© 2010 Avaya Inc.
All Rights Reserved.

Notice
While reasonable efforts have been made to ensure that the information in this document is complete and accurate at the time of printing, Avaya assumes no liability for any errors. Avaya reserves the right to make changes and corrections to the information in this document without the obligation to notify any person or organization of such changes.

Documentation disclaimer
“Documentation” means information published by Avaya in varying mediums which may include product information, operating instructions and performance specifications that Avaya generally makes available to users of its products. Documentation does not include marketing materials. Avaya shall not be responsible for any modifications, additions, or deletions to the original published version of documentation unless such modifications, additions, or deletions were performed by Avaya. End User agrees to indemnify and hold harmless Avaya, Avaya's agents, servants and employees against all claims, lawsuits, demands and judgments arising out of, or in connection with, subsequent modifications, additions or deletions to this documentation, to the extent made by End User.

Link disclaimer
Avaya is not responsible for the contents or reliability of any linked Web sites referenced within this site or documentation provided by Avaya. Avaya is not responsible for the accuracy of any information, statement or content provided on these sites and does not necessarily endorse the products, services, or information described or offered within them. Avaya does not guarantee that these links will work all the time and has no control over the availability of the linked pages.

Warranty
Avaya provides a limited warranty on its Hardware and Software (“Product(s)”). Refer to your sales agreement to establish the terms of the limited warranty. In addition, Avaya’s standard warranty language, as well as information regarding support for this Product while under warranty is available to Avaya customers and other parties through the Avaya Support Web site: http://support.avaya.com. Please note that if you acquired the Product(s) from an authorized Avaya reseller outside of the United States and Canada, the warranty is provided to you by said Avaya reseller and not by Avaya.

Licenses
THE SOFTWARE LICENSE TERMS AVAILABLE ON THE AVAYA WEBSITE, HTTP://SUPPORT.AVAYA.COM/LICENSEINFO/, ARE APPLICABLE TO ANYONE WHO DOWNLOADS, USES AND/OR INSTALLS AVAYA SOFTWARE, PURCHASED FROM AVAYA INC., ANY AVAYA AFFILIATE, OR AN AUTHORIZED AVAYA RESELLER (AS APPLICABLE) UNDER A COMMERCIAL AGREEMENT WITH AVAYA OR AN AUTHORIZED AVAYA RESELLER. UNLESS OTHERWISE AGREED TO BY AVAYA IN WRITING, AVAYA DOES NOT EXTEND THIS LICENSE IF THE SOFTWARE WAS OBTAINED FROM ANYONE OTHER THAN AVAYA, AN AVAYA AFFILIATE OR AN AVAYA AUTHORIZED RESELLER. AVAYA RESERVES THE RIGHT TO TAKE LEGAL ACTION AGAINST YOU AND ANYONE ELSE USING OR SELLING THE SOFTWARE WITHOUT A LICENSE. BY INSTALLING, DOWNLOADING OR USING THE SOFTWARE, OR AUTHORIZING OTHERS TO DO SO, YOU, ON BEHALF OF YOURSELF AND THE ENTITY FOR WHOM YOU ARE INSTALLING, DOWNLOADING OR USING THE SOFTWARE (HEREINAFTER REFERRED TO INTERCHANGEABLY AS “YOU” AND “END USER”), AGREE TO THESE TERMS AND CONDITIONS AND CREATE A BINDING CONTRACT BETWEEN YOU AND AVAYA INC. OR THE APPLICABLE AVAYA AFFILIATE (“AVAYA”).

Copyright
Except where expressly stated otherwise, no use should be made of materials on this site, the Documentation, Software, or Hardware provided by Avaya. All content on this site, the documentation and the Product provided by Avaya including the selection, arrangement and design of the content is owned either by Avaya or its licensors and is protected by copyright and other intellectual property laws including the sui generis rights relating to the protection of databases. You may not modify, copy, reproduce, republish, upload, post, transmit or distribute in any way any content, in whole or in part, including any code and software unless expressly authorized by Avaya. Unauthorized reproduction, transmission, dissemination, storage, and or use without the express written consent of Avaya can be a criminal, as well as a civil offense under the applicable law.

Third-party components
Certain software programs or portions thereof included in the Product may contain software distributed under third party agreements (“Third Party Components”), which may contain terms that expand or limit rights to use certain portions of the Product (“Third Party Terms”). Information regarding distributed Linux OS source code (for those Products that have distributed the Linux OS source code), and identifying the copyright holders of the Third Party Components and the Third Party Terms that apply to them is available on the Avaya Support Web site: http://support.avaya.com/Copyright.

Trademarks
The trademarks, logos and service marks (“Marks”) displayed in this site, the Documentation and Product(s) provided by Avaya are the registered or unregistered Marks of Avaya, its affiliates, or other third parties. Users are not permitted to use such Marks without prior written consent from Avaya or such third party which may own the Mark. Nothing contained in this site, the Documentation and Product(s) should be construed as granting, by implication, estoppel, or otherwise, any license or right in and to the Marks without the express written permission of Avaya or the applicable third party.

Avaya is a registered trademark of Avaya Inc.

All non-Avaya trademarks are the property of their respective owners, and “Linux” is a registered trademark of Linus Torvalds.

Downloading Documentation
For the most current versions of Documentation, see the Avaya Support Web site: http://support.avaya.com.

Contact Avaya Support
Avaya provides a telephone number for you to use to report problems or to ask questions about your Product. The support telephone number is 1-800-242-2121 in the United States. For additional support telephone numbers, see the Avaya Web site: http://support.avaya.com.
# Contents

## Chapter 1: New in this release
- Features .................................................................................................................. 7
- Other changes .......................................................................................................... 7

## Chapter 2: Introduction .......................................................................................... 9

## Chapter 3: System architecture overview ............................................................. 11
- Interpreter components .......................................................................................... 11
- Hardware architecture ............................................................................................ 12
- Development tools ................................................................................................ 13
- Interpreters ............................................................................................................ 13

## Chapter 4: Plan and engineer the interpreters ....................................................... 15
- Element Manager (EM) parameter configuration ................................................ 15
- VoiceXML and CCXML parameter configuration ............................................... 15
- General parameters - VoiceXML .......................................................................... 15
- Interpreter default behavior ................................................................................. 16
- Resource configuration parameters - VoiceXML ................................................ 21
- CCXML configuration parameters ...................................................................... 22
- Common VoiceXML Advanced Parameters ....................................................... 22
- Conversion of MMF files to WAV applications ................................................... 23
- mmfexport utility .................................................................................................. 23
- mmfexport utility format ....................................................................................... 24

## Chapter 5: Development environment .................................................................. 25
- XML Basics ............................................................................................................ 25
- Tags and elements .................................................................................................. 25
- CCXML development environment ...................................................................... 26
- CCXML basics ....................................................................................................... 26
- CCXML event handling ......................................................................................... 26
- CCXML call control .............................................................................................. 27
- CCXML conference ............................................................................................... 27
- CCXML sessions .................................................................................................... 27
- Understand programming elements ..................................................................... 28
- Counters ................................................................................................................ 28
- Protocols ................................................................................................................ 28
- CCXML enhancements ......................................................................................... 28
- Named conferences .............................................................................................. 29
- Expanded conference size .................................................................................. 29
- Conference ownership change ............................................................................ 29
- Conference enter and exit tones ......................................................................... 30
- Conference and dialogs ....................................................................................... 30
- Conference student and coach ............................................................................ 31
- Conference resizing (named conferences only) .................................................. 31
- DTMF squelch ...................................................................................................... 32
- SIP header access for inbound calls .................................................................... 32
- Outbound call SIP header manipulation ............................................................. 33
- Enhanced application tracing ............................................................................... 33
- Application invocation ......................................................................................... 33
Chapter 6: Coding elements................................................................................................................................. 81
  CCXML compliancy..................................................................................................................................................... 81
  Document control flow and execution.......................................................................................................................... 81
  Dialog elements.......................................................................................................................................................... 82
  Variables and expression elements............................................................................................................................. 83
  Event handling.......................................................................................................................................................... 83
  Telephony/Operations and resources.......................................................................................................................... 84
  VoiceXML compliancy............................................................................................................................................... 84
  Control flow and scripting........................................................................................................................................ 85
  User input.................................................................................................................................................................. 85
  System output............................................................................................................................................................ 85
  SIP Data Access.......................................................................................................................................................... 85
  JavaScript Extensions for CCXML and VoiceXML........................................................................................................ 87
  Logging.................................................................................................................................................................... 88
  Log object................................................................................................................................................................. 88
  Event Logging.......................................................................................................................................................... 89
  Counters and Gauges................................................................................................................................................ 89
  Session detail records.............................................................................................................................................. 90
  SDR reporting fields application association.......................................................................................................... 92
  Cluster object.......................................................................................................................................................... 94

Chapter 7: Integrate CCXML and VoiceXML............................................................................................................ 97
  Integration summary.................................................................................................................................................. 97
  CCXML - dialogstart.............................................................................................................................................. 98
  CCXML - dialogterminate................................................................................................................................... 98
  VXML - object......................................................................................................................................................... 98
  VXML - exit............................................................................................................................................................... 98
  VXML - disconnect................................................................................................................................................ 99
  Setting SIP headers in bye...................................................................................................................................... 99
  VXML - transfer...................................................................................................................................................... 99
  Setting SIP headers during transfer........................................................................................................................ 100

Chapter 8: Event handling......................................................................................................................................... 101
  CCXML events......................................................................................................................................................... 101
  Connection events.................................................................................................................................................. 101
Chapter 1: New in this release

The following sections detail what's new in Avaya Media Server Configuration – VoiceXML and CCXML Application Programming, NN44471-501 for Avaya Media Server Release 7.0.

- Features on page 7
- Other changes on page 7

Features

This document is new for Avaya Media Server Release 7.0.

Other changes

There are no other changes to this document.
New in this release
Chapter 2: Introduction

The Avaya Media Server is designed to execute applications written in Call Control eXtensible Markup Language (CCXML) and Voice Extensible Markup Language (VoiceXML) from a local file system or a remote Web server using HTTP.

CCXML is designed to provide telephony call control support for dialog systems like VoiceXML. VoiceXML applications are capable of creating interactive audio dialogs using prerecorded audio, synthesized speech, and input using DTMF or Speech.

This book is designed to describe the development environments, coding elements, and other programming requirements for developing applications in CCXML and VoiceXML.

Navigation

- System architecture overview on page 11
- Plan and engineer the interpreters on page 15
- Development environment on page 25
- Coding elements on page 81
- Integrate CCXML and VoiceXML on page 97
- Event handling on page 101
- Deployment on page 109
- Code validation on page 121
- Logging errors and troubleshooting code on page 123
Chapter 3: System architecture overview

This section describes the components, tools and architecture used in developing VoiceXML and CCXML applications.

Navigation

- Interpreter components on page 11
- Hardware architecture on page 12
- Development tools on page 13
- Interpreters on page 13

Interpreter components

The interpreter processes VoiceXML and CCXML applications. In performing these actions, the interpreter relies on components included on the server and client side of any operation.

Avaya MS is shared between the CCXML, and VoiceXML interpreters.

You use the VoiceXML Interpreter is a client-side application to process VoiceXML documents sent by the document server.

The document server hosts VoiceXML and CCXML documents, which are sent to the VoiceXML and CCXML Interpreters on request, where they are processed.

The implementation platform generates events in response to user actions, such as spoken or typed input, and system events, such as a timer expiration. The VoiceXML and CCXML
interpreters act on some events, as specified by the VoiceXML document, the VoiceXML and CCXML interpreters context act on others.

The VoiceXML Interpreter is a client-side application used to process VoiceXML documents sent by the document server or read from the local file system.

The CCXML Interpreter is another client-side application used to process CCXML documents sent by the document server or read from the local file system.

### Hardware architecture

VoiceXML and CCXML are developed on the Avaya MS which is a software only platform designed for generic multimedia processing and is based on open standards protocols.

You can use a SIP PSTN gateway (TP-260 or Mediant 2000) to allow the Avaya MS to provide optional PSTN access.

The gateway routes calls to the appropriate Avaya MS and provides load balancing to calls across multiple Avaya MS in the same server cluster. The gateway also detects if an Avaya MS is down or is unable to route a call, in which case the gateway transfers the call to an alternate Avaya MS.

The Media Resource Control Protocol (MRCP) is simply a protocol used to communicate with the speech servers (LVR and/or TTS).

For more information on hardware architecture, see *Avaya Media Server 7.0 Planning and Engineering, NN44471-200* for more information.
Development tools

CCXML and VoiceXML are text based XML languages, they can be written and modified in an XML editor or basic text editors (such as MS Notepad). Avaya MS supports any development tool as long as it generates standard VXML or CCXML output.

Interpreters

A solution can use either VoiceXML, CCXML or a combination of both. The requirements for a project will determine the appropriate interpreter for a particular development effort.

VoiceXML applications focus on the interaction with a caller. Speaking prompts and collecting information. VoiceXML applications are a tied to a single call.

CCXML applications provide call control and conference features that are not available to VoiceXML. A CCXML application can manage several calls in one application. However CCXML applications do not provide direct interaction with a caller. A CCXML application will use VoiceXML application(s) to serve this purpose. Use CCXML when implementing applications aimed mainly at call control, conferencing, and routing.

In most cases you will use VXML only. CCXML will be introduced when advanced call control features and/or conferences are required.
System architecture overview
Chapter 4: Plan and engineer the interpreters

This section includes information required for configuring VoiceXML and CCXML interpreters.

Navigation

- **Element Manager (EM) parameter configuration** on page 15
- **VoiceXML and CCXML parameter configuration** on page 15
- **Conversion of MMF files to WAV applications** on page 23

---

**Element Manager (EM) parameter configuration**

Element Manager is available from the Avaya MS. Use this tool to configure the VoiceXML and CCXML interpreters.

---

**VoiceXML and CCXML parameter configuration**

VoiceXML and CCXML interpreter parameters are set in Element Manager (EM).

---

**General parameters - VoiceXML**

VoiceXML parameters are set in EM under **System Configuration, Interpreters, VoiceXML, General Settings**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Range of values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow LVR Vendor Params</td>
<td>Enable setting of vendor specific recognition parameters from within a VoiceXML application.</td>
<td>Check box</td>
<td>Disabled</td>
</tr>
<tr>
<td>Proxy server</td>
<td>WEB proxy server.</td>
<td>Free form text</td>
<td>No Default Value</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Range of values</td>
<td>Default Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Proxy server port</td>
<td>WEB proxy server port number</td>
<td>0-65535</td>
<td>0</td>
</tr>
<tr>
<td>Built-in prefetching</td>
<td>Enable builtin audio file pre-fetch to validate existence of prompts.</td>
<td>Check box</td>
<td>Disabled</td>
</tr>
<tr>
<td>Recording beep URI</td>
<td>Beep tone to be played to a caller prior to recording.</td>
<td>URI</td>
<td>builtin:beep</td>
</tr>
<tr>
<td>Force DTMF to use a recognizer</td>
<td>Force the VoiceXML interpreter to use a recognizer for all DTMF grammars.</td>
<td>Check box</td>
<td>Disabled</td>
</tr>
<tr>
<td>Auto-accept server certificate</td>
<td>Always accept server certificate to create SSL connection regardless of the certificate being invalid or expired.</td>
<td>Check box</td>
<td>Enabled</td>
</tr>
<tr>
<td>Two session consultation transfer</td>
<td>Configures the type, single or two session, of consultation transfer the VoiceXML interpreter will perform.</td>
<td>Check box</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

**Interpreter default behavior**

The interpreter default behavior is shown below:
Plan and engineer the interpreters

---

Support for the 'CANCEL' universal grammar
---

```xml
<!--
** NORTEL
** Uncomment this to support the 'cancel' universal grammar for ScanSoft.
-->

<!Link event="cancel">
  <Grammar xmlns="application/x-VoiceXML" root="CANCELITEMLIST" xml:lang="en-US" version="2.0">
    <Rule id="CANCELITEMLIST" scope="public">
      <Item> cancel
        <Tag>$=cancel" /></Tag>
      </Item>
    </Rule>
  </Grammar>
</Link>
```

---

** NORTEL
** Uncomment this to support the 'cancel' universal grammar for nuance.

```xml
<!--
** NORTEL
** Uncomment this to support the 'cancel' universal grammar for IBM.
-->

<!Link event="cancel">
  <Grammar xmlns="application/x-VoiceXML" root="CANCELITEMLIST" xml:lang="en-US" version="1.0">
    <Rule id="CANCELITEMLIST" scope="public">
      <Item> cancel
        <Tag>$=cancel" /></Tag>
      </Item>
    </Rule>
  </Grammar>
</Link>
```
Plan and engineer the interpreters

---

Support for the "HELP" universal grammar

---

&.lt;--
** &ORTEL
 &lt;Uncomment this to support the 'help' universal grammar for ScanSoft.
---

