require parameters whenever they are invoked.

Solution
The `session.connection.protocol.sip.requesturi` can be used to pass parameters to a VoiceXML script. The following script provides a detailed example of this capability, assuming that the Media Server received a SIP INVITE with a SIP request URI containing the following:

```
sip:dialog@ms;voicexml=http://as/play.vxml;myfile=http://as/menu.wav
```

The VoiceXML script “play.vxml” shown plays the prompt passed as a parameter.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
<form>
  <block>
    <prompt>
      <audio expr="session.connection.protocol.sip.requesturi['myfile']" />
    </prompt>
  </block>
</form>
</vxml>
```

7.1.2 Pass Parameters to Script Using User Portion of SIP Request URI

Problem
Some SIP proxies remove parameters passed in the SIP request URI (that is, some SIP proxies strip “;ccxml=file:///script.ccxml” from the “sip:dialog@ms;ccxml=file:///script.ccxml”, resulting in a SIP request URI of “sip:dialog@ms”). In this case, how does the Media Server know which script to invoke and which parameters to pass to this script?

Solution
A workaround involves passing the name of the script and its parameters in the user portion of the SIP request URI. For example, sip:dialog_scriptname.ccxml_param1=2345_param2=helloworld@mediaserver. To help the Media Server parse the script name and its parameters, a default CCXML script runs on the Media Server. The default CCXML script parses the SIP request URI and invokes the CCXML script or VoiceXML script specified. The name and path to the default CCXML script is specified using the CLI at the `MS_CLI/Applications/MediaStreaming/Services/Dialog/DefaultXMLScript` level.

In pre-Version 17.0, this was configured from `MS_CLI/Service/DefaultXMLScript`.
The following code shows an example of a default CCXML script.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0" xmlns="http://www.w3.org/2002/09/ccxml">
  <var name="incomingCallId" expr="''" />
  <var name="dialogId" expr="''" />
  <var name="state" expr="'initial'" />
  <var name="scriptToRun" expr="''" />

  <eventprocessor statevariable="state">
    <transition state="initial" event="connection.alerting">
      <!-- Retrieve the name of the script to run -->
      <script>
        var scriptToRunRegex = new RegExp("^dialog_(.+)_");
        scriptToRun = scriptToRunRegex.exec(event$.info.sip.requesturi.user);
      </script>
      <if cond="scriptToRun">
        <!-- Found name of script to run; accept incoming call -->
        <accept />
        <assign name="incomingCallId" expr="event$.connectionid"/>
      </else/>
      <!-- Did not find name of script to run; rejecting call -->
      <reject />
      <log expr="'Rejecting call because the script name is missing in ' + event$.info.sip.requesturi.user"/>
      <exit/>
    </if>
  </transition>

  <transition state="initial" event="connection.connected">
    <log expr="'Call answered, starting script '+scriptToRun[1]'/">
    <assign name="state" expr="'connectionestablished'" />
    <dialogstart connectionid="incomingCallId" src="scriptToRun[1]"
      dialogid="dialogId" />
  </transition>

  <transition event="dialog.started">
    <log expr="'Dialog is running...'"/>
  </transition>

  <transition event="dialog.error">
    <log expr="'Error in dialog, exiting..'"/>
    <exit/>
  </transition>

  <transition event="connection.disconnected">
    <log expr="'Call has been disconnected. Ending CCXML Session.'"/>
    <exit/>
  </transition>

  <transition event="connection.failed">
    <log expr="'Connection '+event$.connectionid + ' failed, ending CCXML Session.'"/>
    <exit/>
  </transition>
</eventprocessor>
</ccxml>
```
WARNING: This method, which is used to invoke a script and pass parameters as part of the user portion of a SIP request URI, is not standard and should be used only for SIP proxies and application servers that cannot handle the method described in section 2.2.1 Specify Script in SIP INVITE.

7.1.3 Pass Parameters to CCXML Script Created Using <createccxml>

Problem
A CCXML script invokes another CCXML script using tag <createccxml>. How does the parent script pass parameters to the child script?

Solution
To pass parameters from a parent CCXML session to a newly created CCXML session, use the parameters attribute of the <createccxml> tag. The parameters appear in session variable "session.values", as described in the CCXML specification section 8.3.

The following two scripts illustrate this solution.

Parent.ccxml
```xml
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0" xmlns="http://www.w3.org/2002/09/ccxml">
  <var name="myparam1" expr="'123ABC'" />
  <var name="myparam2" expr="'ABC'" />
  <eventprocessor>
    <transition event="ccxml.loaded">
      <createccxml next="child.ccxml" parameters="myparam1 myparam2" />
    </transition>
  </eventprocessor>
</ccxml>
```

child.ccxml
```xml
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0" xmlns="http://www.w3.org/2002/09/ccxml">
  <eventprocessor>
    <transition event="ccxml.loaded">
      <log expr="'parameter values are ' + session.values.myparam1 + ' ' + session.values.myparam2" />
    </transition>
  </eventprocessor>
</ccxml>
```
7.2 Provide User ID/Password in HTTP Request

Problem
The Web Server from which you get a VoiceXML, CCXML script, or audio file is protected by a user ID and password. (When trying to access files, you get an “unauthorized” response.)

Solution
Include the credentials in the HTTP URL. For example:
http://userid:password@example.com/path/file.wav

7.3 VoiceXML Without Speech Recognition and Text-to-Speech

7.3.1 Use <menu> with Only DTMF Tones

Problem
Due to cost and scalability issues, adding Text-to-Speech and Speech Recognition servers to an Interactive Voice Response (IVR) application may not be appropriate. In this case, the VoiceXML <menu> tag should operate only in dual-tone multi-frequency (DTMF) mode.

Solution
Use the attribute `dtmf=true` of the <menu> tag to enable DTMF tones. Use <audio> tags to describe the menu options. Do not include <enumerate> in the <menu> tag. Do not include speech recognition keywords in the <choice> tag.

The following code shows a simple DTMF-only menu. The following DTMF tones are mapped to scripts:
- 1: askExtension.vxml
- 2: namedirectory.vxml
- 3: record.vxml
- 0: operator.vxml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
  <menu dtmf="true">
    <prompt>
      <audio src="audio_files/AAdefaultBusinessHoursGreeting.wav" />
    </prompt>
    <choice next="askExtension.vxml" />
    <choice next="namedirectory.vxml" />
    <choice next="record.vxml" />
    <choice dtmf="0" next="operator.vxml" />
  </menu>
</vxml>
```
7.4 Internationalize VoiceXML Scripts

7.4.1 Retrieve Language of VoiceXML Script

**Problem**
The VoiceXML application must retrieve the value of the xml:lang header.

**Solution**
The `optimtalk_info.doc_lang` property variable contains the value defined in the xml:lang header, for example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml"
Xml :lang="en-US">
<form>
<prompt>Language of this document is:
<value expr="session._optimtalk_info.doc_lang"/>
</prompt>
<prompt>A shorter way for the same:
<value expr="_optimtalk_info.doc_lang"/>
</prompt>
</form>
</vxml>
```