&lt;tag name="i18n:help">
   &lt;grammar type="application/args+xml:root="HELP ITEM LIST" xml:lang="en-US" version="1.0">
      &lt;rule id="HELP ITEM LIST" scope="public">
         &lt;one-of>
            &lt;item>
               &lt;tag name="i18n:help"&gt; help &lt;/tag&gt;
               &lt;/item>
            &lt;/one-of>
         &lt;/one-of>
      &lt;/rule>
      &lt;/grammar>
      &lt;/i18n:help>
   &lt;/tag>
&lt;/grammar>
---

&lt;!--
** &ORTEL
 &lt;Uncomment this to support the 'help' universal grammar for nuance.
---

&lt;!--
   &lt;tag name="i18n:help">
      &lt;grammar type="application/nuance-gml">
         &lt;DATA>
            help
         &lt;/DATA>
      &lt;/grammar>
   &lt;/tag>
---

&lt;!--
** &ORTEL
 &lt;Uncomment this to support the 'help' universal grammar for IBM.
---

&lt;!--
   &lt;tag name="i18n:help">
      &lt;grammar type="application/args+xml:root="HELP ITEM LIST" xml:lang="en-US" version="1.0" format="semantic":1.0">
         &lt;rule id="HELP ITEM LIST" scope="public">
            &lt;one-of>
               &lt;item>
                  &lt;tag name="i18n:help"&gt; help &lt;/tag&gt;
               &lt;/item>
            &lt;/one-of>
         &lt;/one-of>
      &lt;/rule>
      &lt;/grammar>
   &lt;/tag>
---

You can set these parameters are set in EM under **System Configuration, Interpreters, VoiceXML, General Settings.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Range of values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TTS Implicit Allocation Algorithm</td>
<td>TTS Resource allocation algorithms: As Needed: The resource is acquired when needed and released</td>
<td>As Needed, As Needed, As Needed &amp;</td>
<td>As Needed</td>
</tr>
</tbody>
</table>
### Parameter Table

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Range of values</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
<tr>
<td>imLVRImplicitAllocationAlgorithm</td>
<td>LVR Resource allocation algorithms: As Needed: The resource is acquired when needed and released immediately after the dialog is complete. Full Session: The resource is acquired when the call starts and released at the end of the call.</td>
<td>As Needed, As Needed &amp; Hold, Full Session</td>
<td>As Needed &amp; Hold</td>
</tr>
</tbody>
</table>

---

### CCXML configuration parameters

CCXML parameters are set in EM under **System Configuration, Interpreters, CCXML**.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administration application guard timer field</td>
<td>The total guard time for admin applications in seconds.</td>
<td>300</td>
</tr>
<tr>
<td>Default fetch timeout field</td>
<td>The fetch timeout in seconds that is used during a CCXML &lt;fetch&gt;, &lt;createccxml&gt;, and &lt;dialogstart&gt;.</td>
<td>30</td>
</tr>
<tr>
<td>Inbound guard timer field</td>
<td>The total guard time for inbound applications in seconds.</td>
<td>300</td>
</tr>
<tr>
<td>Post disconnect guard timer field</td>
<td>The guard time in seconds for inbound applications to exit after the call disconnects.</td>
<td>10</td>
</tr>
</tbody>
</table>

---

### Common VoiceXML Advanced Parameters

You can set these parameters are set in EM under **System Configuration, Interpreters, VoiceXML, Advanced Settings**.
### Conversion of MMF files to WAV applications

Multi-Media Format (MMF) files, used on Media Processing Server (MPS) systems, contain audio announcements, Caller Message Recording (CMR), and/or fax data.

Avaya MS supports WAV files instead of MMF files. An MMF file may be converted on the MPS system to a set of WAV files using the mmfExport utility. The converted WAV files can then be placed in the appropriate directories on Avaya MS.

For example the following command in mmfExport converts the MMF file named numdemo into a set of linear 16-bit 8KHZ WAV files.

```
mmfExport -c WAV -p 16 -f 8000 numdemo
```

See *Avaya Media Server Commissioning, NN44471-301* for more information on WAV codecs.

---

### mmfexport utility

mmfexport exports non-CMR items from a Multimedia format (MMF) file to .wav, .au, or .aiff format files. The output file(s) may be newly created or overwritten to existing files. An MMF’s items can be exported selectively by item name, by range of EAP#, or the entire MMF can be exported. Only one MMF can be specified for each execution of the command. The MMF_basefilename is always specified. To export CMR items from an MMF file, use the cmrexport command.
**mmfexport utility format**

```
mmfexport [-q] -o output-file [-i item-name] [-i item_name...] [-c WAV | AU | AIFF] [-f #] [-p #] <mmf_file>
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-q</td>
<td>No query (silent) mode. Command is executed without status or errors reported.</td>
</tr>
<tr>
<td>-b #</td>
<td>Starting (first) EAP# of items to export. Default is 1. This option cannot be used with the -o or -i option.</td>
</tr>
<tr>
<td>-e #</td>
<td>Ending (last) EAP# of items to export. Default is last in MMF. This option cannot be used with the -o or -i option.</td>
</tr>
<tr>
<td>-o output-file</td>
<td>Specifies base filename of a single output file. If omitted, output defaults to the standard output (stdout). Must be used with at least one -i item_name option-argument. If multiple items are specified, they are merged into the output file. This option cannot be used with the -b or -e option.</td>
</tr>
<tr>
<td>-i item_name</td>
<td>Specifies the name of an item to be exported. Must be used with the -o option. Multiple items can be exported, however, the -i option must be used for each item_name argument. If item name contains spaces, enclose the item_name argument in quotation marks.</td>
</tr>
<tr>
<td>-c WAV</td>
<td>AU</td>
</tr>
<tr>
<td>&lt;MMF_FILE&gt;</td>
<td>MMF file to be exported.</td>
</tr>
<tr>
<td>-f #</td>
<td>Sample frequency in Hz for .wav files only. Allowable arguments are 8000 and 11025. Default is 11025.</td>
</tr>
<tr>
<td>-p #</td>
<td>Precision in bits per sample for .wav files only. Allowable arguments are 8 and 16. Default is 16.</td>
</tr>
</tbody>
</table>
Chapter 5: Development environment

This section describes the components of VoiceXML and CCXML applications that you need to consider during development.

Navigation

- CCXML development environment on page 26
- VoiceXML development environment on page 34

XML Basics

Extensible Markup Language (XML) is a text-based programming language used to structure data and content. It consists of a series of tags, or elements, that can be customized according to the needs of the developer.

These customizations are defined in Document Type Definition (DTD) files, which governs the behavior, relationship, and structure of all elements.

VoiceXML and CCXML are both examples of an implementation of XML. The structure of all elements is governed by the VXML and CCXML DTDs respectively.

Like other XML applications, VoiceXML and CCXML require an editing and authoring environment. This can be as simple as a text tool like MS Notepad.

Tags and elements

Both CCXML and VXML use markup tags and plain text.

A tag is a keyword enclosed by angle bracket (<and>). It contains attributes inside the angle brackets. Each attribute has a name and value, separated by an equal (=) sign with the value in quotation marks.

Tags occur in pairs, with a start tag (<tag>) that corresponds to an end tag (</tag>). Other tags and text can appear between the start and end tags. All text between the start and end tags is called an element.

If one element contains another element, the containing element is called the parent of the contained element, or the container. The contained element is called the child element of the containing element.
Though both HTML and VXML/CCXML use markup tags, the two languages use tags differently. While the markup tags in HTML describe how to render data, markup tags in XML (and therefore, VXML/CCXML) describe the data itself. The XML interpreter or browser can display data any way possible.

**CCXML development environment**

This section covers what you need to know about the CCXML development environment.

**CCXML basics**

CCXML 1.0 is an event driven declarative markup language designed to complement dialog systems such as VoiceXML by providing advanced telephony and conference functions. CCXML applications define an extended finite state machine where the developer defines a series of actions/statements to be executed based on a state variable (currentstate) and an event input (connection.connected).

Connections, Conferences, Voice Dialogs or other CCXML sessions deliver events to the application input queue. Each event is uniquely identified by its event name and event source, thus allowing the CCXML application to manage multiple events. A CCXML session maintains an identifier for each object, allowing the application to control its objects.

An inbound call or session from the telephony network initiates a CCXML Session, which is announced to the CCXML platform through an alerting event (connection.alerting). The CCXML application can accept, reject, or redirect the inbound call.

After a call enters a connected state, the CCXML application can perform operations to manipulate the media stream associated with the connection. Operations include initiating a VoiceXML dialog, placing a call and then bridging or transferring the callers, moving a connection to a conference, transferring the call to another destination, disconnecting the call or moving the connection to another CCXML session.

**CCXML event handling**

Event handling in CCXML is defined by the Event Handler Interpretation Algorithm (EHIA). Events are queued to a CCXML session then the CCXML execution engine processes each event in document order. Starting with the first <transition> element the execution engine will find the first match based on a state variable and event name. At most one transition will be entered. If no match is found the event is removed, no action is performed.
CCXML call control

Most call control actions can be controlled by the following high level tags.

- `<accept>` : Answer/accept an incoming call
- `<createcall>` : Originate or outdial
- `<reject>` : Reject an incoming call prior to connecting
- `<redirect>` : Transfer an incoming or connected call
- `<merge>` : Transfer and release two calls to the network
- `<disconnect>` : Disconnect a caller

Advanced call control is supported through the manipulation of protocol data associated with the connection event that arrive from the platform and protocol data sent from an application to the platform.

CCXML conference

CCXML supports conferencing through the implementation of four high level tags.

- `<createconference>` : Create multi-party conference
- `<destroyconference>` : Destroy a conference
- `<join>` : Join caller to a conference or bridge two connections on platform
- `<unjoin>` : Disconnect from conference or bridge

CCXML sessions

A CCXML session is comprised of one or more CCXML applications. A session can receive events from a number of "event sources" including connections, dialogs and conferences. A CCXML session is an executing CCXML document or sequence of CCXML documents. The documents in a CCXML session control and manage Voice Connections, Conferences, and Dialogs.

Voice dialog objects interact with Connections or Conferences using one-way or two-way media streams, often under the control of dialog environment such as VoiceXML.

Connection objects are real-world phone connections or system resources that facilitate interaction with a Voice Dialog.
Conference objects provide a resource for mixing media streams.

### Understand programming elements

The programming elements of a CCXML application define the actions taken by the application. This includes declaring variables, assigning values, using simple conditional and loop logic, answering/transferring/disconnecting a call, creating/destroying/joining a conference, collection input from the user, or running JavaScript.

### Counters

In addition to the CCXML 1.0 structure defined by the W3C, Avaya has added counter elements to the structure. You can use the counter elements for tracking the processing of CCXML documents on the Avaya MS server.

### Protocols

CCXML supports the SIP protocol for data transfers. You can forward SIP protocol data from the CCXML interpreter to the Dialog/VXML interpreter. This method is similar to the implementation of VXML dialog initialization without a CCXML interpreter and does not require application programming.

If you require additional protocol data then the application can use the parameter attribute of the `<dialogstart>` element.

### CCXML enhancements

The following sections describe enhancements to the CCXML programming environment used by the Avaya MS.
Named conferences

The CCXML interpreter supports “named” or global conferences. In the <createconference> tag the confname property defines a conference object that is globally accessible to other CCXML sessions on a single application processor. When the CCXML interpreter encounters a confname in the <createconference> tag the interpreter creates a new conference on the first occurrence of that unique name. All additional reference to the confname receive a reference to the common object. The object is available until the conference is destroyed. The number of participants on a conference is determined by the number of reserved listeners or talkers processed by the initial <createconference> request.

Expanded conference size

The size or number of ports reserved for a conference was extended beyond the 16 port limit of previous implementations. The maximum number of ports on a conference is 256 ports. However, the actual limit is also controlled by the CPU load on an application processor.

Conference ownership change

When using named conferences, the initial <createconference> request is tagged as the owner of the conference. When the application that owns the conference destroys or terminates the conference, all participants are removed. In order to support a meet-me type conference, the CCXML interpreter is enhanced to transfer the ownership through the hints attribute of a <createconference>. A “setowner” “hint” transfers ownership to a new document. Once ownership is set, the conference is tied to the new application instance not the application that initially created the conference. It is not an error to use the setowner property in conjunction with the initial <createconference> execution. The interpreter does not support a setowner=”false” option which could leave a conference without an owner.

Conference ownership example

<transition state="somestate" event="someevent"> <script> var confobj = new Object(); confobj.setowner = 'true'; </script> <createconference reservedtalkers="max_conf_size" conferenceid="confid" confname="testconfname" hints="confobj"/> </transition>

Or an inline JavaScript object achieves the same result.

<transition state="somestate" event="someevent"> <createconference confname="testconfname" hints="{setowner: 'true'}" reservedtalkers="128"/> </transition>
Conference enter and exit tones

Enter and exit tones are supported on a conference. An application can set either the entertone or exittone to "true" or a web or file reference. The tones are played into the conference as each participant joins or leaves the conference. The tones are tied to each connection. If the enter and exit tone properties are not set, no tone is played to the conference when a participant joins or leaves the conference. When either property is set “true” the interpreter plays a default enter/exit tone. The default tones are configurable through the Element Manager console.

When using custom enter or exit tones the platform only supports mono, 8/16 bit PCM audio format. If the audio source cannot be referenced or is not in the correct format, no tone is heard.

Enter and exit tones are only supported through a conference, they are not played when joining two callers through a bridge without a conference.

Conference enter and exit tones example

<!-- Play default enter and exit tones --> <join id1="confid" id2="conn_id" entertone=""true"" exittone=""true"" /> <!-- Play custom tones --> <assign name="enter" expr=""http://localhost:8080/ccxml/tones/verifier-youhaveentered.wav""/> <!-- Relative path find tone relative to this application uri --> <assign name="exit" expr=""verifier-goodbye.wav""/> <join id1="confid" id2="conn_id" entertone="enter" exittone="exit" />

Conference and dialogs

An application can now attach a VXML dialog to a conference resource. The CCXML interpreter supports both <dialogstart> and <dialogterminate> for a conference, in accordance with the CCXML specification. As defined in the CCXML specification an application can specify either a connectionid or conferenceid in a <dialogstart>. The connectionid and conferenceid are mutually exclusive. Only one active dialog is allowed per conference, additional attempts fail while a dialog is active. All events for a dialog (dialog.started, dialog.exit etc.) are posted to the application that initiated the dialog. The dialog terminates if this application exits while the dialog is active. A dialog can only be started by the application that is defined as the owner of the conference. Conference ownership is set in two ways, in an ad-hoc conference the ccxml session that issues the <createconference> is the owner. In global or named conferences the first ccxml session that issues a <createconference> is the owner, however ownership can be changed through the "setowner:" hint as described in Conference ownership change on page 29.
Conference student and coach

The CCXML interpreter supports Student and Coach attributes for a conference connection, also known as whisper coaching. For this feature, three or more connections are participating in a conference. A port or conference connection that is designated as the "student" is able to hear all non-muted participants in the conference including the "coach". A conference port configured as a "coach" is also able to hear all non-muted participants, however the output audio from the "coach" is only heard by the student. Each conference can have one student and one coach.

The “hints” attribute of a <join> tag enable a “student” or “coach” connection on a conference.

Conference student and coach example

```
<join id1="connection_S" id2="conference" hints="{mode:'student'}"/> <join id1="connection_C" id2="conference" hints="{mode:'coach'}"/>
```

OR