7.4.2 Write Multilingual VoiceXML Script That Uses Prerecorded Prompts

**Problem**
A single VoiceXML application, which is using pre-recorded prompts, must serve users in different languages.

**Solution**
Multilingual support can be achieved by using localized prompts whereby the language is passed as a parameter to the VoiceXML script (for more information, see section 7.1.1 Pass Parameters to Script in SIP Request URI Using RFC 5552). The following is an example of how this can be done assuming that the Media Server received a SIP INVITE with a SIP request URI containing:

sip:dialog@ms;voicexml=http://as/play.vxml;myLang=fr-CA

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
<script>
<![CDATA[
    function getLocalizedFileName (locale, name) {
        return "http://someAppServer/media/" + locale + "/" + name;
    }
]]>
</script>
<form id="Welcome">
 <block>
      <prompt>
        <audio expr="getLocalizedFileName('myLang',
        'welcome.wav')"/>
      </prompt>
     </block>
  </form>
</vxml>
```
7.4.3 Write Multilingual VoiceXML Script That Uses Speech Recognition

Problem
A single VoiceXML application, which is using speech recognition, must recognize input from users in different languages.

Solution
Grammar items can be specified with different languages, for example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xml version="2.0" xmlns="http://www.w3.org/2001/vxml">
<form>
  <field name="flavor">
    <prompt>What is your favorite flavor?</prompt>
    <help>Say one of vanilla, chocolate, or strawberry.</help>
    <grammar>
      <rule id="chocolate">
        <one-of>
          <item>chocolate</item>
          <item xml:lang="fr-CA">chocolat</item>
        </one-of>
      </rule>
      <rule id="vanilla">
        <one-of>
          <item>vanilla</item>
          <item xml:lang="fr-CA">vanille</item>
        </one-of>
      </rule>
      <rule id="strawberry">
        <one-of>
          <item>strawberry</item>
          <item xml:lang="fr-CA">fraise</item>
        </one-of>
      </rule>
    </grammar>
    </field>
  <filled>
    <if cond="flavor == 'fraise' || flavor == 'strawberry'">
      <prompt>I also like strawberry</prompt>
    </if>
    <elseif cond="flavor == 'vanille' || flavor == 'vanilla'">
      <prompt>I also like vanilla, but I prefer strawberry</prompt>
    </elseif>
    <elseif cond="flavor == 'chocolat' || flavor == 'chocolate'">
      <prompt>I also like chocolate, but I prefer strawberry</prompt>
    </elseif>
  </filled>
</form>
</xml>
```
7.5 Record Audio and Video in VoiceXML

7.5.1 Save WAV or MOV Files on External Web Server

**Problem**
A VoiceXML script must record an audio or video message (or greeting) and then save it on a remote Web Server.

**Solution**
The VoiceXML `<record>` is used to record a message (or greeting). After the recording is successful, the `<subdialog>` tag is used to transmit the message to the Web Server using an HTTP POST. (The `<submit>` can be used instead of `<subdialog>`; however, `<submit>` forces a transition to another VoiceXML script, while `<subdialog>` does not have this limitation. Using `<subdialog>` allows the returning of the control to the calling script through the `<return>` tag.)

On the Web Server, a Perl script is running to retrieve the message from the HTTP POST and write it to the disk.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml" version="2.1"
xmllns:xsi="http://www.w3.org/2001/XMLSchema-instance"
<!-- Compose a new message -->
<form id="ComposeMessage">

  <record name="newMessage" beep="true" maxtime="60s" dtmfterm="true">
    <prompt timeout="5s">
      <audio src="media/VMRecordNewMsg.wav" />
    </prompt>

    <noinput>
      <audio src="media/VMMessageTooShort.wav" />
    </noinput>
  </record>

<!-- Save media file onto HTTP server -->
<subdialog name="saveWav" src="saveMedia.pl" namelist="newMessage"
  method="post" enctype="multipart/form-data">
  <filled>
    <log label="ComposeMessage" expr="Response is '+saveWav.response'" />
    <if cond="saveWav.response == 'SUCCESS'">
      <log label="ComposeMessage" expr="Response is '+saveWav.response'" />
      <prompt>
        <audio src="media/VMRMsgSaved.wav" />
      </prompt>
    </if>
    <else/>
      <prompt>
        <audio src="media/TrtSystemUnavailable.wav" />
      </prompt>
    </else/>
  </filled>
</subdialog>

</form>
</vxml>
```
```perl
#!/usr/bin/perl
use strict;
use warnings;
use CGI;
use File::Temp qw(tempfile);

# Configuration
my $mediaDir = "/var/www/html/media/";

# Access HTTP request
my $query = new CGI;

# Retrieve reference to media stream
my $mediaStream;
$mediaStream = $query->param("newMessage")
    or die "Missing newMessage parameter in HTTP request";

# Is the media stream a WAV file or a MOV file?
my $firstLine = <$mediaStream>; # Read first line from media stream
my $fileExtension;
if ($firstLine =~ /^RIFF/) {
    $fileExtension=".wav";
} else {
    $fileExtension=".mov";
}

# Open WAV or MOV file
my ($mediaFileHandle, $mediaFilename) = tempfile("messageXXXXXX",
    SUFFIX => $fileExtension, DIR => $mediaDir )
    or die "Failed creating temp file: $!";

# Write file
print $mediaFileHandle $firstLine;
while (<$mediaStream>)
    {print $mediaFileHandle $_;
    }

# Tell VoiceXML script that recording is successful
sendResponse('SUCCESS');

# Build a VoiceXML response message
sub sendResponse
    {
        my $response = $_[0];
        print "Content-type: text/xml\n\n";
        print <<VOICEXML;
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
    version="2.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2001/vxml
    http://www.w3.org/TR/2007/REC-voicexml21-20070619/vxml.xsd">
<form>
    <block>
        <var name='response' expr="$response"/>
        <log expr="Response is $response"/>
        <return namelist='response'/>
    </block>
</form>
</vxml>
VOICEXML
    }
```

**saveMedia.pl** (a perl script running on the Web Server to receive the uploaded file)
7.5.2 Record Utterances During Speech Recognition

**Problem**

Speech recognition failure should be saved on an external file server to determine why the VoiceXML application was not able to handle the call successfully.

**Solution**

User utterances during speech recognition can be recorded when setting the `recordutterance` property to "true", for example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
     version="2.1"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.w3.org/2001/vxml
                      http://www.w3.org/TR/voicexml21/vxml.xsd">
  <form>
    <property name="recordutterance" value="true"/>
    <field name="city_state">
      <prompt>
        Say a city
      </prompt>
      <grammar>
        <rule id="available_city">
          <one-of>
            <item>Brno</item>
            <item>Montreal</item>
            <item>Gaithersburg</item>
          </one-of>
        </rule>
      </grammar>
      <nomatch>
        I'm sorry. I didn't get that.
        You said: <audio expr="application.lastresult$.recording" />
      </nomatch>
    </field>
  </form>
</vxml>
```
7.6 Script Communicating with External Servers

7.6.1 VoiceXML Script Communicating with External Server Using <data>

Problem
Occasionally, due to the changing nature of its application data, a VoiceXML application should be able to load the most recent data dynamically, without modifying the application.

Solution
Using <data> allows a VoiceXML application to retrieve XML data from a document server. The XML data is assigned to a read-only ecmaScript variable and can be accessed through a DOM parser. Different name spaces can be used to organize data in the XML data file, for example:

```
toronto.xml
<?xml version="1.0"?>
<?access-control allow="*"?>
<ac:flights xmlns:ac="http://aircanada.ca"
            xmlns:bw="http://bestwestern.com"
            xmlns:hl="http://hilton.com">
  <ac:flight>
    <ac:dest>YYZ</ac:dest>
    <ac:time>07:15</ac:time>
    <ac:price>139</ac:price>
    <bw:price>109</bw:price>
    <hl:price>119</hl:price>
  </ac:flight>
  <ac:flight>
    <ac:dest>YYZ</ac:dest>
    <ac:time>08:00</ac:time>
    <ac:price>129</ac:price>
    <bw:price>109</bw:price>
    <hl:price>119</hl:price>
  </ac:flight>
  <ac:flight>
    <ac:dest>YYZ</ac:dest>
    <ac:time>08:25</ac:time>
    <ac:price>109</ac:price>
    <bw:price>109</bw:price>
    <hl:price>119</hl:price>
  </ac:flight>
</ac:flights>
```
VoiceXML Script Performing Xtended Services Interface Operations

Problem
A VoiceXML script must perform Xtended Services Interface operations.