```
<script> var hintsobj = new Object(); hintsobj.mode = 'student'; </script> <join id1="connection_C" id2="conference" hints="hintsobj"/>
```

Notes: 1. The “hints” takes precedence over the “duplex” property of the <join> tag. A <join duplex=""half"" hints=""student"" …/> does not result in a half duplex student connection, but a student connection. 2. The join “hints” are only supported when joining a call to a conference and not a two party bridge.

Conference resizing (named conferences only)

The number of conference ports available to a CCXML application session is defined by the reservedtalkers and or reservedlisteners attributes of a <createconference> request. The number of ports reserved for the conference can be adjusted dynamically by the CCXML application session that “owns” the conference through additional <createconference> commands. The number of ports reserved can be increased or decreased. Decrementing the number of conference ports reserved is only possible if the number of ports in use is less than the number of ports requested in the <createconference> request. Once all the ports allocated for a conference are exhausted, additional requests to join a conference fail. Conference ports are freed when a connection <unjoins> the conference or a caller disconnects.

The resizing of a conference is only supported when the confname property is defined in the <createconference> request. If the confname property is not defined, each <createconference> request allocates a unique instance of a conference resource.
Important:
The confname property of a <createconference> identifies a globally unique instance of a conference resource.

DTMF squelch

A hint was added to disable DTMF squelch in a conference. By default, DTMF squelch is enabled in a conference so DTMF touch tones are not heard by all participants in the conference. Once DTMF squelch is enabled it cannot be disabled. The DTMF squelch control is accepted on first invocation of a <createconference>.

DTMF squelch example

To disable DTMF squelch:

<createconference reservedtalkers="conf_size" conferenceid="confid" hints="{dtmfsquelch='false'}"/>

To enable DTMF squelch, default:

<createconference reservedtalkers="conf_size" conferenceid="confid" hints="{dtmfsquelch='true'}"/>

OR without the hints attribute:

<createconference reservedtalkers="conf_size" conferenceid="confid"/>

SIP header access for inbound calls

CCXML is Draft Burke compliant for inbound calls. The information contained in a SIP header is attached to the event$.info.protocol.sip.header[] of a connection.alerting event.

SIP header access for inbound calls example

<transition event="connection.alerting"> <log label="appname" expr=""header['to']": '+event $.info.protocol.sip.header['to']" /> <log label="appname" expr=""header['call-id']": '+event $.info.protocol.sip.header['call-id']" /> <accept/> </transition>
Outbound call SIP header manipulation

CCXML supports the manipulation of a SIP header on an outbound call through the hints object of a <createcall> tag. In order to use this feature an application programmer must have an understanding of the SIP protocol. While the CCXML interpreter does not validate data attached in a SIP header the Avaya MS SIP stack may prevent an application from modifying some values. If this occurs the SIP stack creates a log when a value is not supported or cannot be modified by a user.

Outbound call SIP header manipulation example

<transition event="someevent"> <!-- define a custom header object and initialize --> <script>
var hintsobj = new Object(); hintsobj.headers = new Object(); hintsobj.headers['user-to-user'] = 'This is my UUI'; hintsobj.headers['subject'] = 'This is my subject'; hintsobj.headers['user-defined'] = 'This is my user defined header'; </script> <!-- attach the header in a createcall -->
<createcall dest="somedest" hints="hintsobj"/>

Enhanced application tracing

The CCXML interpreter has enhanced application tracing. When enabled, each tag execution creates a log entry in the CCXMLApp.txt file. Application trace is enabled by setting the log.trace object in a <script> tag. An application can enable or disable trace at any time.

Enhanced application tracing example

To enable enhanced application tracing:

<script> log.trace=true; // enable tracing log.enable=true; //enable log </script>

To disable enhanced application tracing:

<script> log.trace=false; // disable tracing log.enable=false; //disable log </script>

Application invocation

As with VXML, a CCXML application session can start through a SOAP request. However, in CCXML, the SOAP request is not required to fill the <To> and <From> fields in the SOAP envelope. When the <To> and <From> fields are filled, the Avaya MS platform initiates a call then “hands off” the request to the CCXML interpreter. The CCXML interpreter fetches the document identified by the <ApplicationUrl> field in the SOAP envelope. When the new application session is started a connection object associated with the <To> field is added to
the CCXML application session. This connection, from the CCXML application perspective, is in an inbound call in an “alerting” state. Therefore, the CCXML interpreter posts a connection.alerting event to the CCXML application session. Upon executing the <accept> tag, the application moves to the connected state. From the Avaya MS platform perspective the connection is an outbound call, however, within the CCXML application it is handled as an inbound call since the CCXML application did not initiate the call.

When the <To> and <From> fields are empty in the SOAP envelope, the CCXML interpreter fetches and starts the application identified in the <ApplicationUrl> field. The ccxml.loaded event is posted to the application. This event triggers any processing required by the application, for example, issuing a <createcall>.

⚠️ Important:

The <To> and <From> fields are required in the SOAP envelope but the contents of these fields are optional as illustrated in the following examples.

**Application invocation examples**

**Invoking an application with a call:**

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:v1="http://www.nortel.com/xmlprotocol/wsdls/session_control/applications/invoke/v1_0/">
  <soapenv:Header/>
  <soapenv:Body>
    <v1:InvokeApplication>
      <ApplicationAlias>interp::ccxml</ApplicationAlias>
      <ApplicationUrl>http://47.185.25.179:8080/apps/ccxml/load_dialog/launcherload.ccxml</ApplicationUrl>
      <To>ccxml_callee@47.185.25.178</To>
      <From>me@47.185.25.179</From>
    </v1:InvokeApplication>
  </soapenv:Body>
</soapenv:Envelope>
```