Solution
Xtended Services Interface/REST-API functionalities are available to VoiceXML application via the <data> element. The following four <data> methods can be used:

- Get: `<data name="result" method="get" ... />`
- Delete: `<data method="delete" ... />`
- Post: `<data method="post" enctype="application/xml" ... />`
- Put: `<data method="put" enctype="application/xml" ... />`
7.6.2.1 Use HTTP GET with Xtended Services Interface from VoiceXML Application

An Xtended Services Interface GET command is typically used to retrieve information from the BroadWorks Xtended Services Platform. The following example shows how to retrieve information from the Xtended Services Platform and how to use it from within a VoiceXML application.

```xml
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
<script><![CDATA[
function urlEscape(url) {
  var AT = new RegExp('@', 'g');
  return url.replace(AT, '%40');
}

function getSpeedDial100SpeedCodes(speedDial100Config) {
  var root = speedDial100Config.documentElement;
  var entries = root.getElementsByTagNameNS('http://schema.broadsoft.com/xsi', 'speedDial100Entry');
  var speedCodes = new Array();
  for(var i = 0; i < entries.length; i++) {
    var entry = entries.item(i);
    var speedCode = entry.getElementsByTagNameNS('http://schema.broadsoft.com/xsi', 'speedCode').item(0).firstChild.data;
    speedCodes[i] = speedCode;
  }
  return speedCodes;
}]]>
</script>

<var name="userId" expr="'south02@mtlasdev87.net'" />
<var name="urlReadyUserId" expr="urlEscape(userId)" />
<var name="passwd" expr="'mtllab'" />
<var name="xspAddress" expr="'192.168.13.21'" />

<form id="listSpeedDial100">
  <block>
    <log>Xsi::listSpeedDial100</log>
    <log expr="' User '+userId+ ' has the following speedDial100 speedCode:'"></log>
    <var name="url" expr="'https://'+urlReadyUserId+':'+passwd+'@'+xspAddress+ '/com.broadsoft.xsi-actions/v2.0/user/'+userId+ '/services/speeddial100'" />
    <data name="speedDial100Config" method="get" srcexpr="url" />
    <var name="speedCodes" expr="getSpeedDial100SpeedCodes(speedDial100Config)" />
    <foreach item="name" array="speedCodes">
      <log expr="' SpeedCode: ' + name"></log>
    </foreach>
  </block>
<form>
</vxml>
```
7.6.2.2 Use HTTP DELETE with Xtended Services Interface from VoiceXML Application

An Xtended Services Interface DELETE command is typically used to delete an entry on the BroadWorks Xtended Services Platform. The following example shows how to delete an entry on the Xtended Services Platform and how to use it from within a VoiceXML application.

```xml
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
<script><![CDATA[
    function urlEscape(url) {
        var AT = new RegExp("@", "g");
        return url.replace(AT, "%40");
    }
]]></script>

<var name="userId" expr="'south02@mtlasdev87.net'" />
<var name="urlReadyUserId" expr="urlEscape(userId)" />
<var name="passwd" expr="'mtllab'" />
<var name="xspAddress" expr="'192.168.13.21'" />

<form id="deleteSpeedDial100">
    <block>
        <log>Xsi::deleteSpeedDial100</log>
        <var name="speedCode" expr="'2'" />
        <var name="url"
            expr="'https://'+urlReadyUserId+':'+passwd+'@'+xspAddress+'/'+com.broadsoft.xsi-actions/v2.0/user/'+userId+'/services/speeddial100/'+speedCode" />
        <data method="delete"
            srcexpr="url" />
        <log expr="'Xsi::deleteSpeedDial100 Deleted:'+'speedCode'"/>
    </block>
</form>
</vxml>
```
7.6.2.3 Use HTTP POST with Xtended Services Interface from VoiceXML Application

An Xtended Services Interface POST command is typically used to add an entry on the BroadWorks Xtended Services Platform. The following example shows how to add an entry on the Xtended Services Platform and how to use it from within a VoiceXML application.

```xml
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">

<script><![CDATA[
    function urlEscape(url) {
        var AT = new RegExp("@", "g");
        return url.replace(AT, "%40");
    }

    function genSpeedDial100Entry(sp,pn,d) {
        return
            '<SpeedDial100 xmlns="http://schema.broadsoft.com/xsi">' + 
            '  <speedDial100Entry>' + 
            '    <speedCode>'+sp+'</speedCode>' + 
            '    <phoneNumber>'+pn+'</phoneNumber>' + 
            '    <description>'+d+'</description>' + 
            '  </speedDial100Entry>' + 
            '</SpeedDial100>'; 
    }
]]></script>

<var name="userId" expr="'south02@mtlasdev87.net'" />
<var name="urlReadyUserId" expr="urlEscape(userId)" />
<var name="passwd" expr="'mtllab'" />
<var name="xspAddress" expr="'192.168.13.21'" />
<form id="addSpeedDial100">
    <block>
        <log>Xsi::addSpeedDial100</log>
        <var name="url" expr="'https://'+urlReadyUserId+':'+passwd+'@'+xspAddress+ 
            '/com.broadsoft.xsi-actions/v2.0/user/'+userId+ 
            '/services/speeddial100'" />
        <var name="newSpeedDial100" expr="genSpeedDial100Entry(2,15145553456,"four")" />
        <data method="post" 
            enctype="application/xml" 
            namelist="newSpeedDial100" 
            srcexpr="url" />
        <log>Xsi::addSpeedDial100 Successfully added</log>
    </block>
    <log expr="newSpeedDial100"/>
</form>
</vxml>
```
7.6.2.4 Use HTTP PUT with Xtended Services Interface from VoiceXML Application

An Xtended Services Interface PUT command is typically used to modify an entry on the BroadWorks Xtended Services Platform. The following example shows how to modify an entry on the Xtended Services Platform and how to use it from within a VoiceXML application.

```xml
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
  <script><![CDATA[
function urlEscape(url) {
  var AT = new RegExp("@", "g");
  return url.replace(AT, "%40");
}

function genSpeedDial100Entry(sp, pn, d) {
  return
  '<SpeedDial100 xmlns="http://schema.broadsoft.com/xsi">' +
  '  <speedDial100Entry>' +
  '    <speedCode>'+sp+'</speedCode>' +
  '    <phoneNumber>'+pn+'</phoneNumber>' +
  '    <description>'+d+'</description>' +
  '  </speedDial100Entry>' +
  '</SpeedDial100>,'
}
]]></script>

<var name="userId" expr="'south02@mtlasdev87.net'" />
<var name="urlReadyUserId" expr="urlEscape(userId)" />
<var name="passwd" expr="'mtllab'" />
<var name="xspAddress" expr="'192.168.13.21'" />

<form id="modSpeedDial100">
  <block>
    <log>Xsi::modSpeedDial100</log>
    <var name="url"
      expr="'https://'+urlReadyUserId+':'+passwd+'@'+
          xspAddress+
          '/com.broadsoft.xsi-actions/v2.0/user/'+userId+
          '/services/speeddial100'"
    />
    <var name="modSpeedDial100"
      expr='genSpeedDial100Entry(2,15145553456,"four")'/
    <data method="put"
      enctype="application/xml"
      namelist="modSpeedDial100"
      srcexpr="url" />
    <log>Xsi::modSpeedDial100 Successfully modified</log>
  </block>
</form>
</vxml>
```
7.6.3 CCXML Script Posting Messages on Web Server

Problem

A CCXML script must collect a dialed number (for example, a calling card number) and then save it to a remote Web Server.

Solution

The dialed number can be extracted from event$.info.local_uri or event$.info.sip.headers.to, as shown in the following script. Afterwards, the CCXML <send> is used to transmit the message with collected data to the Web Server using an HTTP POST.

On the pre-Release 17.0 Media Server, the IP address of the destination Web Server must be in the access control list (ACL) configured using CLI at the MS_CLI/Service/Dialog/Security/AccessControlList level. Otherwise, “error.send.failed” is generated with the reason “ACL denied to dispatch event to the given address”. By default, the only allowed IP address is “localhost” (127.0.0.1).

`collect_number_and_post.ccxml`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0" xmlns="http://www.w3.org/2002/09/ccxml">
  <var name="currentState" expr="'initial'" />
  <var name="dialedUri" expr="'0'" />
  <eventprocessor statevariable="currentState">
    <transition state="initial" event="connection.alerting">
      <log expr="'In transition connection.alerting, received:'"/>
      <log expr="'event$.info.local_uri= ' + event$.info.local_uri"/>
      <log expr="'event$.info.sip.headers.to= ' + event$.info.sip.headers.to"/>
      <assign name="dialedUri" expr="event$.info.local_uri" />
      <accept/>
    </transition>
    <transition state="initial" event="connection.connected">
      <log expr="'Call has been answered. Send events to basic http processor.'"/>
      <assign name="currentState" expr="'answered'" />
      <!-- Send event to basichttp processor -->
      <send target="'http://127.0.0.1:4080?sessionid='+session.id" targettype="'basichttp" name="'dialedUri'" namelist="dialedUri"/>
    </transition>
    <transition state="answered" event="send.successful">
      <log expr="'Sent events to basic http processor successfully.'"/>
    </transition>
    <transition state="answered" event="connection.disconnected">
      <log expr="'Call has been disconnected. Ending CCXML Session.'"/>
      <exit/>
    </transition>
  </eventprocessor>
</ccxml>
```
7.6.4 CCXML Script Receiving Messages from Web Server

Problem
A CCXML script must receive an authorization from a remote web server to accept the incoming call based on the dialed number (for example, a calling card number).

Solution
The dialed number can be extracted from an event$.info.local_uri or event$.info.sip.headers.to, as shown in the following script. Afterwards, the CCXML <send> is used to transmit the message with collected data to the Web Server using an HTTP POST.

Note that Web Servers that are allowed access are controlled by the blacklist/white-list configuration found in MS_CLI/System/Security.

On the pre-Release 17.0 Media Server, Web Servers that are allowed access are controlled by the access control list (ACL) configured using the CLI at the MS_CLI/Service/Dialog/Security/AccessControlList level. By default, the only allowed IP address is “localhost” (127.0.0.1).

accept_conditionally.CCXML

Note that this script continues on the next page.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0" xmlns="http://www.w3.org/2002/09/ccxml">
<!-- This example tests basic http processor. -->
<!-- As per http://www.w3.org/TR/ccxml/#basichttpio. -->
<var name="currentState" expr="'initial'" />
<var name="dialedUri" expr="'0'" />
<eventprocessor statevariable="currentState">
  <transition state="initial" event="connection.alerting">
    <log expr="'In transition connection.alerting, received:'"/>
    <log expr="'event$.info.local_uri= ' + event$.info.local_uri" />
    <log expr="'event$.info.sip.headers.to= ' + event$.info.sip.headers.to" />
    <assign name="dialedUri" expr="event$.info.local_uri" />
    <!-- Send event to basichttp processor -->
    <send target="'http://127.0.0.1:4080?sessionid='+session.id"
      targettype="'basichttp'"
      name="dialedUri" namelist="dialedUri"/>
  </transition>
  <transition state="initial" event="send.successful" >
    <log expr="'Sent authorization request to basic http processor successfully.'"/>
  </transition>
  <transition state="initial" event="send.failed" >
    <log expr="'Sending events to external basic http processor failed.'"/>
    <reject/>
    <exit/>
  </transition>
  <!-- Process incoming basichttp event -->
  <transition state="initial" event="authorization" >
    <log expr="'Received event'" />
    <log expr="'name= ' + event$.name" />
    <log expr="'sourcetype= ' + event$.sourcetype" />
    <log expr="'eventsource= ' + event$.eventsource" />
    <log expr="'authorization = ' + event$.authorization" />
    <if cond="event$.authorization == 'yes'">
      <log expr="'Authorized. Accepting.'"/>
      <accept/>
    </if>
    <else/>
    <log expr="'Not authorized. Rejecting.'"/>
    <reject/>
  </transition>
</transition>  
</eventprocessor>
</ccxml>
```
7.6.5 Use CCXML Script Run at Startup to Receive HTTP POST from External Source

**Problem**
A default CCXML script, which is run when the Media Server starts up, must listen to the events posted by a remote Web Server.

**Solution**
Set the CLI parameter, `RunScriptOnStartup`, to the desired script using the CLI at the `MS_CLI/Applications/MediaStreaming/Services/Dialog` level or `MS_CLI/Service/Dialog` for the pre-Release 17.0 system. A CCXML `<send>` is used to transmit the sessionid associated with the running on Media Server CCXML script to the Web Server using an HTTP POST.

Note that accesses to the Web Server are controlled by the blacklist/white-list configuration found in `MS_CLI/System/Security`.

On the pre-Release 17.0 Media Server, the IP address of the destination (Web Server) must be in the access control list (ACL), which is configured using the CLI at the `MS_CLI/Service/Dialog/Security/AccessControlList` level. Otherwise, an “error.send.failed” is generated with the reason “ACL denied to dispatch event to the given address”. By default, the only allowed IP address is “localhost” (127.0.0.1).

The Web Server then uses the received `sessionid` in its HTTP POST request to send an event to the CCXML script. The following is an example of this type of script.

```
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0" xmlns="http://www.w3.org/2002/09/ccxml">
  <!-- This example tests basic http processor. -->
  <!-- As per http://www.w3.org/TR/ccxml/#basichttpio. -->
  <var name="currentState" expr="'initial'" />
  <eventprocessor statevariable="currentState">
    <transition state="initial" event="ccxml.loaded">
      <log expr="'CCXML script created. sessionid=' + session.id'/">
      <log expr="'\tevent$.sessionid=' + event$.sessionid'/">
    </transition>
    <transition state="initial" event="send.successful">
      <log expr="'Sent authorization request to basic http processor successfully.'"/>
    </transition>
    <transition state="initial" event="send.failed">
      <log expr="'Sending events to external basic http processor failed. Ending CCXML Session.'"/>
      <reject/>
      <exit/>
    </transition>
    <!-- Process incoming basichttp event -->
  </eventprocessor>
</ccxml>
```
7.7 Conferences

7.7.1 Broadcast Conference to Many Listeners (Hundreds)

Problem
Large phone conferences are typically composed of a small group of active participants (that is, people who talk) and a large group of participants who only listen to the conference. How can this behavior be implemented?

Solution
Create a conference using `<createconference>` and set attribute `reservedlisteners` to a non-zero value. When joining a participant to the conference, indicate whether the participant is listening only by setting “duplex='half’” in the `<join>` tag.

The following example establishes a conference between an incoming call and an outgoing call. The outgoing call can only listen to the conference.

Note that this script continues on the next page.
7.7.2 Upgrade Conference Listener to Talker Status

Problem
A conference participant who is only listening to a conference may want to be upgraded to talker status (for example, to ask a question).

Solution
The general problem of identifying how a conference participant signals to the conference controller that they want to speak is left to the developer. This solution addresses the low-level instructions to upgrade a conference listener to talker status. In summary:

```xml
<unjoin id1="participantConnectionId" id2="ConfId" />
<join id1="participantConnectionId" id2="ConfId" duplex="full" />
```

Note that the duplex attribute in <join> is optional because full duplex is the default value.

Similarly, to transform a talker into a listener:

```xml
<unjoin id1="participantConnectionId" id2="ConfId" />
<join id1="participantConnectionId" id2="ConfId" duplex="half" />
```
7.8 Call Control

7.8.1 Accept Incoming Call and Connect it to Outgoing Call

Problem
A user dials an 800 number. Call Control server must replace this fixed 800 number with the "actual" phone number and must stay in the media path between the user and final destination.

Solution
The SIP proxy should have the 800-number associated with a CCXML script, such as shown the following example. SIP proxy then forwards the incoming call to the Media Server running this CCXML script. The CCXML script accepts the incoming call and originates a new call to the "real number" using <createcall/> tag with joinid parameter set to the incoming call ID. When the second call is answered, the script joins the incoming and outgoing calls in full duplex mode (default).

The following is an example of this script.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0" xmlns="http://www.w3.org/2002/09/ccxml">
  <var name="incomingCallConnId" expr="''" />
  <var name="outgoingCallConnId" expr="''" />
  <var name="currentState" expr="'initial'" />

  <eventprocessor statevariable="currentState">
    <transition state="initial" event="connection.alerting">
      <log expr="'We like you! We are going to answer the call.'"/>
      <assign name="incomingCallConnId" expr="event$.connectionid" />
      <accept/>
    </transition>

    <transition state="initial" event="connection.connected">
      <log expr="'Incoming call is answered; originating call to other party.'"/>
      <createcall dest="tel:6000" connectionid="outgoingCallConnId" joinid="incomingCallConnId" />
      <assign name="currentState" expr="'originatingcall'" />
    </transition>

    <transition state="originatingcall" event="connection.connected">
      <log expr="'Outgoing call answered; joining it to incoming call.'"/>
      <assign name="currentState" expr="'twowayspeechpath'" />
    </transition>

    <transition event="connection.disconnected">
      <log expr="'Call has been disconnected. Ending CCXML Session.'"/>
    </transition>
  
    <transition event="connection.failed">
      <log expr="'Connection ' + event$.connectionid + ' failed, ending CCXML Session.'"/>
    </transition>
  </eventprocessor>
</ccxml>
```
7.8.2 Redirect Incoming Call Before Answer

**Problem**
A user dials an 800 number. The Call Control server must replace this fixed 800 number with the “actual” phone number.

**Solution**
The SIP proxy should have the 800 number associated with a CCXML script, such as shown in the following example. The SIP proxy then forwards the incoming call to the Media Server running this CCXML script. The CCXML script immediately redirects the incoming call with the SIP response “302 Moved Temporarily” to the “real number” using `<redirect/>` tag with `dest` parameter set to the target number.

The following is an example of this script.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0" xmlns="http://www.w3.org/2002/09/ccxml">
<eventprocessor>
  <transition event="connection.alerting">
    <log expr=""Received a call, redirecting it""/>
    <redirect dest="tel:1403"/>
  </transition>
</eventprocessor>
</ccxml>
```

7.8.3 Blind Transfer Call

**Problem**
Blind transfer can be used in applications, such as for an Auto Attendant.

**Solution**
Both CCXML and VoiceXML scripts can be used to perform a blind transfer. An example of an Auto Attendant VoiceXML script performing a blind transfer (extension dialing) is shown the following example. In this case, the `<transfer/>` tag with the parameter bridge set to “true” is used to perform the blind transfer to extension number.

`autoattendant.vxml`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  <var name="transferToParty" />
  <form id="askExtension">
    <field name="requestedExtension">
      <grammar type="application/sgrs+xml" src="dtmffourdigits.grxml"/>
      <prompt>
        <audio src="audio_files/AAdefaultPromptForExtension.wav">
          Please dial the four digit extension of the party you want to reach.
        </audio>
      </prompt>
      <noinput>
        <log label="autoattendant - getextension">noinput</log>
        <prompt>
          <audio src="audio_files/AAdefaultRepromptForExtension.wav"/>
        </prompt>
        <noinput>
          <filled>
            <log label="autoattendant - getextension">
```

```xml
</filled>
</noinput>
</form>
</vxml>
```
An example of a CCXML script performing blind transfer is shown in the following example. In this case, the incoming call is being accepted and then the <redirect/> tag with parameter dest equal to the transfer destination number is used to perform the blind transfer. This makes the CCXML platform (Media Server) issue the SIP REFER to transfer the incoming call to the destination.

**accept_and_blindtransfer.CCXML**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0" xmlns="http://www.w3.org/2002/09/ccxml">
<eventprocessor>
  <transition event="connection.alerting">
    <log expr=""We like you! We are going to answer the call.""/>
    <accept/>
  </transition>
</eventprocessor>
</ccxml>
```
7.8.4 Bridge Transfer of Call Using VoiceXML

**Problem**

A user dials an 800 number. The Call Control server must replace this fixed 800 number with the "actual" phone number and must stay in the media path between the user and final destination.

**Solution**

The SIP proxy should have the 800 number associated with a VoiceXML script, such as shown in the following example. The SIP proxy then forwards the incoming call to the Media Server running this VoiceXML script. The VoiceXML script accepts the incoming call and originates a new call to the "actual number" using the `<transfer/>` tag with `destexpr` parameter set to the target number. When the second call is answered, the script bridges the incoming and outgoing calls and sets up a two-way media path.

The following is the example of this script.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
  <var name="transferToParty" expr="'1401'"/>
  <form id="transfer">
    <block>
      <log label="autoattendant - transfer">Initiating transfer to <value expr="'tel:1401'" /></log>
    </block>
    <transfer name="transferresult" connecttimeout="30s" destexpr="transferToParty" bridge="true">
      <filled>
        <if cond="transferresult=='busy' || transferresult=='network_busy' ||
             transferresult=='unknown'">
          <log label="autoattendant - transfer">transfer result is busy or unknown</log>
          <prompt>Sorry, transfer is currently unavailable. Please try again later.</prompt>
        </if>
      </filled>
    </transfer>
    <catch event="error.unsupported.transfer.blind">
      <log label="autoattendant - transfer">blind transfer is not supported</log>
      <prompt>Sorry, transfer is not available on this system</prompt>
      <exit expr="'error.unsupported.transfer.blind'"/>
    </catch>
    <catch event="error.connection.baddestination">
      <log label="autoattendant - transfer">Wrong number</log>
      <prompt>You have entered a wrong number</prompt>
      <exit expr="'error.connection.baddestination'"/>
    </catch>
    <catch event="error.connection.noresource">
      ...
    </catch>
  </form>
</vxml>
```
7.8.5 Transfer of Call from VoiceXML Script Invoked from CCXML Session

Problem
What events should a CCXML session handle when a VoiceXML session created using `<startdialog>`, must perform a call transfer?

Solution
The CCXML session and its VoiceXML dialog communicate through events to handle a call transfer. When a VoiceXML dialog must perform a call transfer, it simply uses the `<transfer>` tag. This automatically triggers a `dialog.transfer` event to be sent to the CCXML session.

In the `dialog.transfer` transition, the CCXML session must redirect the call if it is a blind transfer. The dialog is then notified with a `connection.disconnect.transfer` event sent by the CCXML session using the `<send>` tag. No additional event handling is required since neither the CCXML session nor the VoiceXML session is involved in this transfer.

For a bridge transfer, the CCXML session must initiate a `<createcall>` to the destination. If the `<createcall>` fails, then it must notify the dialog (that is, the VoiceXML session) about `connection.failure` event by issuing a `<send>` `dialog.transfer.complete` with the actual failure reason. For success, the CCXML session has to `<unjoin>` the originating caller from the dialog before doing a `<join>` with the transferred party.

The following is an example of a bridge transfer based on Appendix D.9 and D.11 of CCXML 1.0:

**Calltransfer.vxml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
  <var name="transferToParty" />

  <form id="transfer">
    <block>
      <assign name="transferToParty" expr="'tel:5015'" />
      <log label="calltransfer.vxml">Initiating transfer to <value expr="transferToParty" /></log>
      <value expr="transferToParty" />
    </block>

    <transfer name="transferresult" connecttimeout="30s" destexpr="transferToParty" bridge="true">
      <filled>
        <if cond="transferresult=='busy' || transferresult=='network_busy' || transferresult=='unknown'">
          <log label="calltransfer.vxml">transfer result is busy or unknown</log>
          <prompt>Sorry, transfer is currently unavailable. Please try again later.</prompt>
        </if>
      </filled>
    </transfer>
  </form>
</vxml>
```
<exit expr="'error.unsupported.transfer.blind'"/>
</catch>
<catch event="error.connection.baddestination">
<log label="calltransfer.vxml">Wrong number</log>
<prompt>You have entered a wrong number</prompt>
<exit expr="'error.connection.baddestination'"/>
</catch>
<catch event="error.connection.noresource">
<log label="calltransfer.vxml">Network busy</log>
<prompt>Sorry, transfer is currently unavailable. Please try again later.</prompt>
<exit expr="'error.connection.noresource'"/>
</catch>
</form>
</vxml>

**ccxmlCalltransfer.CCXML**

Note that this script continues on the following pages.
<!-- Just send the success event to the dialog -->
<send name="'connection.disconnect.transfer'"
     target="dialogId"
     targettype="'dialog'/">

<!-- Bridge == true. In this case we need to
   place a call and bridge the calls -->

<!-- save off maxtime -->
<assign name="vxml_maxtime" expr="event$.maxtime"/>

<!-- Update our state var -->
<assign name="currentState" expr="'calling'"/>

<!-- Place the call using the values from the transfer request -->
<assign name="URI" expr="event$.URI"/>
<log expr="'calling createCall with uri: '+event$.URI"/>
<createcall dest="event$.URI"
        connectionid="outgoingCallConnId"
        aai="event$.aai"
        timeout="event$.connecttimeout"/>
</if>
</transition>

<!-- We will get the following events but we do not do anything
    because in VoiceXML 2.0 you just ignore redirect errors.
    We do however process the dialog.exit and shutdown
    the CCXML Session. -->
<transition state="redirecting" event="connection.redirected">
</transition>
<transition state="redirecting" event="connection.failed">
</transition>
<transition state="redirecting" event="dialog.exit">
<exit/>
</transition>

<!-- Handle bridge=true Events
   This first event is for if the outbound call failed.

<transition state="calling" event="connection.failed">
<!-- Just send the error event to the dialog -->
<send name="'dialog.transfer.complete'
        target="dialogId"
        targettype="'dialog'"
        namelist="results"/>

<!-- Update our state var back to the original state -->
<assign name="currentState" expr="'dialogActive'"/>
</transition>

<!-- The outbound call has been answered. -->
<transition state="calling" event="connection.connected">
<!-- Update our state var back to show that we are connected -->
<assign name="currentState" expr="'outgoing_call_active'"/>
<!-- Unjoin the calls before it can be connected to other call -->
<unjoin id1="incomingCallConnId" id2="dialogId"/>
</transition>
<!-- Now connect the outbound. -->
<transition state="outgoing_call_active" event="conference.unjoined">
<!-- Join the two calls together -->
<join id1="incomingCallConnId" id2="outgoingCallConnId" duplex="full"/>
</transition>

<!-- We will get here once the join completes. -->
<transition state="outgoing_call_active" event="conference.joined">
<!-- If maxtime has been set then we setup a timer -->
<if cond="vxml_maxtime != null">
<send name="vxml_maxtime" target="session.id" delay="vxml_maxtime" sendid="maxtime_sendid"/>
</if>
</transition>

<!-- Deal with someone disconnecting. -->
<transition state="outgoing_call_active" event="connection.disconnected">
<!-- Cancel any maxtime events that are waiting to be fired -->
<if cond="maxtime_sendid != null">
<cancel sendid="maxtime_sendid" />
<assign name="maxtime_sendid" expr="null"/>
</if>
<!-- Branch off based on what call leg this is for and send the proper event to the dialog -->
<if cond="event$.connectionid == outgoingCallConnId">
<assign name="results" expr="far_end_disconnect" />
<send name="dialog.transfer.complete" target="dialogId" targettype="dialog" namelist="results" />
</if>
<!-- Update our state var back to the original state -->
<assign name="currentState" expr="dialogActive" />
</if>
</transition>

<transition event="connection.disconnected">
<log expr="Call has been disconnected. Ending CCXML Session." />
<exit />
</transition>

<transition event="connection.failed">
<log expr="Connection 'event$.connectionid + ' failed, ending CCXML Session." />
<exit />
</transition>

7.8.6 Set Optional SIP Headers in SIP INVITE, SIP 302, and SIP REFER

Problem
When a CCXML script uses `<createcall>` to originate a new call or `<redirect>` to transfer a call, the script may want to set optional SIP headers in SIP messages transmitted by the Media Server.

Note that the Media Server does NOT perform any checks on the syntax of optional SIP headers passed in “hints”. It is the responsibility of the developer of the script to make sure that optional SIP headers are valid.

Solution
Attribute hints present in `<createcall>` and `<redirect>` are used to add optional SIP headers to a SIP INVITE, SIP 302 response, or SIP REFER.

The following is an excerpt from a CCXML script that sets the optional `Diversion` and `P-Asserted-Identity` headers in an outgoing SIP INVITE.

```xml
<script><![CDATA[
  var hints = { sipHeaders: 'Diversion: <sip:BWUser@AS>; reason=unknown; counter=15;privacy-off;diversion-inhibited
  P-Asserted-Identity: sip:ms@broadsoft.com' };
]]></script>
```

<!-- SIP INVITE will include Diversion and P-Asserted-Identity headers -->
`<createcall dest="tel:8194246000" connectionid="connId" hints="hints" />

<!-- SIP 302/REFER will include Diversion and P-Asserted-Identity headers -->
`<redirect dest="sip:5010@192.168.12.46" hints="hints" />
```

7.8.7 Originate Call from CCXML Session Created Using `<createccxml>`

Problem
When a CCXML session is created using `<createccxml>` instead of being created by a call terminating to the Media Server, the SIP session variables are not set in the CCXML session. When it is time to originate a call from a session created by `<createccxml>`, the Media Server does not know which SIP proxy to send SIP messages to and which SIP From: address to use.

Solution
Two solutions exist for this problem:

- Use SIP: URIs in the attributes `dest` and `caller` of `<createcall>`. This way, the Media Server has the information it needs to originate a call and the Media Server does not have to rely on the (empty) SIP session variables.

- Provide a default SIP proxy and SIP From: address in using the CLI `MS_CLI/Applications/MediaStreaming/Services/Dialog level` or `MS_CLI/Service/Dialog` for the pre-Release 17.0 system. Then, you can use “tel:” URIs and the Media Server can transform them into a valid SIP address and SIP proxy.
The following example shows a script that invokes another script using `<createccxml>`, and the later script makes an outgoing call using `<createcall>` with SIP URIs.

**startwakes.ccxml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0" xmlns="http://www.w3.org/2002/09/ccxml">
  <var name="currentState" expr="'initial'" />
  <eventprocessor statevariable="currentState">
    <transition state="initial" event="ccxml.loaded">
      <assign name="currentState" expr="'firstwakeup'" />
      <createccxml next="" />
      <createccxml next="" />
    </transition>
    <transition state="firstwakeup" event="ccxml.created">
      <assign name="currentState" expr="'secondwakeup'" />
      <var name="destination" expr="'sip:8194246000@lin.mtl.broadsoft.com'" />
      <var name="caller" expr="'sip:8194246006@lin.mtl.broadsoft.com'" />
      <send target="event$.sessionid" targettype="ccxml" name="call.phonenumber" namelist="destination caller" />
    </transition>
    <transition state="secondwakeup" event="ccxml.created">
      <assign name="currentState" expr="'done'" />
      <var name="destination" expr="'sip:8194246003@lin.mtl.broadsoft.com'" />
      <var name="caller" expr="'sip:8194246006@lin.mtl.broadsoft.com'" />
      <send target="event$.sessionid" targettype="ccxml" name="call.phonenumber" namelist="destination caller" />
    </transition>
  </eventprocessor>
</ccxml>
```

**wakeup.ccxml**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ccxml version="1.0" xmlns="http://www.w3.org/2002/09/ccxml">
  <var name="outgoingCallConnId" expr="''" />
  <var name="currentState" expr="'initial'" />
  <eventprocessor statevariable="currentState">
    <transition event="call.phonenumber">
      <assign name="currentState" expr="'originatingcall'" />
      <createcall dest="event$.destination" callerid="event$.caller" connectionid="outgoingCallConnId" />
    </transition>
    <transition state="originatingcall" event="connection.connected">
      <!-- Call is connected, do something such as start a dialog -->
      <assign name="currentState" expr="'twowayspeechpath'" />
    </transition>
    <transition state="twowayspeechpath" event="connection.disconnected">
      <!-- Call has been disconnected. End CCXML Session. -->
      <exit/>
    </transition>
    <transition state="twowayspeechpath" event="connection.failed">
      <!-- Connection failed, ending CCXML Session. -->
      <exit/>
    </transition>
  </eventprocessor>
</ccxml>
```
7.9 Collect Digits

7.9.1 Collect Phone Number

Problem
The Media Server does not provide built-in grammars as defined in VoiceXML 2.0 Appendix P. To collect digits, a DTMF grammar must be manually defined.

Solution
The following DTMF grammar can be easily imported into a VoiceXML document or cloned into your own document and modified as required.

dtmffourdigits.grxml

```xml
<?xml version="1.0"?>
<grammar mode="dtmf" version="1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xmlns="http://www.w3.org/2001/06/grammar"
root="#fourdigits"/>

<rule id="digit">
  <one-of>
    <item> 0 </item>
    <item> 1 </item>
    <item> 2 </item>
    <item> 3 </item>
    <item> 4 </item>
    <item> 5 </item>
    <item> 6 </item>
    <item> 7 </item>
    <item> 8 </item>
    <item> 9 </item>
  </one-of>
</rule>

<rule id="fourdigits" scope="public" >
  <one-of>
    <item repeat="4"><ruleref uri="#digit"/></item>
    <item> * </item>
  </one-of>
</rule>
</grammar>
```

The following is an example of a VoiceXML script that imports the grammar shown previously.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
  <var name="phoneNumber" />
  <form id="askExtension">
    <field name="requestedExtension">
      <grammar mode="dtmf" src="dtmffourdigits.grxml" type="application/srgs+xml"/>
      <prompt>
        <audio src="audio_files/AAdefaultPromptForExtension.wav"/>
      </prompt>
    </field>
  </form>
</vxml>
```
7.10 Play Video Clips

7.10.1 Play Video Clips with Multiple Video Codecs

Problem
Different video phones support different video codecs. To reach the maximum number of potential customers, how can a VoiceXML script support as many different video codecs as possible?

Solution
The Media Server does not perform video transcoding. For a VoiceXML script to support multiple video codecs simultaneously, the video clips must be recorded in different video codecs. In the VoiceXML script, include all versions of the video clip that are available. The Media Server plays the version of the video clip that is compatible with the video phone that is currently connected.

The following script shows an example of multiple video codecs.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
  <menu dtmf="true">
    <prompt>
      <!-- Try to play the H.264 version of the video clip -->
      <audio src="media/h264/AAdefaultBusinessHoursGreeting.mov">
        <!-- If H.264 fails, try to play the H.263 version of the clip -->
        <audio src="media/h263/AAdefaultBusinessHoursGreeting.mov">
          <!-- Play audio only if everything else fails -->
          <audio src="media/AAdefaultBusinessHoursGreeting.wav"/>
        </audio>
      </audio>
    </prompt>
    <choice next="askExtension.vxml" />
    <choice next="namedirectory.vxml" />
    <choice next="record.vxml" />
    <choice dtmf="0" next="operator.vxml" />
  </menu>
</vxml>
```

Note that this technique was recommended by Hewlett-Packard and VoxPilot in Video Interactive Services with VoiceXML, March 2006 [4].
7.10.2 Record Video Clips

Problem
How do you record a video clip in a VoiceXML script?

Solution
In this case, the solution described in section 7.5 Record Audio and Video in VoiceXML applies. Note that the video clip is recorded in a single video codec. Only video phones that use that same codec are able to view the video clip.

7.11 VCR-type Controls in VoiceXML Scripts

Problem
Some applications such as Voice Mail offer the capability to pause, fast forward, and rewind a message while listening to it. The VoiceXML 2.0 and 2.1 standards do not offer these VCR-type controls.

Solution
A BroadSoft proprietary extension to VoiceXML allows VCR-type controls. The `offsetexpr` attribute is added to the `<prompt>` tag as follows. The `offsetexpr` evaluates the offset (in milliseconds) from the beginning of the media file where playback would start.

The following is an example of a VoiceXML script implementing the VCR-type controls. Note that this script continues on the next page.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
  <!--
    DTMF 4 = backward
    DTMF 6 = forward
    DTMF 5 = pause/resume
    -->
  <var name="jumpsize" expr="5000"/> <!-- 5s jumps -->
  <form>
    <var name="offset" expr="0"/>
    <var name="paused" expr="false"/>
    <field name="key">
      <property name="termchar" value=""/>
      <noinput>
        <log>Playback finished</log>
        <disconnect/>
      </noinput>
      <prompt timeout="0s" cond="!paused">
        <mark name="start"/>
        <audio src="vcr_test.wav" offsetexpr="offset+'ms'"/>
      </prompt>
      <grammar root="num" mode="dtmf">
        <rule id="num">
          <one-of>
            <item>4</item>
            <item>5</item>
            <item>6</item>
          </one-of>
        </rule>
      </grammar>
      <filled>
        <if cond="typeof(key$.markname) == 'string' &amp;&amp; (key$.markname=='start')">
          <if cond="key == '4'">
            <script><![CDATA[
              offset = offset + key$.marktime - jumpsize; // backward
              if (offset < 0) offset = 0;
            ]]]></script>
            <log>jump forward to offset: <value expr="offset"></log>
          </elseif>
          <if cond="key == '6'">
            <script><![CDATA[
              offset = offset + key$.marktime + jumpsize; // forward
            ]]]></script>
            <log>jump backward to offset: <value expr="offset"></log>
          </elseif>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```
7.12 Use Older April 2003 Semantic Interpretation for Speech Recognition

Problem
This version of the document includes the following changes.
After a Media Server upgrade, the Media Server raises error.semantic events when parsing Semantic Interpretation for Speech Recognition (SISR) grammar elements.

Solution
Since Release 16.sp2 and Release 17.sp1, the Media Server VoiceXML interpreter supports the April 2007 SISR recommendation, whereas the earlier version only supported the April 2003 SISR draft. Names of variables assigned to the nodes of the grammar parse tree changed between these documents (“$” is “out” now, “$” is rules[], and so on).

AP117442 also brought the updated Media Server VoiceXML interpreter to Release 16.0, Release 16.sp1, and Release 17.0.

If a SISR grammar specifies <grammar tag-format="semantics/1.0" ... >, then the VoiceXML interpreter uses the April 2007 SISR recommendation. This is the default behavior when no tag-format attribute is specified.

However, if the older April 2003 SISR semantic is required, then the VoiceXML application should specify <grammar tag-format="semantics/1.0.2003" ... >.
A SOAP client is integrated into the VoiceXML interpreter to provide tighter integration with external servers. This section describes how to use this Simple Object Access Protocol (SOAP) client to build a simple script.

Using the SOAP client
The SOAP client is driven through the VoiceXML 2.1 tag <data>.

In addition, two objects for SOAP message manipulation have been created and are accessible in the session scope of the VoiceXML document.

- SoapRequest simplifies SOAP request creation by automating the process of crafting well-formed SOAP messages from supplied message bodies.
- SoapResponse exposes DOM of received response and has methods for error as well as correct response retrieval.

The following sections show how to use <data>, SoapRequest, and SoapResponse within a VoiceXML script to access a SOAP server.

Simple SOAP service
A simple SOAP service is available on a SOAP server, doubleAnInteger. This service takes an integer and returns its value multiplied by 2. The request message, response message, and fault message follow:

SOAP request
This message requests execution of the procedure, doubleAnInteger, with parameter 123.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:doubleAnInteger xmlns:ns1="urn:MySoapServices">
      <param1 xsi:type="xsd:int">123</param1>
    </ns1:doubleAnInteger>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP response
This message is a response to the request above and returns a result with value “246”.

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<SOAP-ENV:Envelope
```

Appendix A: SOAP Client
Fault message

This message is a response to the request above and indicates that there was a problem fulfilling the request because the supplied parameter was out of range.

Example VoiceXML Script

The VoiceXML script that follows shows how to use the SOAP server that uses the messages defined above. It creates a SOAP request, sends it to a web service located at www.example.com/soap.jsp, and handles its response.
The following sections describe common scenarios that occur when this example VoiceXML script is run.

**Scenario 1: SOAP request is sent and a valid response is received.**

1) Create SOAP request.

   Lines 5 through 8: New SoapRequest object is created and initialized with the setRequest() method by supplying the message body.

2) Send the request to the server.

   Lines 12 and 13: SOAP request is sent using the VoiceXML data tag. The request destination URL is specified in the src attribute of the data tag.

   It is necessary to set the method attribute to "soap". This way the HTTP headers are constructed in accordance with the SOAP specification. It also forces a different namelist handling, that is, exactly one variable in namelist is allowed and only its value is submitted as opposed to the usual name=value construction.
The well-formed SOAP request message is stored in the SoapRequest.soapMessage member, which is listed as the only variable in the namelist attribute. As the variable expected in namelist is a string containing a SOAP message, it is not necessary to use the SoapRequest object at all. Users can provide another variable containing a well-formed SOAP request message. SoapRequest is just for convenience.

3) Receive the response.

Lines 14 through 19: Variable that was passed to the data tag in the name attribute contains the received SOAP response message. This message is already parsed and exposes DOM.

4) Access the result.

Line 28: Method getResponseBody() returns the <SOAP-ENV:Body> element. The desired result is a child of this element and can be accessed via DOM API.

Scenario 2: SOAP Request is successfully sent and a fault message is received.
Steps 1 and 2 are the same as in the previous scenario. This description covers lines 14 through 25.

3) Receive the response.

Line 14: Variable that was passed to the data tag in the name attribute contains the received SOAP fault message. This message is already parsed and exposes DOM.

4) Obtain the error.

Line 21: Using the isFaultMsg() method, one can find out whether or not the response was faulty.

Line 24: The error description can be accessed using method getErrorString().

Scenario 3: HTTP error.
An HTTP error usually occurs due to network outage and results in an error.badfetch event being thrown by the VoiceXML interpreter.

Lines 32 through 34: Catch the error.badfetch event and handle the error.

Limitations of the embedded SOAP client

<table>
<thead>
<tr>
<th>R&amp;L #</th>
<th>Feature</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SOAP headers.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>2</td>
<td>SOAP Message Transmission Optimization Mechanism.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>3</td>
<td>XML-binary Optimized Packaging.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>4</td>
<td>Extended HTTP.</td>
<td>Not supported.</td>
</tr>
<tr>
<td>5</td>
<td>Request-response message exchange pattern.</td>
<td>Only synchronous transfer and POST method are supported.</td>
</tr>
<tr>
<td>6</td>
<td>HTTP 302 in response to POST.</td>
<td>Causes an error.badfetch. Only HTTP 307 response is supported.</td>
</tr>
</tbody>
</table>

Table 11 Limitations of Embedded Soap Client
## Acronyms and Abbreviations

This section lists the acronyms and abbreviations found in this document. The acronyms and abbreviations are listed in alphabetical order along with their meanings.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>Admin</td>
<td>Administrator</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>AS</td>
<td>Application Server</td>
</tr>
<tr>
<td>BW</td>
<td>BroadWorks</td>
</tr>
<tr>
<td>CC</td>
<td>Call Control</td>
</tr>
<tr>
<td>CLI</td>
<td>Command Line Interface</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DTMF</td>
<td>Dual-Tone Multi-Frequency</td>
</tr>
<tr>
<td>FQDN</td>
<td>Fully Qualified Domain Name</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IVR</td>
<td>Interactive Voice Response</td>
</tr>
<tr>
<td>MGCP</td>
<td>Media Gateway Control Protocol</td>
</tr>
<tr>
<td>MRF</td>
<td>Media Resource Function</td>
</tr>
<tr>
<td>MS</td>
<td>Media Server</td>
</tr>
<tr>
<td>NS</td>
<td>Network Server</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PM</td>
<td>Performance Measurement</td>
</tr>
<tr>
<td>POP</td>
<td>Point Of Presence</td>
</tr>
<tr>
<td>SIP</td>
<td>Session Initiation Protocol</td>
</tr>
<tr>
<td>SISR</td>
<td>Semantic Interpretation for Speech Recognition</td>
</tr>
<tr>
<td>SRV</td>
<td>Service Locator</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
References


Index

Abbreviations, 76
Acronyms, 76
April 2003 SISR, 71
Audio, recording in VoiceXML, 44
Call Control, 58
CCXML, 11
  Cookbook, 38
  Invoking script, 26
  Logs, 28
  Script originating call, 34
  Standards compliancy, 23
Collecting Digits, 68
Compliance
   CCXML, 23
   File formats, 15
   Redundancy, 30
   VoiceXML, 14
Conferences, 56
Cookbook
   Call Control, 58
   Collecting Digits, 68
   Conferences, 56
   Internationalizing VoiceXML scripts, 42
   Playing video clips, 69
   Recording Audio and Video in VoiceXML, 44
   Script communicating with external servers, 47
   Starting scripts, 38
   Starting scripts, VoiceXML and CCXML, 38
   Use older April 2003 SISR, 71
   User ID/Password in HTTP request, 41
   VCR controls in VoiceXML scripts, 70
   VoiceXML without speech recognition and Text-to-Speech, 41
DNS SRV lookup, 30
External database, 13
External servers, script communicating, 47
Internationalizing
   VoiceXML, 21
   VoiceXML scripts, 42
Licensing, 36
Logs
   CCXML, 28
   VoiceXML, 19
Media Server, 12
Media Server Selection policy, Network Server, 32
Mime-types, 35
Platform, VoiceXML, 18
Redundancy
   CCXML script originating call, 34
   DNS SRV lookup, 30
   Loading VoiceXML or CCXML script, 33
   Media Server Selection policy, Network Server, 32
   Standards compliancy, 30
Script
   Communicating, external servers, 47
   Invoking CCXML, 26
Security, VoiceXML, 18
Semantic Interpretation for Speech Recognition, 71
SIP Proxy, 12
SISR, 71
SOAP client, 72
Speech Recognition
   Server, 13
   VoiceXML without, 41
Starting scripts
   CCXML, 38
   VoiceXML, 38
Text-to-Speech
   Server, 13
   VoiceXML without, 41
User ID/password in HTTP request, 41
VCR controls in VoiceXML scripts, 70
Video clips, playing, 69
Video, recording in VoiceXML, 44
VoiceXML, 11
  Cookbook, 38
  Internationalization, 21
  Internationalizing scripts, 42
  Invoking script, 15
  Logs, 19
  Platform values, 18
  Recording audio and video, 44
  Security, 18
  Standards compliancy, 14
  VCR controls, 70
VoiceXML or CCXML script, loading, 33
VoiceXML/CCXML system, components, 12
Web Server, 12