**Invoking an application without a connection (<To> and <From> are empty):**

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/
xmlns:v1="http://www.nortel.com/xmlprotocol/wsdls/session_control/applications/invoke/v1_0/">
  <soapenv:Header/>
  <soapenv:Body>
    <v1:InvokeApplication>
      <ApplicationAlias>interp::ccxml</ApplicationAlias>
      <ApplicationUrl>http://47.185.25.179:8080/apps/ccxml/load_dialog/outbound_campaign.ccxml</ApplicationUrl>
    </v1:InvokeApplication>
  </soapenv:Body>
</soapenv:Envelope>
```

---

**VoiceXML development environment**

This section covers what to know about the VoiceXML development environment.
VoiceXML basics

VoiceXML is a widely accepted open standards based language used for developing interactive voice response applications. Avaya MS supports the W3C VoiceXML 2.0 and 2.1 specifications with exceptions.

Components of the VoiceXML environment

To support a telephone interface, the VXML interpreter runs in an execution environment that includes a telephony component, pre-recorded audio, a text-to-speech (TTS) speech-synthesis component, and a speech-recognition component.

The VXML interpreter interacts transparently with the following infrastructure components based on need:

- TTS is used to render text elements in output strings.
- The telephony component handles connection issues like picking up incoming calls, detecting a hang-up, and transferring calls.
- The speech-recognition component listens to a user’s spoken input and interprets the words.
- Pre-recorded audio such as Wave files is used to communicate with the user.

VoiceXML documents

A document is an executable VXML file. The VXML interpreter loads a document file to execute it. Every VXML document starts with header information that conforms to the XML standard:

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

<!DOCTYPE vxml PUBLIC "-//W3C//DTD VoiceXML 2.0//EN" "http://www.w3.org/TR/voicexml20/voicexml.dtd">

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

These headers describe the language in which the document is written. The first tag indicates that the document is an XML document. This tag is essential. For a valid XML document, the first four letters of any XML file (including the VXML document) must be "<?xml>."
Important:

No characters, including white space characters such as space or newline, can precede these four characters in a VXML document.

The second tag identifies the Document Type Definition (DTD) which is used to validate whether the contents represent well-formed VXML. A DTD describes the data format that appears in the XML document and defines valid tags by specifying the attributes each tag possesses, and the child and content tags each tag can contain. This tag is optional.

The third tag identifies the VXML version used in the document and the designated namespace for VXML. This tag is mandatory.

All content in a VXML document is within a <vxml> element - a <vxml> start tag and a </vxml> end tag.

In the previous example, the header information contains no encoding specification. In this case, UTF-8 is presumed. Optionally, you can add an encoding declaration to indicate which encoding scheme the document uses. For example, <?xml version='1.0' encoding='ISO-8859-1'>.

---

VoiceXML applications

A VXML application consists of one or more documents. Any multiple-document application has a single application root document. Each document in an application identifies the application root document with the application attribute of the <vxml> tag. When an interpreter executes a document, it loads that document. Any application root document specified is also loaded.

Any document can use certain variables in the application. Such variables are said to have application scope.

---

VoiceXML dialogs

An application produces an auditory output through a dialog, which typically asks for additional information. The user provides this information by speaking into or pressing keys on a telephone. VXML has two kinds of dialogs: forms and menus.
VoiceXML forms

A user interacts with a form to fill in a number of fields. Every field has an associated variable, called the input-item variable or the input variable. Initially, the variable has the value of undefined. The field is filled in when the speech-recognition engine recognizes a valid response in the user utterance. The VXML <form> tag defines a form, and the <field> tag defines a field in the form.

VoiceXML menus

A menu presents the user with a number of choices. It transitions to a different dialog based on the user’s selection. The menu tag defines a <menu> and each choice consists of a <choice> element. The next attribute of a <choice> element specifies the destination dialog the interpreter transitions to when the user makes the choice. If a <form> or <menu> element is configured to be a transition destination, the id attribute for the destination dialog must specify a unique identifier.

VoiceXML grammars

The speech-recognition engine uses grammars to interpret user input. Each form field can have a grammar that specifies valid user responses. An entire form has a grammar that specifies how to fill multiple input variables from a single user utterance. Each menu choice has a grammar that specifies user responses that indicate how to fill input variables.

Customization of interpreter behavior

You can customize interpreter behavior by using the <property> tag. Various properties control how an interpreter behaves when prompting the user for input, recognizing speech or DTMF input, and fetching documents and other resources.
VoiceXML event handling

The VXML interpreter detects many predefined events based on errors, telephone disconnects, or user requests, and invokes the associated event handler to process the event.

See the Event handling on page 101 chapter in this book for more information.

User interaction components

A VoiceXML application produces verbal dialog, typically designed to ask users for information. The user provides this information by speaking into or pressing keys on a telephone.

VoiceXML has two kinds of dialogs: forms and menus.

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forms</td>
<td>The (&lt;form&gt;) element contains (&lt;field&gt;) elements with initially undefined associated variables. Users verbally enter information in a field through the speech-recognition engine, which recognizes the user's words as matched in the associated grammar. A user interacts with a form to fill in a number of fields.</td>
</tr>
<tr>
<td>Menu</td>
<td>The (&lt;menu&gt;) element contains a list of (&lt;choice&gt;) elements, which are selected by the user to transition the interpreter to a different dialog based on the user's selection. The (&lt;choice&gt;) element contains an attribute that defines the dialog to which the interpreter can transition.</td>
</tr>
</tbody>
</table>

Components of user interaction

Users interact with a VoiceXML application by speaking into or pressing buttons on a telephone or similar device. The application responds with a combination of dialogs to collect and respond to user inputs.

Types of user interaction

A user interacts with a VoiceXML application either through prompts from the application (application-directed), or through input from both the user and the application (mixed-initiative).

In application-directed inputs, the application prompts the user to either enter information in a form, through the use of the \(<form>\) element, or to select an option from a menu, through the
use of the <menu> element. In either case, the grammar available to the user is limited to that available for the form or menu.

Grammars

Grammars are the rules used by the speech-recognition engine to interpret the spoken word submitted by application users. Forms and menus have their own grammars that are used to interpret the entries and options provided by the user.

Built-in grammars can be referenced as a type attribute of a <field> element, or in a <grammar> element with the src attribute set to the with the src attribute of the form: <grammar src="builtin:grammar/digits"/>.

For mixed-initiative inputs, both the user and the application can direct the actions of the application. The user can initiate an action with or without a prompt, leaving the application to respond accordingly to the direction of the user. This interaction can extend beyond a single dialog, allowing the user to interact or activate different dialogs than the one currently being used.

Universal grammars

A universal grammar is a spoken command you can use at any point in an interaction.

VoiceXML supports both predefined universal grammars (help, exit and cancel) and application-defined universal grammars.

Universal grammars are deactivated by default. Applications can activate a universal grammar by configuring the universals property to identify which universal grammars are active.

The <link> tag contains the universal grammar, which determines the action to be taken when the universal grammar is spoken by the user.

Links

A link specifies a grammar that is independent of a specific dialog. A <link> element defines a link. Each <link> element contains a <grammar> element. A link grammar is active in the scope of the element that contains the link.

A link can specify one of the two possible actions to take if the speech-recognition engine detects a grammar match.
The link can cause a transition to a different location; in which case, the next attribute specifies the destination of the transition. Links, like menu choices, can cause transitions to other dialogs or documents.

The link can throw an event; in which case, the expr attribute specifies the event to throw. After the event is handled, execution resumes with the element that is executed when the link grammar was matched.

---

**Flow of execution**

An application executes in the order presented in a VoiceXML document until a dialog (form or menu) is encountered.

After a dialog is encountered, the application transitions based on either an explicit transition statement in the current dialog, or by responding to a user's spoken phrase to transition to a different dialog.

In addition, execution can temporarily leave the current dialog to execute a subdialog, and return to the current dialog when execution of a subdialog is complete.

If the current dialog completes execution without transitioning to a different location, the application exits. In addition, an <exit> element explicitly ends the application.

---

**Programming elements**

The programming elements of a VoiceXML application define the actions taken by the application. This includes declaring variables, assigning values, using simple conditional and loop logic, presenting speech or audio output to the user, or running JavaScript.

---

**Variables**

Variables are declared by the <var> tag and can appear in a document, a form, or in executable content. The <var> tag can assign an initial value for the variable, otherwise the value is left as undefined. The scope of the variable is determined by which element declares the variable.

Variables are declared by <var> elements:

- <var name="home_phone"/>
- <var name="pi" expr="3.14159"/>
- <var name="city" expr=""Sacramento""/>
They are also declared by form items:

- `<field name="num_tickets">`
- `<grammar type="application/srgs+xml" src="/grammars/number.grxml"/>`
- `<prompt>How many tickets do you wish to purchase?</prompt>`

Variables declared without an explicit initial value are initialized to the ECAvaya MScript undefined value. Variables must be declared before being used either in VoiceXML or ECAvaya MScript. Use of an undeclared variable results in an ECAvaya MScript error which is thrown as an error.semantic. Variables declared using "var" in ECAvaya MScript can be used in VoiceXML, just as declared VoiceXML variables can be used in ECAvaya MScript. In a form, the variables declared by `<var>` and those declared by form items are initialized when the form is entered.

If a `<var>` element specifies a variable that is already in scope, it does not declare a new variable with the same name but assigns a value to the existing variable. If the `<var>` element has an `expr`, the variable is assigned the specific value; otherwise, the variable is assigned the value undefined.

---

**Declaring variables**

Declare variables using the `<var>` tag. Use the following attributes when declaring a variable:

- `name` - mandatory - specifies the variable name
- `expr` - optional - assigns an initial value to the variable

---

**Conditional logic**

Conditional logic uses `<if>` and `<elseif>` tags to test if certain conditions are true. If the conditions are true, then the tags also execute a block of code. If no conditions are true from the `<if>` and `<elseif>` tags, then the `<else>` tag identifies a final block of code that can be run.

---

**Audio output**

Audio output comes in the form of speech output and prerecorded audio clips.

Use the `<prompt>` and `<reprompt>` tags to generate speech output. The `<value>` tag in this case is a child of the `<prompt>` and `<reprompt>` tags and is used to evaluate an expression and produce the spoken output.
Use the `<audio>` tag to play prerecorded audio.

Speech output and audio clips can be combined by using the `<audio>` tag as a child of a `<prompt>` or `<reprompt>` tag.

---

**Scripts**

A `<script>` element executes a JavaScript script, which runs in the scope of the parent element. A `<script>` element can also define functions that can be called by JavaScript expressions in the same scope.

VXML variables are equivalent to JavaScript variables and are part of the same namespace. You can use these variables in a script, just as you can use variables defined in a `<script>` element in CCXML. Declaring a variable using a `<var>` element is the same as using a `<var>` statement in a `<script>` element.

---

**Session variables**

Session variables are read-only and pertain to an entire user session. A session begins when the user starts to interact with a VoiceXML interpreter context, continues as documents are loaded and processed, and ends when requested by the user, a document, or the interpreter context. The interpreter context declares and assigns session variables. VoiceXML documents cannot declare new session variables.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>session.com.avaya.ivr.call.id</td>
<td>The unique call identifier that application and speech vendor logs use to synchronize for tuning purposes. The GSLID/CallID associated with the current call.</td>
</tr>
<tr>
<td>session.com.avaya.ivr.call.lineno</td>
<td>The line number of the call.</td>
</tr>
<tr>
<td>session.com.avaya.ivr.call.sysno</td>
<td>The IVR system number of the call.</td>
</tr>
<tr>
<td>session.com.avaya.ivr.call.isappaccept</td>
<td>Indicates whether this call is a new call (false) or a transferred call (true).</td>
</tr>
<tr>
<td>session.com.avaya.ivr.interp.sysno</td>
<td>The IVR system number that the interpreter is processing the current document on.</td>
</tr>
</tbody>
</table>
### Session variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Important:</strong>&lt;br&gt;For calls that are sent to the VoiceXML Interpreter by application handoff, it is possible for this to be different from <strong>com.avaya.ivr.call.sysno.</strong>&lt;br&gt;&lt;br&gt;The interpreter identifier that is processing the document.</td>
<td></td>
</tr>
<tr>
<td>session.com.avaya.ivr.interp.id</td>
<td>The interpreter identifier that is processing the document.</td>
</tr>
<tr>
<td><strong>Important:</strong>&lt;br&gt;For calls that are sent to the VoiceXML Interpreter by application handoff, it is possible for this to be different from <strong>com.avaya.ivr.call.lineno.</strong>&lt;br&gt;&lt;br&gt;Identifies the number of bytes in the variable avaya.ivr.accept.appdata.</td>
<td></td>
</tr>
<tr>
<td>session.com.avaya.ivr.appaccept.appdatalen</td>
<td>Identifies the number of bytes in the variable avaya.ivr.accept.appdata.</td>
</tr>
<tr>
<td>session.com.avaya.ivr.appaccept.appdata</td>
<td>Identifies the application that transferred the call to the source VoiceXML application. This variable is valid only when avaya.ivr.isappaccept=true.</td>
</tr>
<tr>
<td>session.com.avaya.ivr.appaccept.sa</td>
<td>The application-specific data that passed to the VoiceXML application, where the data format must be agreed upon between the source and target applications. This variable is valid only when avaya.ivr.isappaccept=true.</td>
</tr>
<tr>
<td>session.com.avaya.navigator.userAgent</td>
<td>This variable is configured to the user agent.</td>
</tr>
<tr>
<td>session.connection.aai</td>
<td>Evaluates to session.connection.protocol.sip.requesturi[“aai”]</td>
</tr>
<tr>
<td>session.connection.ccxml</td>
<td>Evaluates to session.connection.protocol.sip.requesturi[“ccxml”].</td>
</tr>
<tr>
<td>session.connection.protocol.name</td>
<td>Represents the name of the protocol that runs on the phone port from which the interpreter received the call. Evaluates to “sip”.</td>
</tr>
<tr>
<td>session.connection.protocol.version</td>
<td>Evaluates to “2.0”</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>session.connection.protocol.sip.headers</td>
<td>This is an associative array where each key in the array is the non-compact name of a SIP header in the initial INVITE converted to lower-case (note the case conversion does not apply to the header value). <strong>Important:</strong> VoiceXML applications can now use SIP headers as described in SIP Interface to VoiceXML Media Services. This text can be found at <a href="http://potaroo.net/ietf/all-ids/draft-burke-vxml-03.txt">http://potaroo.net/ietf/all-ids/draft-burke-vxml-03.txt</a> address. Subset 2.4 has been implemented. REFER session variable mappings is not supported.</td>
</tr>
<tr>
<td>session.connection.local.uri</td>
<td>This session variable is populated with the DNIS, if available, that the interpreter receives from the platform. If DNIS is not available when the call is presented to the interpreter, this session variable is an empty string. Evaluates to the SIP URI specified in the To: header of the initial INVITE.</td>
</tr>
<tr>
<td>session.connection.protocol.sip.media.</td>
<td></td>
</tr>
<tr>
<td>session.connection.protocol.sip.requesturi</td>
<td>This is an associative array where the array keys and values are formed from the URI parameters on the SIP Request-URI of the initial INVITE (or REFER).</td>
</tr>
<tr>
<td>session.connection.redirect</td>
<td>This array is populated by information contained in the History-Info [RFC4244] header in the initial INVITE or is otherwise undefined. Each entry (hi-entry) in the History-Info header is mapped, in reverse order, into an element of the session.connection.redirect array</td>
</tr>
<tr>
<td>session.connection.remote.uri</td>
<td>This session variable is populated with the ANI, if available, that is received by the interpreter from the platform. If ANI is not available</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>when the call is presented to the interpreter, this session variable is an empty string. Evaluates to the SIP URI specified in the From: header of the initial INVITE.</td>
</tr>
</tbody>
</table>

**Property reference**

You can use properties to assign values that affect platform behavior, such as the recognition process and timeouts. Properties can be defined for the whole application, for the whole document at the `<vxml>` level, for a particular dialog at the `<form>` or `<menu>` level, or for a particular field item.

Child elements inherit the properties of their parent elements, except where properties have been specified for the child. Properties set for a child element override those of the parent element. When different values for a property are specified at the same level, the last value in document order applies.

com.avaya.ivr.xfer.play_whisper.last_record When configured as true, the last CMR recording is played to the caller on the outbound line before the bridge is established. If no recording exists or if this property is configured as false, then nothing is played.

**Important:**
Currently, this property is supported only for bridge and two session consultation transfers.

The default is false.

com.avaya.ivr.prompt.audioserver.prefix Prefixes all audio file names with the specified prefix. For example, you can configure this as s- for Spanish.

The default is not to prefix a string to the URL.

com.avaya.ivr.prompt.builtin.gender Specifies the gender of the spoken MMF prompts. This variable is necessary for certain languages where the way the audio is spoken depends on the gender of the person speaking it.

The default gender is neutral.

com.avaya.ivr.prompt.builtin.prefix Prefixes all MMF prompts spoken with the specified prefix. For example, you can configure this as s- for Spanish.

The default is not to prefix a string to the prompt.

com.avaya.ivr.prompt.tts.usevendor
Changes the synthesizer resource to the attribute set for ‘value’ where the ‘value’ can be either ‘IBM’ or ‘Nuance’.

The default is to make a vendor independent queries for TTS resources.

com.avaya.ivr.rec.recognition.usevendor

Changes the recognition resource to the attribute set for ‘value’ where the ‘value’ can be either ‘IBM’ or ‘Nuance’.

The default is to make vendor independent queries for LVR resources.

com.avaya.ivr rec.recognition.usepoolname

Forces the allocation of the synthesizer resource to be from the pool set for ‘value’ where the ‘value’ can be any valid synthesizer pool that exists on Avaya MS.

The default is to make pool independent queries for TTS resources.

com.avaya.ivr prompt.tts.usepoolname

Forces the allocation of the recognition resource to be from the pool set for ‘value’ where the ‘value’ can be any valid recognizer pool that exists on Avaya MS.

The default is to make pool independent queries for LVR resources.

---

**Character encodings**

A VoiceXML document must contain a proper XML header. You can use this header to determine which character encodings are used by the document.

The basic header, `<xml version='1.0'?>` does not specify a character encoding, so the system uses the default encoding UTF-8.

You can add an optional encoding declaration to tell the VoiceXML interpreter which encoding scheme the document uses. Use the encoding header only if you use an encoding scheme other than UTF-8, such as ISO-8859-1. An example of this is `<xml version='1.0' encoding='ISO-8859-1'?>`.

If you declare an encoding scheme, you must use the declared encoding scheme consistently throughout your document.

---

**Resource fetching**

A VoiceXML application uses the fetchtimeout value to determine how long to wait for a resource to be returned before causing an error.badfetch event. If this value is not specified, then the default value of 60 seconds is used.
You can use the default timeout to invoke a URL for a new incoming call or a VoiceXML dialog started by a CCXML application. If this interval is too large then you must consider invoking a page from the local file system, which configures fetchtimeout to a smaller value.

Object tag references

Use the `<object>` element to expose platform-specific functionality for use by a VoiceXML application. The `<object>` element uses the tags in the `<object>` tag during initialization and execution. As a result, tags within `<object>` cannot be treated as alternative content. If a prompt tag is specified within the `<object>` tag, then the prompt is spoken prior to executing the `<object>` tag.

The following table lists the classid attributes available in the `<object>` element.

<table>
<thead>
<tr>
<th>Classid attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.avaya.ivr.dialog.user</td>
<td>Generates and sends a dialog user event to the CCXML application. For more details, see <a href="#">VXML - object</a> on page 98.</td>
</tr>
<tr>
<td>com.avaya.ivr.releaseresource</td>
<td>Forces the release of a resource that is currently acquired. For more details, see <a href="#">VoiceXML resource management</a> on page 60.</td>
</tr>
<tr>
<td>com.avaya.ivr.vcrcontrol.playmedia</td>
<td>Initiates a media play with VCR controls enabled. For more details, see <a href="#">VCR Controls of Media Playback</a> on page 61.</td>
</tr>
<tr>
<td>com.avaya.ivr.delay</td>
<td>Delays the execution of the VoiceXML script. For more details, see <a href="#">Delay</a> on page 78.</td>
</tr>
<tr>
<td>com.avaya.ivr.cpdDetect</td>
<td>Enables Call Progress Detection. For more details, see <a href="#">Call Progress Detection</a> on page 65.</td>
</tr>
</tbody>
</table>
Transfer element

The `<transfer>` element directs the interpreter to connect the caller to another entity (such as a telephone line or another voice application), indicated by the `dest` attribute. During the transfer operation, the current interpreter session is suspended.

Src and Srcexpr attribute

The VoiceXML specification (2.0 or 2.1) does not define how to provide the capability to forward the calling parties URI in a transfer request. The VoiceXML schema has been extended to allow the application programmer to specify the calling party number that should be used by the Avaya MS when initiating the transfer. The `<transfer>` tag has been extended to support the following additional attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>URI of the source/calling party (telephone, IP telephony address). The tel:URL and sip:URL syntax are supported</td>
</tr>
<tr>
<td>srcexpr</td>
<td>An ECAvaya MScript expression yielding the URI of the source (calling party).</td>
</tr>
</tbody>
</table>

The following table summarizes the three types of transfers that a VoiceXML application can perform.

<table>
<thead>
<tr>
<th>Transfer type</th>
<th>Attribute value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blind</td>
<td>bridge/type</td>
<td>The Avaya MS redirects the caller to the receiver without remaining in the connection, and does not monitor the outcome.</td>
</tr>
<tr>
<td>bridge</td>
<td>bridge/type</td>
<td>The Avaya MS adds the receiver to the connection. Document interpretation suspends until the transferred call terminates. The Avaya MS remains in the connection for the duration of the transferred call.</td>
</tr>
<tr>
<td>consultation</td>
<td>type</td>
<td>Similar in function to the blind transfer, however the outcome of the transfer call setup (outdial to third party) is known and the caller is not dropped because of an unsuccessful transfer (outdial) attempt.</td>
</tr>
</tbody>
</table>
Consultation Transfers

Avaya MS supports two forms of consultation transfer single session and two session. The goal of a transfer is to connect the caller to another entity (for example, another telephone). This other entity is referred to as the callee.

In a single session consultation transfer the caller is transferred to the callee using the callers existing IVR session by way of the SIP REFER method (refer to RFC3515). The VXML application is informed of the success or failure of the transfer request based on the SIP signaling that transpires.

The event connection.disconnect.transfer concludes the transfer and is thrown when Avaya MS receives the SIP Notify (200). In the event of an error in attempting the transfer the application will receive one of the conditions identified in Table 1: SIP Mappings for VoiceXML Transfers on page 49.

Single session consultation transfers are enabled using EM. In EM navigate to: “Home > System Configuration > Interpreters > VoiceXML > General Settings”, and de-select: “Two session consultation transfer”.

Note: The connecttimeout attribute of <transfer> is currently not supported with single session consultation transfers.

Table 1: SIP Mappings for VoiceXML Transfers

<table>
<thead>
<tr>
<th>SIP Code</th>
<th>Event</th>
<th>Form Item Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>404: Not Found</td>
<td>error.connection.baddestination</td>
<td></td>
</tr>
<tr>
<td>405: Method Not Allowed</td>
<td>error.unsupported.transfer.blind or error.unsupported.transfer.bridge or error.unsupported.transfer.consultation</td>
<td></td>
</tr>
<tr>
<td>480: Temporarily Unavailable</td>
<td>error.connection.noroute</td>
<td></td>
</tr>
<tr>
<td>486: Busy Here 600: Busy Everywhere</td>
<td>error.connection.noroute (blind)</td>
<td>busy (consult &amp; bridge)</td>
</tr>
<tr>
<td>408: Request Timeout 487: Request Cancelled</td>
<td>error.connection.noroute (blind)</td>
<td>noanswer (consult &amp; bridge)</td>
</tr>
<tr>
<td>503: No Service</td>
<td>error.connection.noresource</td>
<td></td>
</tr>
<tr>
<td>504: Server Timeout</td>
<td>error.connection.noroute (blind)</td>
<td>netbusy (consult &amp; bridge)</td>
</tr>
</tbody>
</table>
In a two session consultation transfer a new session is established between the IVR and the callee. Upon successful establishment of this new session between the IVR and the callee, the IVR will initiate a SIP REFER of the caller to the callee with REPLACES to request that a direct connection between the caller and callee be established to replace the existing connection between the IVR and the callee (refer to RFC3891). The VXML application is informed of the success or failure of the transfer request based on the SIP signalling that transpires.

The event connection.disconnect.transfer concludes the transfer and is thrown when Avaya MS receives the SIP Notify (200). In the event of an error in attempting the transfer the application will receive one of the conditions identified in Table 1: SIP Mappings for VoiceXML Transfers on page 49.

Two session consultation transfers are enabled using EM. In EM navigate to: “Home > System Configuration > Interpreters > VoiceXML > General Settings” and select “Two session consultation transfer” (this is the default).

Transfer Message Flows

The following figures are transfer diagrams describing: SIP Blind and Bridge, SIP Single Session, and SIP Two Session Transfers.
SIP Blind and Bridge Transfer

SIP Consultative Transfer (Single Session)
Prompting

VoiceXML applications use prompting to request information from a user. VoiceXML 2.1 defines a new tag `<foreach>` and calls for the implementation of the `<mark>` (SSML) tag.

Speech Synthesis Markup Language (SSML)

The following are important factors when you use SSML:

- **Built-in audio use of SSML supports only the `<value>` tag with the class attribute.**

**Important:**

Not all SSML tags are supported by all speech synthesis vendors. See vendor-specific documentation for SSML support issues.

mark

The `<mark>` tag is an SSML tag you can use to detect barge-in during prompt playback.
This tag places a marker (named by the VoiceXML application) into the text or tag sequence. VoiceXML 2.1 extends this tag (beyond the SSML 1.0 specification that includes a name attribute) to include a nameexpr attribute (ECAvaya MScript expression that evaluates to the name of the mark).

The following properties on the application.lastresult$ object are set whenever the application.lastresult$ object is assigned and a <mark> has been executed:

- application.lastresult$.markname - The name of the mark last executed by the SSML processor before barge-in occurred or the end of audio playback occurred - If no mark was executed, this variable is undefined.
- application.lastresult$.marktime - The number of milliseconds that elapsed since the last mark was executed by the SSML processor, until barge-in occurred or the end of audio playback occurred - If no mark was executed, this variable is undefined.

When multiple marks exist in the SSML text only the last mark executed is assigned.

**Important:**
Not all SSML processors support the <mark> tag, and some impose restrictions on the characters that can be used for the mark name.

**Important:**
When a <mark> is executed during the processing of a form item, the interpreter also sets itemName$.markname and itemName$.marktime shadow variables for the form item, with the same values used for the lastresult shadow preceding variables.

---

**Audio prompts**

After they are loaded, audio prompts can be played by specifying the keyword builtin: inside the attribute src of the <audio> element. The prompt is referenced by name or directory and name.

For example: <audio src="builtin:NUMBER DEMO GREETING"/>

---

**Text To Speech (TTS)**

SSML specifies only the <Say-As> element, its attributes, and their purpose. It does not enumerate the possible values for the attributes. The result is the potential for different <Say-As> support from each vendor. For more information about SSML (Say-As) support issues, see the vendor documentation.
See Conversion of MMF files to WAV applications on page 23 for information on converting MMF files to WAV files.

### foreach

The `<foreach>` tag allows VoiceXML applications to iterate through an ECAvaya MScript array and to execute the content contained within the `<foreach>` element for each item in the array. One of the more common applications of the `<foreach>` tag is during prompting. This tag gives the VoiceXML application the ability to concatenate prompts dynamically.

### Interactive Voice Response applications and VoiceXML 2.1

Interactive Voice Response (IVR) applications are a compilation of VoiceXML documents that define the instructions the application performs.

### Version tag

The `<vxml>` tag of any document built according to VoiceXML 2.1 standards must have a version attribute of 2.1 or higher.

If the version is set to less than 2.1, then the enhancements and additions made to the VoiceXML 2.1 specifications, except for `<mark>` and `<property>` that are ignored, results in errors in the application.

### Form tag

Inside the `<vxml>` tag, a document is divided into discrete dialog elements called forms. Each form has a name and executes a portion of the dialog. For example, a form called mainMenu prompts the caller to select from a list of options and then recognizes the response.

A form is denoted by the use of the `<form>` tag and can be specified by including the `<id>` attribute to specify the form name. The denotation helps if the form is referenced at another point in the application or by another application.
Gather user input

Field items are element groups you can use to gather information from a caller. This information is assigned as values to variables in the application.

Field items contain prompts to guide the caller on what to say, grammars to define the interpretation of what is said, and event handlers.

The field item encloses a <dialog> tag that contains a <prompt> tag used to ask a user for an input, recognizes the input according to the rules supplied by a <grammar> tag, and can use the <catch> tag to receive events suited to that portion of the dialog.

The following form items are available in the VoiceXML 2.1 specification.

<table>
<thead>
<tr>
<th>Form Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;field&gt;</td>
<td>Gathers input from a user through speech or DTMF recognition as defined by a grammar.</td>
</tr>
<tr>
<td>&lt;record&gt;</td>
<td>Records an audio clip from the user.</td>
</tr>
<tr>
<td>&lt;transfer&gt;</td>
<td>Transfers the user to another phone number.</td>
</tr>
<tr>
<td>&lt;object&gt;</td>
<td>Invokes a platform-specific object that can gather user input and returns the result as an ECAvaya MScript object.</td>
</tr>
<tr>
<td>&lt;subdialog&gt;</td>
<td>Performs a call to another dialog or document (similar to a function call) and returns the result as an Script object.</td>
</tr>
</tbody>
</table>

Each form item has an associated variable which, if not previously defined and assigned a value (for field items, this form is the same as the field item variable), is initially configured as undefined. The variable name can be defined with the name attribute, or it can remain nameless. A guard condition exists for each item, that tests if an item variable currently has a value. If it does, the execution of that particular form item is skipped; otherwise, execution proceeds normally.

Creating a multiform application

A multiform ASR or IVR application can be created to implement elements that control dialog flow, perform recognition tasks, and handle events.

The field item encloses a <dialog> unit that <prompt>s a user for an input, recognizes the input according to the rules supplied by a <grammar>, and can <catch> events suited to that portion of the dialog.

An application with several interdependent functions uses multiple forms within a document. The application can transition between forms, as the dialog dictates <goto> tag. Use the <if>,
<elseif>, and <else> elements for conditional statements in VoiceXML. Each element must use a cond attribute to specify an ECAvaya MScript boolean condition.

Use the <goto> tag to transition to another form item in a current form, to another form dialog in the current document, to another document, or to another form in another document. The syntax for this tag is as follows:

- transitioning to a form item within a form: `<goto nextitem="some_form_items_var_name" />`
- transitioning to another form within the current document: `<goto next="#some_form_id" />`
- transitioning to another document `<goto next="http://www.some_url.com/some_doc.vxml" />`

---

**Caching resources**

A VoiceXML hosting site can simultaneously attend to multiple phone calls. All phone calls share the same VXML interpreter cache. Resources downloaded for an application on one phone call can be available to the next phone call in that application or to a different application on the same media gateway.

A VoiceXML hosting site typically contains several VXML media gateways to enable multiple and simultaneous call handling and uses a proxy cache for VXML resources. Prior use of an application results in pages made available in the persistent state-wide proxy cache. When a new call arrives for that application, the call can use pages from this cache, rather than downloading the information again.

When interacting with an application, a user typically calls a phone number that is associated with a specific site. That site starts a local copy of the VXML interpreter for the application and runs a copy of the application on one of its media servers, retrieving resources as required from wherever they reside.

**Important:**

Each phone call to a hosting site uses a local copy of the VXML interpreter, which is not shared with other phone calls. However, the VXML interpreter cache is common to the entire VXML media gateway, not for a single phone call into that media gateway. An individual instance of a VXML interpreter (that is, a single phone call) does not have a separate cache.
Multiple users can use various applications on the same hosting site. Each call uses a different set of resources, which can overlap. Assuming that the resources are fresh forever, the sequence is as follows:

- The first time a call accesses a particular resource, the interpreter generates a request, looks in the local VXML interpreter cache and then in the site-wide proxy cache, and finally retrieves the resource from the appropriate Web server.
- When the server sends back a successful response, the resource is first stored in the site-wide proxy cache, then passed to the local interpreter cache where it is stored before it is used by the running application.
- If the same resource is requested again in the same call, the interpreter receives it directly from the local interpreter cache without having to download it again from the proxy cache or the original server.
- When the call ends, the copy remains in the local VXML interpreter cache for the VXML media gateway and in the site-wide proxy cache.
- If another call arrives at the same media gateway asking for the same resource, the call can use the copy in the local cache. If the call arrives at a different media gateway and asks for the same resource, the call does not have the resource in its local cache, but it can use the resource copy in the proxy cache.

---

**Fetch resources**

VoiceXML applications use VXML documents and other file types, including recorded audio data and speech and DTMF grammars. You can access these resources can be accessed with standard Web URIs, and they are located on a Web server.

Resources needed by an application are fetched from the Web server where they are stored. This location can be different than the location of the VXML interpreter. You can save resources locally at the hosting site (cached) by the VXML interpreter for later use by the same or different application on another call.

A resource request consists of a request type (typically a GET request for VXML requests), the URI of the resource wanted, and headers that specify what is valid data in the returned resource. The response contains information about the type of response, headers describing the actions the resource can perform, and possibly the actual resource.
Application control

If a resource is controlled completely from the VXML application, the basic HTTP fetch sequence for a GET request is as follows:

• The first time the resource is requested, the request goes to the origin server or to the server on which the resource resides.

• The origin server returns a response, including date header and optionally an Etag header.

• The requester records the fetch date, after which the resource is stored in the cache, and then returned to the requesting application.

• If the server has specified no expiration information, the resource expires immediately.

• The next time the resource is requested, the interpreter makes a new request for the resource, optionally including the If-Modified_Since or If-None-Match headers configured to the cached Etag.

• The server uses the Modified_Since or If-None-Match headers to determine whether a new copy of the resource needs to be sent in the body of the response or whether it must indicate that the requesting application can use the copy in its cache.

• If the response includes a new copy of the resource, the new copy and its headers replace the old copy in the cache.

Server control

If a resource is primarily controlled from a server, the basic HTTP fetch sequence for a GET request is as follows:

• The first time the resource is requested, the request goes to the origin server as a GET request

• The origin server returns a response.

• Unless the response header includes a Cache-control : no-cache or a Cache:control : no-store header. The response headers are cached for later use by the requester.

• The fetch date is recorded.

• The resource stored and returned to the requesting application.

The response still includes the Date header and can include Last-modified or Etag headers. With server control, the response typically includes either an Expires header or a Cache-control : max-age header. If it contains both, the max-age header takes precedence over the Expires header. The Expires header indicates an exact date and time at which the resource
expires. Cache-control: max-age indicates the number of seconds after the date at which the resource expires.

The following conditions apply to server control:

- The next time the resource is requested, if the resource was cached, the interpreter uses the appropriate combination of Expires, max-age and Date headers to determine whether the resource expired.
- If the resource has not expired, the interpreter returns to the cached copy.
- If the resource has expired, the interpreter makes a new request for the resource.
- If available, the new request includes both the If-Modified-Since header, configured to the Last-Modified header of the original request, and the If-None-Match header, configured to the Etag of the original response.
- If both are sent, the If-None-Match header takes precedence.
- If neither is present, the request must be for a new copy of the resource.
- The server uses these headers to determine if a new copy of the resource needs to be sent in the body of the response or whether it must indicate that the requesting application can use the copy in its cache.
- If the response includes a new copy of the resource, the new copy and its headers replace the old copy in the cache.

Fetch policies

The attributes and properties that control the aspects of fetching and caching resources are collectively referred to as the application fetch policies. These policies govern the following aspects of fetching:

- Prefetching resources - The VXML interpreter can try to start fetching resources before they are actually needed (prefetch them), in an attempt to have them available when required.
- Handling fetching delays - Noticeable delays can occur between when a resource is requested and when it is available.
- Controlling the use of cached resources - These policies control request and response headers.

Some fetch policies are configured by a single property for all types of resources. You can configure other fetch policies separately for various types of resources. For these policies, a corresponding set of properties exist, one for each resource type.
The following conditions apply to server control:

- VXML Documents
- Recorded Audio Data
- grammar files
- JavaScript Source Files
- SSML Files (Extension)
- XML Data Files (Extension)

In addition, for all these fetch policies, the VXML tags support a corresponding attribute.

All policies have default settings. An application can change any default setting with a <property> element that assigns a property corresponding to the policy to be changed. Any tag that requests a fetch operation includes attributes you can configure to override the current policy settings during that one fetch operation:

- A property configured in the <vxml> element of a single-document application or the application root document of a multiple-document application assigns the policy for fetching resources from that document and the application, overriding the default setting.
- A property configured in the <vxml> element of a nonroot document of a multiple-document application assigns the policy for fetching resources from that document, overriding the setting for the application.
- A property configured in a <form> or <menu> element assigns the policy for fetching resources from that dialog, overriding the setting for the containing document.
- A property configured in the form item assigns the policy for fetching resources from that form item, overriding the setting for the containing form.
- An attribute configured in an element that fetches a resource assigns the policy for that fetch, overriding any other setting for that policy.

---

**VoiceXML resource management**

Programmers can specify when a particular type of resource (TTS, ASR, IAS) should be allocated and when that resource should be freed. This functionality is an Avaya specific custom extension implemented using the <object> tag.

The TTS Implicit Allocation Algorithm and LVR Implicit Allocation Algorithm parameters, which define the implicit allocation algorithm for TTS and Recognition resource handling support a third mode for better control of the resources.

<table>
<thead>
<tr>
<th>Parameter setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>As Needed (0)</td>
<td>The Resource is acquired when needed and released immediately after the dialog is</td>
</tr>
</tbody>
</table>
An object tag has been added to force the release of a resource that is currently acquired. This object tag takes a parameter to determine the type of resource to free. When this tag is executed and the requested resource is currently acquired, the resource is freed and a success status is returned. If the resource is not currently acquired, the request is silently ignored and a success status is returned. The object tag is described as follows:

classid 'com.avaya.ivr.releaseresource'

param 'restype' where restype can be one of the following:

• 'lvr' – recognition resource
• 'tts' – synthesizer

VCR Controls of Media Playback

This feature adds VCR type controls on the playback of prompts. These controls are the following: stop, back, forward, delete, pause, resume, replay, slow, and fast.

The object tag “com.avaya.ivr.vcrcontrol.playmedia” extends the VoiceXML language to support VCR type controls of Media Playback. It allows a caller to skip the prompt playback forward and backward a pre-determined amount of time, pause it, resume playing, restart playback from the beginning, slow down the playback, speed up the playback, return the playback to normal speed, and stop the playback giving control back to the VoiceXML.
application. The VoiceXML application upon receiving control back is informed of how the prompt was terminated (for example: user request, or normal completion), and if terminated by user request the application is informed of the fully programmable action requested by the user (stop and continue, stop and delete). This object tag is described as follows:

Table 2: VCR Controls of Media Playback (com.avaya.ivr.vcrcontrol.playmedia)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Supported as defined by the specification.</td>
</tr>
<tr>
<td>expr</td>
<td>Supported as defined by the specification.</td>
</tr>
<tr>
<td>cond</td>
<td>Supported as defined by the specification.</td>
</tr>
<tr>
<td>classid</td>
<td>Configure as com.avaya.ivr.vcrcontrol.playmedia to initiate a media play with VCR controls enabled.</td>
</tr>
<tr>
<td>codebase</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>codetype</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>data</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>type</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>archive</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>fetchhint</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>fetchtimeout</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>maxage</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>maxstale</td>
<td>Not used and ignored.</td>
</tr>
</tbody>
</table>

The following values are passed using the <param> element.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mediaName</td>
<td>no</td>
<td>The URI of the media item to be played.</td>
</tr>
</tbody>
</table>
| vcrcommand     | yes      | The VCR commands available to the caller for manipulating the playback of the media item. This parameter is used to define the mapping between caller DTMF input and VCR controls (skip forward, skip back, pause, resume, replay, slow, fast, and normal). Format: <param name="vcrcommand" value="dtmf=A,action=B,adjust=C,..."/>
  • Where A is a single value from the DTMF set {1,2,3,4,5,6,7,8,9,0,*,#}. |
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
</table>
| vcrgrammar     | yes      | Defines what is assigned to the properties of the form item variable (the `<object>` name variables properties) when the caller requests to terminate the media play. This parameter is used to define the mapping between caller DTMF input and the application defined actions to be taken for this input. Format: `<param name="vcrgrammar" value="dtmf=A,action=B;...">`  
  
  • Where A is a single value from the DTMF set `{1,2,3,4,5,6,7,8,9,0,*,#}`  
  Note: The values picked here must occur in this parameter only once, and must not have been already specified in the "vcrgrammar" parameter.  
  
  • Where B is any text that you would like to have returned to the application in the `<object>` name input item variable when the caller enters the specified touch tone. |

When the “com.avaya.ivr.vcrcontrol.playmedia” `<object>` is executed, it returns an ECAvaya MScript object as the value of the form item variable. This ECAvaya MScript object has the following properties:

status - The variable that stores the status of the playmedia operation.
  
  • A value of zero (0) is assigned if the media play fails for any reason (cannot speak the media item specified by the mediaName parameter).
  
  • A value of one (1) is assigned if the media play completes because of user input (callers input matched the vcrgrammar parameter).
  
  • A value of two (2) is assigned if the media play completes normally.
action - The variable that is used to store the users DTMF input action as defined by the vcrgrammar parameter. This shadow variable is valid only when status is set to 1, and is set to "" in all other cases.

Rules:

1. A default vcrcommand is used when the vcrcommand parameter is not specified. The following represents the default behavior (the default value for vcrcommand if not specified):

   "dtmf=1,action=backward,adjust=5000ms;dtmf=3,action=forward,adjust=5000ms;dtmf=4,action=pause;dtmf=6,action=resume"

   No default is used when the application has defined the vcrcommand parameter.

2. A default vcrgrammar is used when the vcrgrammar parameter is not specified. The following represents the default behavior (the default value for vcrgrammar if not specified):

   "dtmf=7,action=delete;dtmf=9,action=stop"

   No default is used when the application has defined the vcrgrammar parameter.

3. The supported vcrcommand actions are: forward, backward, pause, resume, replay, fast, normal, and slow. The vcrcommand action value can only be one of these predefined values, otherwise, error.semantic will be thrown. For backward and forward actions, the adjust attribute is used to specify how far to skip backward or forward. The value for adjust can be in milliseconds (ms default), seconds (s), or percentage (%).

4. If duplicate actions/grammars are defined for the same dtmf touch tone input, error.semantic will be thrown.

5. For vcrcommand and vcrgrammar you must specify both the dtmf and action. error.semantic will be thrown if one is omitted.

6. The vcrcommand and vcrgrammar value strings can have white space immediately after the semicolon (;) command separator. White space is not permitted at any other points in the string.

   Avaya does not support issues reported that are related to the occurrence of white space in this string other than the supported locations identified above.

7. Action fast will double the play speed once, duplicate fast actions will be ignored. If the prompt is in slow mode, action fast will bring the prompt into fast mode, rather than normal mode.

8. Action slow will half the play speed once, duplicate slow actions will be ignored. If the prompt is in fast mode, action slow will bring the prompt into slow mode, rather than normal mode.
9. Action normal will bring the prompt into normal speech mode for all cases.

10. The mediaName parameter must be specified, otherwise error.semantic will be thrown.

Example:

```xml
<object name="vcrcontrol" classid="com.avaya.ivr.vcrcontrol.playmedia"> 
  <param name="mediaName" value="http://myserver/recording.wav"/> 
  <param name="vcrcommand" value="dtmf=1,action=backward,adjust=10%; dtmf=3,action=forward,adjust=5000ms;
   dtmf=4,action=pause; dtmf=6,action=resume; dtmf=0,action=replay"/> 
  <param name="vcrgrammar" value="dtmf=7,action=delete; dtmf=9,action=myaction;
   dtmf=",action=stop"/> 
  <filled> 
    <if cond="vcrcontrol.status == 0"> <!-- failure case --> 
      <prompt>Cannot play the URL</prompt> 
      <goto next="#mainmenu"></goto>  
    </if> 
    <else/> 
    <if cond="vcrcontrol.action == 'delete'"> 
      <log>Caller delete request while Playing Msg</log> 
      <goto next="#deletemessage"/> 
    </if> 
    <elseif cond="vcrcontrol.action == 'myaction'"> 
      <log> stop playing the msg</log> 
      <goto next="#nextmessage"/> 
    </elseif> 
    <elseif cond="vcrcontrol.action == 'stop'"> 
      <log> Caller stop request while playing the msg</log> 
      <goto next="#userage"/> 
    </elseif> 
    <else/> 
  </if> 
  <prompt> <object name="vcrcontrol" classid="com.avaya.ivr.vcrcontrol.playmedia"> 
    <param name="mediaName" value="http://myserver/recording.wav"/> 
    <param name="vcrcommand" value="dtmf=1,action=backward,adjust=10%; dtmf=3,action=forward,adjust=5000ms;
     dtmf=4,action=pause; dtmf=6,action=resume; dtmf=0,action=replay"/> 
    <param name="vcrgrammar" value="dtmf=7,action=delete; dtmf=9,action=myaction;
     dtmf=",action=stop"/> 
    <filled> 
      <if cond="vcrcontrol.status == 0"> <!-- failure case --> 
        <prompt>Cannot play the URL</prompt> 
        <goto next="#mainmenu"></goto>  
    </if> 
      <else/> 
      <if cond="vcrcontrol.action == 'delete'"> 
        <log>Caller delete request while Playing Msg</log> 
        <goto next="#deletemessage"/> 
      </if> 
      <elseif cond="vcrcontrol.action == 'myaction'"> 
        <log> stop playing the msg</log> 
        <goto next="#nextmessage"/> 
      </elseif> 
      <elseif cond="vcrcontrol.action == 'stop'"> 
        <log> Caller stop request while playing the msg</log> 
        <goto next="#userage"/> 
      </elseif> 
      <else/> 
  </if> 
</filled> 
</object>
```

---

**Call Progress Detection**

This feature is used to determine if the current call (typically during call setup) is connected to a real person, an answering machine, a fax machine, or a Telecommunications Device for the Deaf (TDD). This information is relayed to the application so that the application can handle the situation as it sees fit.

The feature uses a new object tag, com.avaya.ivr.cpdDetect. When this `<object>` is executed, it returns an ECAvaya MScript string as the value of the form item variable. The value of this form item variable can be one of the following.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD</td>
<td>Answering machine is detected.</td>
</tr>
<tr>
<td>FAX</td>
<td>Fax machine is detected.</td>
</tr>
<tr>
<td>NOISE</td>
<td>Noise is detected.</td>
</tr>
<tr>
<td>SILENCE</td>
<td>Silence is detected.</td>
</tr>
<tr>
<td>TDD</td>
<td>Telecommunications Device for the Deaf is detected.</td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>Nothing is detected and the call progress detection algorithm has terminated.</td>
</tr>
<tr>
<td>VOICE</td>
<td>Voice is detected.</td>
</tr>
</tbody>
</table>
The attributes for the object tag, com.avaya.ivr.cpdDetect are described as follows:

**Table 3: Call Progress Detection (com.avaya.ivr.cpdDetect)**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Supported as defined by the specification.</td>
</tr>
<tr>
<td>expr</td>
<td>Supported as defined by the specification.</td>
</tr>
<tr>
<td>cond</td>
<td>Supported as defined by the specification.</td>
</tr>
<tr>
<td>classid</td>
<td>Configure as com.avaya.ivr.cpdDetect to enable Call Progress Detection.</td>
</tr>
<tr>
<td>codebase</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>codetype</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>data</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>type</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>archive</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>fetchhint</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>fetchtimeout</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>maxage</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>maxstale</td>
<td>Not used and ignored.</td>
</tr>
</tbody>
</table>

The following values are passed using the `<param>` element.

**Important:**

One or more detect parameters (**detectamd**, **detectfax**, **detectnoise**, or **detecttdd**) must be set to true. If not, an error.semantic is thrown.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
</table>
| mediatype      | yes      | **Important:**
<p>|                |          | This parameter has been replaced by detectamd. The mediatype value only supports a setting of AMD. If this parameter is not specified or an invalid mediatype parameter is specified, error.mediatype is thrown. |
| detectamd      | yes      | Enable or disable answering machine detection (AMD). To enable AMD, set this parameter to “true” and to disable AMD, set it to “false”. This parameter defaults to “false” if not specified. |</p>
<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>detectfax</td>
<td>yes</td>
<td>Enable or disable fax machine detection. To enable fax detection set this parameter to “true” and to disable fax detection, set it to “false”. This parameter defaults to “false” if not specified.</td>
</tr>
<tr>
<td>detectnoise</td>
<td>yes</td>
<td>Enable or disable noise detection. To enable noise detection, set this parameter to “true” and to disable noise detection, set it to “false”. Do not use this detection in conjunction with other detectors (for example; tdd, fax, or voice). This parameter defaults to “false” if not specified.</td>
</tr>
<tr>
<td>detecttdd</td>
<td>yes</td>
<td>Enable or disable Telecommunications Device for the Deaf (TDD) detection. To enable TDD detection, set this parameter to “true” and to disable TDD detection, set it to “false”. This parameter defaults to “false” if not specified.</td>
</tr>
<tr>
<td>maxnospeech</td>
<td>yes</td>
<td>The maximum length of silence (in milliseconds) before timing out. On timeout, the form item variable is set to “SILENCE”. Defaults to the Avaya MS configuration parameter “CPD Answering Machine Detection No Speech Timeout (msec)”, currently 6000.</td>
</tr>
<tr>
<td>waittoleavemsg</td>
<td>yes</td>
<td>Upon detection of an answering machine, an application can either leave a message on the answering machine or terminate the call prior to leaving a message and retry another number. This parameter controls the behavior when an answering machine is detected. If set to “true” the object tag does not return until the answering machine greeting is complete and is presumed safe for the application to leave a message. If set to “false” (the default) the object tag returns as soon as the answering machine is detected (and likely still playing its greeting). Defaults to “false”.</td>
</tr>
<tr>
<td>persist</td>
<td>yes</td>
<td>Used to continue AMD detection even after this object returns. AMD detection continues only if “persist” is set to “true” and the object returns “AMD” in its form item variable. It does not continue if “persist” is “false” or the object returns “VOICE” or “SILENCE” in its form item variable. This is typically set to “true” when dealing with answering machines that have an initial greeting followed by a short pause and then the called party's greeting (common with voice mail hosted by wireless carriers). Defaults to “false”.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Optional</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>cpdtimeout</td>
<td>yes</td>
<td>Amount of time the CPD algorithms run, attempting to detect an event. If nothing is detected in this time period then the algorithm times out and the object returns “TIMEOUT”.</td>
</tr>
</tbody>
</table>

When this “com.avaya.ivr.cpdDetect” <object> is executed, it returns an ECAvaya MScript string as the value of the form item variable.

Typically, answering machine detection is enabled when an outbound call first connects. To do this, set the parameter `detectamd` to “true”. Upon completion the ECAvaya MScript string is set to “AMD” if an answering machine is detected, “VOICE” if voice is detected, otherwise it is set to “SILENCE”.

There are situations where the answering machine detected has not finished playing its greeting even though the object tag has returned and the form item variable was set to “AMD”. This typically occurs when the answering machine plays an initial greeting followed by a short pause and then the called party’s greeting. This type of greeting is common when dealing with wireless devices. The wireless carrier typically plays a greeting that the person you called is unavailable and then connects you to the called party’s voicemail. It is for this reason that the “persist” parameter was added and is used (set to “true”). When “persist” is set to “true” and the object tag returns “AMD” in its form item variable, answering machine detection continues. Should a second answering machine detection occur (for example when the wireless carrier transfers the call to the called party’s voice mail) the VXML interpreter throws the event “connection.cpd.amd” to the application. The VXML interpreter hard terminates (aborts) all speech output and input prior to delivering “connection.cpd.amd” to the application. The application, upon receiving the event, is responsible for either executing a new dialog or the same dialog.

⚠️ Important:

There is no hard rule for how an application deals with the “com.avaya.ivr.cpdDetect” object returning “SILENCE” in its form item variable. How to deal with “SILENCE” (treat as an answering machine or a real person) depends on the customers preference. The example below treats “SILENCE” the same as “VOICE”.

Example:

```
<object name="amdtest" classid="com.avaya.ivr.cpdDetect"> <param name="detectamd" value="true"></param> <param name="maxnospeech" value="1000"></param> <param name="persist" value="true"></param> <filled> <if cond="amdtest == ‘AMD’"> <goto next="#amd"/> </if> <else cond="amdtest == ‘VOICE’ || amdtest == ‘SILENCE’"> <goto next="#user"/> </else> <goto next="#error"/></filled> </object>
```

Typically fax and/or TDD detection is enabled when a inbound call first connects so that you can forward the call to another destination. To do this you set the parameter `detectfax` and `detectamd` to “true”. Upon completion the ECAvaya MScript string is set to “FAX” if a fax
machine is detected, “TDD” if a TDD is detected, or “TIMEOUT” if no device is detected in the specified time.

Example:

```xml
<object name="cpdtest" classid="com.avaya.ivr.cpdDetect">
  <param name="detecttdd" value="true"/>
  <param name="detectfax" value="true"/>
  <param name="cpdmaxtimeout" value="5000"/>
  <param name="persist" value="true"></param>
  <filled>
    <if cond="cpdtest == 'TDD'">
      <assign name="CPD" expr="'tdd'"/>
      <goto next="#nextform"/>
    </if>
    <elseif cond="cpdtest == 'FAX'">
      <assign name="CPD" expr="'fax'"/>
      <goto next="#nextform"/>
    </elseif>
    <elseif cond="cpdtest == 'TIMEOUT'">
      <assign name="CPD" expr="'timeout'"/>
      <goto next="#nextform"/>
    </elseif>
    <else>
      <assign name="CPD" expr="'unk'"/>
      <goto next="#nextform"/>
    </else>
  </filled>
</object>
```

Support for Nuance Extra Nbest Keys

Support is available for Nuance’s config parameter “swirec_extra_nbest_keys”. This parameter is used to add additional grammar keys to the XML result.

The configuration parameter “VXML use extra nbest keys for the recognition result” is used to configure the VXML interpreter to accept these additional grammar keys that the recognizer has added to the XML result.

The following represents a sample result returned from the Nuance recognizer when swirec_extra_nbest_keys is set to “SWI_meaning SWI_rawScore SWI_spoken SWI_ssmMeanings SWI_literalTimings”:

```xml
<?xml version='1.0'?> <result> <interpretation grammar="session:_gram1_1_7" confidence="96"> <input mode="speech">Wave File test</input> <instance>audio_server.vxml<SWI_spoken>Wave File test</SWI_spoken> <SWI_meaning>audio_server.vxml</SWI_meaning> <SWI_literalTimings> <alignment type="word" unit_msecs="1"> <word start="210" end="500" confidence="1.00">Wave</word> <word start="500" end="1110" confidence="1.00">File</word> <word start="1110" end="1610" confidence="1.00">test</word> </alignment> </SWI_literalTimings> <SWI_rawScore>-1623.725830</SWI_rawScore> </instance> </interpretation> </result>
```

In addition, here is a sample VXML application that dynamically sets “swirec_extra_nbest_keys” and logs the word scores returned in the result:

```xml
  <form id="Form_MainMenu">
    <property name="mrcp.Vendor-Specific-Parameters" value="swirec_extra_nbest_keys='SWI_meaning SWI_rawScore SWI_spoken SWI_ssmMeanings SWI_literalTimings'"/>
    <script><![CDATA[
    function logWordScores() {
    var length = Field_MainMenu.SWI_literalTimings.alignment.word.length;
    var i = 0; var ret = "";
    for (i = 0; i < length; i++) {
      ret += Field_MainMenu.SWI_literalTimings.alignment.word[i];
      ret += ":
    }
    
    </script>]]>
  </form>
</vxml>
```
Data Tag Attribute Update

VoiceXML applications can now post information using the <data> tag encoded as text/xml (setting enctype attribute to "text/xml"). The following encoding types are supported:

- application/x-www-form-urlencoded (This is the default if enctype is not specified)
- multipart/form-data
- text/xml

For example a VoiceXML developer could use text/xml encoding to post a SOAP request to a SOAP server directly as in the following example:

```xml
```

Finally, here is the sample output of the above application when the caller speaks “wave file test”:

```
EVENT[2]content=word info: Wave:[start:210,end:500,confidence:1.00];File:[start:500,end:1110,confidence:1.00];test:[start:1110,end:1610,confidence:1.00];
```

Refer to Nuance’s documentation for details regarding “swirec_extra_nbest_keys” and its impact on the XML result.
Adding HTTP headers to requests made using the data tag

A VoiceXML application can customize the HTTP request that is sent when using the VoiceXML data tag. Specifically, a VoiceXML application can specify additional headers be included in the HTTP request generated, when the <data> tag is executed.

The hints attribute of <data> is used for this purpose. The hints attribute is used to specify information used by the platform to customize the HTTP request. Currently, the only customization supported is the setting of HTTP headers. The value of this attribute equates to an object that contains an array property of "headers"; for instance:

```
<script> var DataHints = new Object(); DataHints.headers = new Object(); DataHints.headers['SOAPAction'] = ''; DataHints.headers['From'] = 'Me@avaya.com'; </script>
```

Avaya MS does not verify the headers being set, however Avaya MS does restrict the application from setting the following headers:

- cookie
- host
- user-agent
- referer
- connection
- if-modified-since
- if-none-match
- accept
- expect
- content-length
- content-type

Programming Example

This example sets the SOAPAction HTTP header:

```
<script> var DataHints = new Object(); DataHints.headers = new Object(); DataHints.headers['SOAPAction'] = ''; </script> ...
</data>
```

```xml
<SOAP-Envelope
```
Barge-In Notification

VoiceXML applications can determine if barge-in occurred or not during the last dialog.

This information is made available to the VoiceXML application as a shadow variable in the application.lastresult$.bargein, formitemvar$.bargein). The following are some examples of how to access this information:

```xml
<field name="Field_TouchTone"> <grammar src="builtin:dtmf/digits? minlength=1;maxlength=5"/> <prompt> Please enter one to five digits </prompt> <filled> <prompt> You entered <value expr="Field_TouchTone" class="digits" mode="recorded"/> </prompt> <log> Form Item bargein <value expr="Field_TouchTone$.bargein"></value></log> <log> lastresult bargein <value expr="application.lastresult$.bargein"></value></log> </filled> </field>
```

In the event of a nomatch this bargein information will only be available in application.lastresult $.bargein.

Send DTMF digits interface

Avaya MS supports the sending of DTMF digits over SIP based on the negotiated digit relay.

VoiceXML applications can use the `<audio>` tag with the reserved builtin string “builtin://mas/snddigit” for commanding the platform to send a sequence of DTMF digits. For example, the following VXML code transmits DTMF 2 followed by DTMF 0 followed by DTMF 1:

```xml
<audio src="builtin://Avaya MS/snddigit/201"></audio>
```

The sending of the DTMF digits behaves similar to the playing of prompts except that the transmission of the digits can always be interrupted by user input (for example, bargein is always on). One or more digits must be specified. The maximum number of digits that can be specified is 100. Currently only the DTMF digits found on a telephone keypad are supported (1, 2, 3, 4, 5, 6, 7, 8, 9, 0, *, and #).

Note: Though bargein may be on, a string of DTMF digits may not be barged in on if the user input happens after Avaya MS has sent the digits out via SIP. User input must be seen by Avaya MS prior to the application requesting to send a sequence of DTMF digits for that sequence of DTMF digits to be barged in on.
Continuous Streaming

- **Overview** on page 73
- **Directory Provider** on page 74
- **RSS Provider** on page 74
- **SHOUTCast Provider** on page 77
- **Application Programming** on page 77

**Overview**

Avaya MS supports continuous streaming of pre-transcoded real-time audio, which can be used by applications to facilitate a “radio broadcast” effect.

This feature allows applications to give sessions music on hold streaming or connect them to internet streaming radio servers. It is implemented efficiently to allow multiple sessions to “listen” to the same real-time audio stream without the cost of transcoding it on each session or connecting each session to a remote server.

Avaya MS is capable of streaming from the following providers: Directory, Really Simple Syndication (RSS), or SHOUTCast. These providers are described in detail in the next sections. The algorithm employed by Avaya MS for determining a source to stream from is as follows:

- If SHOUTCast is configured and the channel name (key) specified by the application matches the “SHOUTCast Channel Key” then Avaya MS will play from the primary or backup URL.
- If SHOUTCast is not configured, or the channel name doesn’t match, or the primary and backup URLs are not available then Avaya MS plays from the Directory provider.
- If RSS is configured then the RSS provider downloads the audio files into a location on disk and the Directory provider is used to play the audio. Prior to downloading the files RSS creates the directory on disk if it does not already exist, or empties the directory if it already exists.

The volume of the continuous stream can be adjusted by editing the “Audio Volume” parameter in EM under “Home > System Configuration > Media > General Settings” sub-menu “Continuous Streaming Sources”. The value should be specified in decibel (dB).
Directory Provider

The Directory Provider enables files in a local directory to be transcoded and cached, and played indefinitely in alphabetical order.

Continuous playback is achieved by repeating the sequence. Avaya MS will monitor the directory and detect any changes made for dynamic updates.

To configure the Directory Provider you will need to create a directory inside %BASEDIR%\platdata\StreamSource\ChannelRoot. This directory name is the channel name that is specified by the application to stream from this source. All audio files should be placed into this directory.

RSS Provider

The RSS Provider is capable of retrieving and parsing RSS (Real Simple Syndication) documents.

The contents of these documents are downloaded by the RSS Provider so they could be played by the Directory Provider. The RSS provider supports the following features:

• Automatic RSS feed synchronization allows content to be added and removed automatically.
• Fully supported time-to-live attribute facilitates updates to content.
• Fault tolerant solution will preserve local files until files are safely downloaded.
• MP3 and WAV content types.

RSS is a dialect of XML and Avaya MS currently supports RSS 2.0 (http://www.rss-specification.com/rss-2.0-specification.htm). Avaya MS is currently limited to RSS documents that are no larger than 260k. A sample RSS 2.0 document is shown below.
In order to enable the RSS provider you will need to configure the URL of the RSS document. This document is fetched when the platform is started or the URL is changed. The RSS provider will automatically add or delete content when the URL is changed. The URL is configured by using EM to edit the “Audio RSS URL” parameter under Home, System Configuration, Media Processing, General Settings sub-menu Continuous Streaming Sources and is shown below.
The following table lists additional RSS parameters as well as their location in EM:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio RSS Channel Key</td>
<td>Channel name associated with the RSS feed. The default is Audio. Parameter located in EM under “Home &gt; System Configuration &gt; Media &gt; General Settings” sub-menu “Continuous Streaming Sources”.</td>
</tr>
<tr>
<td>Allow One Item Per RSS Channel</td>
<td>Forces the RSS provider to only use the first channel element if the RSS document contains more than one channel elements. The default value is disabled and all channels will be used. Parameter located in EM under “Home &gt; System Configuration &gt; Advanced Settings &gt; Media Processing” sub-menu “Miscellaneous”</td>
</tr>
</tbody>
</table>
SHOUTCast Provider

SHOUTCast provider is capable of streaming audio content from SHOUTCast servers. SHOUTCast is a technology that uses MP3 encoding of audio content and HTTP as the transport protocol to broadcast internet radio.

The SHOUTCast provider supports receiving MP3 audio content or a play list (audio/x-scpl). If a play list is received the SHOUTCast provider will be redirected to the URL of the first file in the play list. Avaya MS currently does not support receiving metadata and will never request metadata by setting the Icy-Metadata tag to "0".

The URL of the SHOUTcast server is configured editing the “SHOUTCast Primary URL” parameter in EM under “Home > System Configuration > Media > General Settings” sub-menu “Continuous Streaming Sources”. An optional backup server can be configured, which will be used if the primary server is unreachable, by editing the “SHOUTCast Backup URL” parameter. If both the primary and backup URL is unreachable then streaming will fallback to the Directory Provider. These settings are shown in the "Continuous Streaming Configuration" figure above.

The following table lists additional SHOUTCast parameters, which are found in EM under “Home > System Configuration > Media > General Settings” sub-menu “Continuous Streaming Sources”, that may be modified:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHOUTCast Channel Key</td>
<td>Channel name associated with the SHOUTCast feed. The default is Audio.</td>
</tr>
</tbody>
</table>

Application Programming

A stream source play is initiated in VoiceXML using the <audio> tag with a reserved builtin format “builtin://mas/ivrmp”.

Syntax: <audio src="builtin://mas/ivrmp/key.lss"/> where key is to be replaced with one of the following:

- Directory name containing the audio files for “Directory Provider” continuous streaming.
- The “Audio RSS Channel Key” for RSS Provider continuous streaming.
- The “SHOUTCast Channel Key” for SHOUTCast Provider continuous streaming.

Example: <audio src="builtin://mas/ivrmp/Audio.lss"/>

Note: This feature is not available for use with the VCR controls object tag.
Delay

A mechanism to cause a delay in the execution of a VoiceXML script can be achieved via the VXML <object> tag whose classid is “com.avaya.ivr.delay”.

The delay object tag is described as follows:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Supported as defined by the specification.</td>
</tr>
<tr>
<td>expr</td>
<td>Supported as defined by the specification.</td>
</tr>
<tr>
<td>cond</td>
<td>Supported as defined by the specification.</td>
</tr>
<tr>
<td>classid</td>
<td>Configure as com.avaya.ivr.delay to delay the execution of the VoiceXML script.</td>
</tr>
<tr>
<td>codebase</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>codetype</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>data</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>type</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>archive</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>fetchhint</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>fetchtimeout</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>maxage</td>
<td>Not used and ignored.</td>
</tr>
<tr>
<td>maxstale</td>
<td>Not used and ignored.</td>
</tr>
</tbody>
</table>

The following values are passed using the <param> element.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay</td>
<td>no</td>
<td>The duration to delay, in milliseconds, the VoiceXML script. If this parameter is not specified or is out of range error.semantic will be thrown. Range: delay &gt;= 1ms; delay &lt;= 3600000ms (1 hour)</td>
</tr>
</tbody>
</table>

When this “com.avaya.ivr.delay” <object> is executed it suspends the execution of the VoiceXML script and, on completion, returns an ECAvaya MScript string as the value of the form item variable. This ECAvaya MScript string will be set to “SUCCESS”.

Example: Delay of 500 milliseconds
Outbound dialer

This web service takes as input a destination, a source, and the application to invoke (among other optional arguments). The web service request causes the platform to make an outbound call to the destination and invoke the application (indicated by the Alias) when the call connects. The request is called **InvokeApplication** and is defined in the Avaya MS Web Service Application API WSDL ([http://<IP_address_of_MAS>/maswsappapi.wsdl](http://<IP_address_of_MAS>/maswsappapi.wsdl)). Web services are invoked by sending a soap request to the Soap Server on Avaya MS ([http://localhost:80/soap](http://localhost:80/soap)).

The arguments supported by the **InvokeApplication** web service are described as follows:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationAlias</td>
<td>Name of the VXML application configured on Avaya MS (translation is not required) that is invoked when the call connects, or &quot;interp::vxml&quot; if required to invoke the VXML application using the ApplicationUrl.</td>
<td>yes</td>
</tr>
<tr>
<td>ApplicationUrl</td>
<td>URL of the VXML application to invoke. This parameter is required only when ApplicationAlias is set to &quot;interp::vxml&quot;.</td>
<td>no</td>
</tr>
<tr>
<td>Domain</td>
<td>Specifies the default domain to use if the domain is not specified in the &quot;To&quot; and/or &quot;From&quot; parameters.</td>
<td>no</td>
</tr>
<tr>
<td>Locale</td>
<td>Specifies language/region.</td>
<td>no</td>
</tr>
<tr>
<td>To</td>
<td>The &quot;logical&quot; recipient of the request. Used to set the SIP To header in the outgoing invite.</td>
<td>yes</td>
</tr>
<tr>
<td>From</td>
<td>The &quot;logical&quot; identity of the initiator of the request. Used to set the SIP From header in the outgoing invite.</td>
<td>yes</td>
</tr>
<tr>
<td>Opaque</td>
<td>This parameter is optional and is ignored by VXML applications.</td>
<td>no</td>
</tr>
</tbody>
</table>

The following represents a sample request that causes the platform to originate a call to "johndoe" and invoke the "numdemo" application once the call connects.

```xml
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:v1="http://www.nortel.com/xmlprotocol/wsdI/session_control/applications/invoe/
```
On completion (connection of the outbound call) the SOAP server responds with the session ID of the new outbound call.

VXML programming example

The following is an example of an outbound dialer web service request using VXML.

```xml
  xmlns:soapenv:Header="" xmlns:soapenv:Body="">
  <soapenv:Header/>
  <soapenv:Body>
    <v1:InvokeApplication>
      <ApplicationAlias>numdemo</ApplicationAlias>
      <Domain>mydomain.com</Domain>
      <To>johndoe</To>
      <From>janedoe</From>
    </v1:InvokeApplication>
  </soapenv:Body>
</soapenv:Envelope>
```

Content Store Built-in URL

VoiceXML applications have the ability to play media files (audio, video, etc) uploaded into the Avaya MS Content Store using Element Manager's media management tool. Media content is referenced in the `<audio>` tag using a built-in URL with the following syntax:

```
builtin://mas/cstore/contentgrp/contentid?ns=namespace
```

<table>
<thead>
<tr>
<th>contentgrp</th>
<th>Content group containing the media file to play</th>
</tr>
</thead>
<tbody>
<tr>
<td>contentid</td>
<td>Content identifier that identifies the media file to play</td>
</tr>
<tr>
<td>namespace</td>
<td>Namespace of the content group and content identifier</td>
</tr>
</tbody>
</table>

As an example, the following `<audio>` tag plays the media file called `july.wav` in the content group `months` of the `calendar` namespace.

```
<audio src="builtin://mas/cstore/months/july.wav?ns=calendar"/>
```
Chapter 6: Coding elements

This section describes differences between CCXML and VoiceXML used in the Avaya MS environment and those recommended by W3C.

Navigation

- CCXML compliancy on page 81
- VoiceXML compliancy on page 84
- JavaScript Extensions for CCXML and VoiceXML on page 87

CCXML compliancy

This section describes how the Avaya MS release of CCXML varies from the W3C release of CCXML 1.0.

Document control flow and execution

<table>
<thead>
<tr>
<th>CCXML tag</th>
<th>W3C Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccxml</td>
<td>Variations</td>
</tr>
<tr>
<td></td>
<td>• xml:base: Must be supported by javascript</td>
</tr>
<tr>
<td></td>
<td>• method</td>
</tr>
<tr>
<td>fetch</td>
<td>Variations</td>
</tr>
<tr>
<td></td>
<td>• application/ccxml+xml: (default) is supported</td>
</tr>
<tr>
<td></td>
<td>• text/ecmascript:</td>
</tr>
<tr>
<td></td>
<td>• text/javascript:</td>
</tr>
<tr>
<td></td>
<td>Unsupported Attributes</td>
</tr>
<tr>
<td></td>
<td>• maxage</td>
</tr>
<tr>
<td></td>
<td>• maxstale</td>
</tr>
<tr>
<td></td>
<td>• enctype</td>
</tr>
</tbody>
</table>
### Coding elements

<table>
<thead>
<tr>
<th>CCXML tag</th>
<th>W3C Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>createccxml</td>
<td>Variations</td>
</tr>
<tr>
<td></td>
<td>method:</td>
</tr>
<tr>
<td></td>
<td>Unsupported Attributes</td>
</tr>
<tr>
<td></td>
<td>• namelist - use the &lt;send&gt; tag to transfer information</td>
</tr>
<tr>
<td></td>
<td>• fetchparam</td>
</tr>
<tr>
<td></td>
<td>• parameters</td>
</tr>
<tr>
<td></td>
<td>• maxage</td>
</tr>
<tr>
<td></td>
<td>• maxstale</td>
</tr>
<tr>
<td></td>
<td>• enctype</td>
</tr>
<tr>
<td>meta, metadata, if, elseif, else, goto, exit, log</td>
<td>Fully supported</td>
</tr>
</tbody>
</table>

### Dialog elements

<table>
<thead>
<tr>
<th>CCXML tag</th>
<th>W3C Non-Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialogprepare</td>
<td>Not supported in the Avaya MS implementation of CCXML</td>
</tr>
<tr>
<td>dialogstart</td>
<td>Variations</td>
</tr>
<tr>
<td></td>
<td>• type: Limited support - only the application/voicexml+xml is supported.</td>
</tr>
<tr>
<td></td>
<td>• method:</td>
</tr>
<tr>
<td></td>
<td>Unsupported Attributes</td>
</tr>
<tr>
<td></td>
<td>• parameters</td>
</tr>
<tr>
<td></td>
<td>• conferenceid</td>
</tr>
<tr>
<td></td>
<td>• mediadirection</td>
</tr>
<tr>
<td></td>
<td>• maxage</td>
</tr>
<tr>
<td></td>
<td>• maxstale</td>
</tr>
<tr>
<td></td>
<td>• enctype</td>
</tr>
<tr>
<td>dialogterminate</td>
<td>Unsupported Attributes</td>
</tr>
<tr>
<td></td>
<td>hints</td>
</tr>
</tbody>
</table>
## Variables and expression elements

<table>
<thead>
<tr>
<th>CCXML tag</th>
<th>W3C Non-Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>script</strong></td>
<td>Variations</td>
</tr>
<tr>
<td></td>
<td>fetchid:</td>
</tr>
<tr>
<td></td>
<td>Unsupported Attributes</td>
</tr>
<tr>
<td></td>
<td>• maxage</td>
</tr>
<tr>
<td></td>
<td>• maxstale</td>
</tr>
<tr>
<td></td>
<td>• charset</td>
</tr>
<tr>
<td><strong>assign, var</strong></td>
<td>Fully supported</td>
</tr>
</tbody>
</table>

## Event handling

<table>
<thead>
<tr>
<th>CCXML tag</th>
<th>W3C Non-Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>send</strong></td>
<td>Variations</td>
</tr>
<tr>
<td></td>
<td>targettype:</td>
</tr>
<tr>
<td></td>
<td>Dependant on</td>
</tr>
<tr>
<td></td>
<td>implementation and</td>
</tr>
<tr>
<td></td>
<td>support for sending</td>
</tr>
<tr>
<td></td>
<td>messages to the</td>
</tr>
<tr>
<td></td>
<td>dialog (VXML) and</td>
</tr>
<tr>
<td></td>
<td>a HTTP Event</td>
</tr>
<tr>
<td></td>
<td>processor.</td>
</tr>
<tr>
<td></td>
<td>Unsupported Attributes</td>
</tr>
<tr>
<td></td>
<td>hints</td>
</tr>
<tr>
<td><strong>move</strong></td>
<td>Unsupported Attributes</td>
</tr>
<tr>
<td></td>
<td>hints</td>
</tr>
<tr>
<td><strong>eventprocessor, transition, cancel</strong></td>
<td>Fully supported</td>
</tr>
</tbody>
</table>
Telephony/Operations and resources

<table>
<thead>
<tr>
<th>CCXML tag</th>
<th>W3C Non-Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept, redirect, reject, destroyconference,</td>
<td>Unsupported Attributes</td>
</tr>
<tr>
<td>unjoin, disconnect, merge</td>
<td>hints</td>
</tr>
<tr>
<td>createcall</td>
<td>Unsupported Attributes</td>
</tr>
<tr>
<td></td>
<td>• hints</td>
</tr>
<tr>
<td></td>
<td>• joinid</td>
</tr>
<tr>
<td></td>
<td>• joindirection</td>
</tr>
<tr>
<td>createconference</td>
<td>Unsupported Attributes</td>
</tr>
<tr>
<td></td>
<td>• hints</td>
</tr>
<tr>
<td></td>
<td>• confname</td>
</tr>
<tr>
<td>join</td>
<td>Variations</td>
</tr>
<tr>
<td></td>
<td>• duplex: Requires support from platform. CCXML will only support a half duplex connection on a bridge or conference if the Avaya MS platform supports a half duplex connection.</td>
</tr>
<tr>
<td></td>
<td>Unsupported Attributes</td>
</tr>
<tr>
<td></td>
<td>• hints</td>
</tr>
<tr>
<td></td>
<td>• enter-tone</td>
</tr>
<tr>
<td></td>
<td>• exit-tone</td>
</tr>
<tr>
<td></td>
<td>• autoinputgain</td>
</tr>
<tr>
<td></td>
<td>• autooutputgain</td>
</tr>
<tr>
<td></td>
<td>• dtmfclamp</td>
</tr>
<tr>
<td></td>
<td>• toneclamp</td>
</tr>
<tr>
<td>cancel</td>
<td>Fully supported</td>
</tr>
</tbody>
</table>

VoiceXML compliancy

This section describes how the Avaya MS release of VoiceXML varies from the W3C release of VoiceXML 2.0.
Control flow and scripting

<table>
<thead>
<tr>
<th>VoiceXML tag</th>
<th>W3C Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>disconnect</td>
<td>Variations namelist: New attribute</td>
</tr>
<tr>
<td>script</td>
<td>Variations srcexpr: Scripts can be referenced dynamically via the srcexpr attribute</td>
</tr>
<tr>
<td>var, assign, clear, if, else, elseif, goto, submit, exit, return, log</td>
<td>Fully supported</td>
</tr>
</tbody>
</table>

User input

<table>
<thead>
<tr>
<th>VoiceXML tag</th>
<th>W3C Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>grammar</td>
<td>Variations srcexpr: New attribute - allows grammars to be referenced dynamically</td>
</tr>
<tr>
<td>meta, metadata, lexicon, rule, token, ruleref, item, one-of, example, tag</td>
<td>Fully supported</td>
</tr>
</tbody>
</table>

System output

The prompt and value tags are fully supported.

SIP Data Access

This section describes the SIP session variables used in VoiceXML 2.1.
**session.connection.local.uri**

Support for the `session.connection.local.uri` session variables in VoiceXML 2.1 is shown in the table below.

- **Supported**: Evaluates to the SIP URI specified in the To: header of the initial INVITE.
- **Not Supported**: Evaluates to the SIP URI specified in the To: header of the initial REFER.

**session.connection.remote.uri**

Support for the `session.connection.remote.uri` session variables in VoiceXML 2.1 is shown in the table below.

- **Supported**: Evaluates to the SIP URI specified in the From: header of the initial INVITE.
- **Not Supported**: Evaluates to the SIP URI specified in the From: header of the initial REFER.

**session.connection.redirect**

Populated by information contained in the History-Info header in the initial INVITE or is otherwise undefined.

Properties of each element of the array are determined as follows:

- **uri**: Set to the hi-targeted-to-uri value of the History-Info entry
- **pi**: Set to true if hi-targeted-to-uri contains a Privacy=history parameter, or if the INVITE Privacy header includes history; false otherwise
- **si**: Set to the value of the si parameter if it exists, undefined otherwise
- **reason**: Set verbatim to the value of the Reason parameter of hi-targeted-to-uri

**session.connection.protocol.name**

Evaluates to sip.

This is intended to reflect the use of SIP in general, and does not distinguish between whether the media server was accessed via SIP or SIPS procedures.

**session.connection.protocol.version**

Evaluates to 2.0.

**session.connection.protocol.sip.headers**

This is an associative array where each key in the array is the non-compact name of a SIP header in the initial INVITE converted to lower-case (note the case conversion does not apply to the header value).

If multiple header fields of the same field name are present, the values are combined into a single comma-separated value. Implementations MUST at a minimum include the Call-ID header and MAY include other headers.

For example, `session.connection.protocol.sip.headers["call-id"]` evaluates to the Call-ID of the SIP dialog.
session.connection.protocol.sip.requesturi
This is an associative array where the array keys and values are formed from the URI parameters on the SIP Request-URI of the initial INVITE (or REFER).

The array key is the URI parameter name. The corresponding array value is derived from the URI parameter value according to the following rules:

- If the URI parameter name is an init-param or dialog-param, the corresponding array value is obtained by evaluating the URI parameter value as a string
- If the URI parameter name is a vxml-param, the corresponding array value is obtained by evaluating the URI parameter value as a JSON value
- If the URI parameter name is present but its value is omitted, the value is an empty string

In addition, the array's toString() function returns the full SIP Request-URI.

session.connection.aai
Evaluates to: session.connection.protocol.sip.requesturi["aai"]

session.connection.ccxml
Evaluates to: session.connection.protocol.sip.requesturi["ccxml"]

session.connection.protocol.sip.media
The session.connection.protocol.sip.media is not supported in VoiceXML 2.1. It is an array where each array element is an object with the following properties:

- type - This required property indicates the type of the media associated with the stream. The value is a string. It is strongly recommended that the following values are used for common types of media: "audio" for audio media, and "video" for video media.
- direction - This required property indicates the direction of the media relative to session.connection.originator. Defined values are sendrecv, sendonly, recvonly, and inactive.
- format - This property is optional. If defined, the value of the property is an array. Each array element is an object which specifies information about one format of the media (there is an array element for each payload type on the m-line). The object contains at least one property called name whose value is the MIME subtype of the media format (MIME subtypes are registered in [RFC4855]). Other properties may be defined with string values; these correspond to required and, if defined, optional parameters of the format.

As a consequence of this definition, there is an array entry in session.connection.protocol.sip.media for each non-disabled m-line for the negotiated media session. Note that this session variable is updated if the media session characteristics for the VoiceXML Session change (for example, due to a re-INVITE).

JavaScript Extensions for CCXML and VoiceXML

The CCXML and VoiceXML interpreters provide a number of extensions accessible from JavaScript for the script writer.
Logging

The log tag is the standard mechanism for logging; however, it cannot be used from JavaScript inside a script tag. The print function is provided for this purpose.

`print(msg)` msg – string to output to log file.

Logs from the print function have a severity of "info".

Example:

```xml
<script> print("Greetings to all the fish!\n"); </script>
```

Log object

The log object gives JavaScript code control over logging.

<table>
<thead>
<tr>
<th>Log object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>log.id</td>
<td>log Global session identifier (GSLID)</td>
</tr>
<tr>
<td>log.enable</td>
<td>enable logging</td>
</tr>
<tr>
<td>log.trace</td>
<td>enable CCXML Document Trace</td>
</tr>
</tbody>
</table>

When `log.trace` is enabled the execution of each CCXML tag will be written to `ccxmlApp.txt`.

Note: Enabling trace for all CCXML documents may have an impact on system performance.

Logs are written to text files that reside in the "common\logs" subdirectory of the Avaya MS install directory (for example: D:\Program Files\avaya\Multimedia_Applications\Avaya MS \common\logs). The interpreters have two log files each:

- `ccxmlDebug.txt`, `vxmliDebug.txt` – logs from the interpreter.
- `ccxmlAppDebug.txt`, `vxmliAppDebug.txt` – logs from applications, logs produced by the log tag and the print function.

Application logging is normally controlled for all sessions using EAM. The log object provides an additional mechanism with finer control. Setting `log.enable` to "true" turns on application logging for the session. Setting it to "false" causes all subsequent application logs to be discarded. This can be used to implement highly selective logging, for example, turning on logging only for specific callers or only during certain phases of a session.

Example:

```xml
<script> print("Hello World!\n"); log.enable="false"; print("This debug is not written!\n"); log.enable="true"; print("This debug is written out to the log file.\n"); </script>
```
Event Logging

The logevent function is used to send an event to the platform. These events are viewable from EM. Additionally Avaya MS integrates with a variety of open protocols (SNMPv1, SNMPv2c) for monitoring and management of events.

logevent(severity, evno, msg, [data])

severity – Must be one of the following: Event.INFO, Event.WARNING, or Event.ERROR

evno – relative event number. The relative event number is converted to an actual event number by adding 50000.

Important:
Care must be taken when selecting relative event numbers to avoid conflicts between CCXML and VXML applications.

msg – string containing event message

Example (the event generated has an event number of 50005):

<catch event=error.noresource.lvr> <script> logevent(Event.ERROR,5,"Error obtaining an LVR resource"); </script> <audio src="builtin:VXML-SERIOUS-ERROR"/> </catch>

Counters and Gauges

Counters and gauges are locations maintained by the system that are capable of holding one integer apiece:

- Counters can be incremented and decremented. They are periodically (default: 15 minutes) reset to zero by the system.
- Gauges can be set to any value. They are not automatically reset.

Applications (CCXML and VoiceXML documents) access counters and gauges by name. System management tools (EAM) show the value stored inside counters and gauges against their title.

The system comes pre-configured with a number of counters and gauges to report on the operation of its various components. Applications may dynamically define their own counters and gauges by using the createcounter and creategauge functions.

createcounter(name, title, description) creategauge(name, title, description)

name – JavaScript name the application will use to access the counter/gauge. title – display name
description – string explaining purpose of counter/gauge return value – 0: counter/gauge created successfully; -2: counter/gauge already exists; any other value: counter/gauge could not be created.
When two documents under the same application engine, for example, CCXML documents A and B, both create a counter or gauge with the same name, they end up using the same object.

Once a counter has been created, it can be accessed as a field in the counters object.

Example:

```xml
<transition event="ccxml.loaded"> <script> createcounter("nactive","avg active calls A","average number of active calls in app A"); </script> </transition>

<transition event="connection.connected"> <script> counters.nactive++; </script> </transition>

<transition event="connection.disconnected"> <script> counters.nactive--; </script> </transition>
```

Similarly, once a gauge has been created, it can be accessed as a field in the gauges object. Gauges behave like normal variables: they can be read and written with any value from JavaScript. In contrast, counters can only have a (signed) delta added; they always read back as zero.

Example:

```xml
<transition event="ccxml.loaded"> <script> if (!creategauge("ncalls","total calls A","total number of calls handled by App A")) gauges.ncalls=0; else gauges.ncalls++; </script> </transition>
```

Note: If a counter or gauge is used by an application in a call to print() the value displayed in the log will not necessarily match the value held by the system statistic. This is due to applications that update counters/gauges of the same name accessing the same exact counter/gauge in the system, and the feature where counters are reset, by the system, to zero on a periodic basis.

---

### Session detail records

In session detail recording (SDR), applications and system components contribute records to a running log associated with a Global Session Identifier (GSLID). For example, an incoming call might be routed to a CCXML document, which then starts a VoiceXML document. The system would assign a GSLID to identify the call. As the CCXML and VoiceXML documents execute, records are logged against the GSLID.

The system periodically gathers all the records associated with a GSLID and writes it to the database as a pair of XML documents. This happens within a few seconds of the GSLID becoming inactive. A GSLID becomes inactive when its associated object goes away, such as a connection object disappears when its call hangs up.
Every SDR is an instance of a reporting field. A reporting field has a name and may hold one of the following types of data:

- Integer
- String
- OM integer

The installSDRField function defines a reporting field.

installSDRField(name, type, title, description) name – JavaScript name the application will use to access the SDR reporting field. type – SDR.INTEGER, SDR.STRING, or SDR.OM title – Name to be used when this SDR reporting field is displayed in EM and reports. description – string explaining purpose of reporting field return value – 1: success; 0: failure

The first time that installSDRField is called for a given name, the definition of the reporting field is saved in the system database. This persists across multiple sessions, until it is removed by a call to the uninstallSDRField function.

uninstallSDRField(name) name – JavaScript name of the reporting field to uninstall.

To emit SDRs against a GSLID, first create an SDR object for it

SDR object

SDR(gslid) -- constructor gslid – GSLID

update(name, value) – update reporting field name – JavaScript name of the reporting field to update. value – value

In CCXML, session, connection, conference, and dialog objects are sources of GSLIDs. In VXML, the session variable session.com.avaya.ivr.dialog.id is the GSLID source.

Reporting fields appear as fields of the SDR object. Assigning to a reporting field produces a record.

Example:

<var name="sdr" expr="new SDR(log.id)"/> ...

<transition event="ccxml.loaded"> <script> installSDRField("ev", SDR.STRING,"event","call event"); installSDRField("ndialogs", SDR.OM,"number of dialogs","number of dialogs during call"); </script> </transition>

<transition event="connection.connected"> <script> sdr.ev="connected"; </script> </transition>

<transition event="connection.disconnected"> <script> sdr.ev="disconnected"; </script> </transition>

<transition event="dialog.started"> <script> sdr.ndialogs+=1; </script> </transition>

A GSLID can collect multiple records of string and integer reporting fields with the same name. For example, the above script would produce an ev record with the value "connected" when a call comes in, and a second ev record with the value "disconnected" when the call hangs up.
On the other hand, a GSLID maintains a single record of an OM integer reporting field with a given name. For example, every time a dialog is started in the above script, the value in the ndialogs record is incremented.

The SDR object's update method can be used to alter the value of an existing record.

Example:

```xml
<var name="sdr" expr="new SDR(log.id)"/> ... 
<transition event="ccxml.loaded"> <script>
installSDRField("outcome",SDR.STRING,"outcome","our best guess"); sdr.outcome="none";
</script> </transition>

<transition event="app.success"> <script> sdr.update("outcome","sale"); </script> </transition>

<transition event="app.failure"> <script> sdr.update("outcome","no sale"); </script> </transition>

<transition event="app.maybe"> <script> sdr.update("outcome","follow up"); </script> </transition>
```

This script maintains a single record for the outcome field. The record is created and initialized to the value "none". As more information becomes available, the update method is used to change its value. Had field assignment been used instead, additional outcome records would have been created.

Please note that:

- There must be at least one record of a reporting field before it can be updated.
- If there is more than one, then all of them will be updated.

---

**SDR reporting fields application association**

By default a VoiceXML application’s reporting fields are associated with the application that is invoked when a call lands on the platform. As an example, assume the following applications and translations.
Figure 2: Sample Custom Applications

<table>
<thead>
<tr>
<th>Application Name</th>
<th>Status</th>
<th>State</th>
<th>Version</th>
<th>Interpreter Type</th>
<th>App Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>account</td>
<td>AVAILABLE</td>
<td>UNLOCKED</td>
<td>1.0</td>
<td>VoiceXML</td>
<td>URL</td>
</tr>
<tr>
<td>ccxml_answer</td>
<td>AVAILABLE</td>
<td>UNLOCKED</td>
<td>1.0</td>
<td>CCXML</td>
<td>URL</td>
</tr>
<tr>
<td>ccs</td>
<td>AVAILABLE</td>
<td>UNLOCKED</td>
<td>1.0</td>
<td>VoiceXML</td>
<td>URL</td>
</tr>
<tr>
<td>LandingPad</td>
<td>AVAILABLE</td>
<td>UNLOCKED</td>
<td>1.0</td>
<td>VoiceXML</td>
<td>Web Archive</td>
</tr>
<tr>
<td>cnumdms</td>
<td>AVAILABLE</td>
<td>UNLOCKED</td>
<td>1.0</td>
<td>VoiceXML</td>
<td>URL</td>
</tr>
</tbody>
</table>

Figure 3: Sample Signaling Translations

Also assume the ccxml_answer application contains the following code:
A call landing on the platform with the SIP URL numdemo@47.185.23.33 invokes the numdemo application and the SDR reporting fields maintained by the numdemo VXML application are associated with numdemo. A call landing on the platform with the SIP URL ccxml_answer@47.185.23.33 invokes the ccxml_answer application which then starts the account.vxml VoiceXML dialog. The SDR reporting fields maintained by the account VXML application are associated with the ccxml_answer application by default, not the account VXML application. This is because the ccxml_answer application was invoked by the platform when the call landed on it.

VoiceXML applications can override this default behavior. An application can explicitly set the application name it wants its SDR reporting fields to be associated with. This is accomplished by setting appid in Avaya’s JavaScript log object to the name of the application to associate the SDR reporting fields with. For example, the following is done in account.vxml to associate its reporting fields with the account application instead of the ccxml_answer application:

```xml
<var name="sdr" expr="new SDR(session.com.avaya.ivr.dialog.id)"/>
<script>
  log.appid = "account";
  installSDRField("NbrOfCalls",SDR.INTEGER,"Number Of Calls","integer sdr");
  installSDRField("Event",SDR.STRING,"Error Event Received","string sdr");
  installSDRField("NbrNoMatches",SDR.OM,"vxml nomatch om sdr","om sdr");
</script>
```

An entry for the application name used to set log.appid must be pre-configured in Element Manager to successfully override the default associations. The applications configured in Figure 2: Sample Custom Applications on page 93 and the translations configured in Figure 3: Sample Signaling Translations on page 93 reveal that a translation was not defined for the VXML application account, this was intentional. The account application was configured in Element Manager strictly for SDR reporting and association purposes. The VoiceXML account application does not need to be configured if it does not override the default SDR reporting associations.

### Cluster object

The cluster object reports on the state of a clustered system.

<table>
<thead>
<tr>
<th>Cluster object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cluster.ip</td>
<td>IP address of local node</td>
</tr>
<tr>
<td>cluster.hostname</td>
<td>hostname of local node</td>
</tr>
<tr>
<td>cluster.uuid</td>
<td>Universally Unique Identifier (UUID) of the local node</td>
</tr>
<tr>
<td>cluster.primary</td>
<td>IP address of primary node (&quot;Standalone&quot; if one is not configured)</td>
</tr>
<tr>
<td>cluster.secondary</td>
<td>IP address of secondary node (&quot;undefined&quot; if one is not configured)</td>
</tr>
</tbody>
</table>
As the state of the cluster changes over time, so do the cluster object's properties. If there is no node acting in a secondary role, the corresponding property returns the JavaScript value undefined.

The getrole function determines the role a particular node plays in a cluster.

getrole(ipaddr) ipaddr – IP address of the node return value – unknown, primary, secondary, or standard

The getuuid function determines the UUID of a particular node in a cluster.

getuuuid(ipaddr) ipaddr – IP address of the node return value – 00000000-0000-0000-0000-000000000000 or a valid UUID string

Example:

<catch event="error.transfer"> <log> Received error.transfer on <value expr="'host:' + cluster.hostname + ' UUID:' + getuuid(cluster.ip)"/></log> <if cond="getrole(cluster.ip) == 'primary'"> …do something special… <elseif cond="getrole(cluster.ip) == 'secondary'"> …do something even more special… <else/> …do something ordinary… </if> </catch>
Chapter 7: Integrate CCXML and VoiceXML

CCXML and VoiceXML applications work together in providing an enhanced user experience. CCXML complements and integrates with VoiceXML by providing telephony call control support. This section describes the interaction between VoiceXML and CCXML applications.

Navigation

- Integration summary on page 97
- CCXML - dialogstart on page 98
- CCXML - dialogterminate on page 98
- VXML - object on page 98
- VXML - exit on page 98
- VXML - disconnect on page 99
- VXML - transfer on page 99

Integration summary

VoiceXML applications can integrate and coexist with CCXML applications. Elements in both CCXML and VoiceXML applications throw events which are received and handled by the receiving application.

See Dialog elements on page 82 for more information on the all elements associated with CCXML and VoiceXML integration.
**CCXML - dialogstart**

The CCXML application uses the `<dialogstart>` element to start a dialog. Use the following attributes to start the dialog:

- namelist - Use to pass data from the CCXML application to the VoiceXML dialog
- src - Use to identify the URI which stores all dialog variables and assigned values

All variables are made available to the VoiceXML application in the session variable com.avaya.ivr.appaccept.appdata.

**CCXML - dialogterminate**

A CCXML application can terminate a dialog using the `<dialogterminate>` element. Use the following attributes to terminate the dialog:

- dialogid - Use to identify the dialog to terminate
- immediate - Set to True to have the dialog terminated immediately - otherwise the dialog terminates in a normal fashion

**VXML - object**

A VoiceXML dialog using the object com.avaya.ivr.dialog.user can generate a user event, which is sent to the CCXML application.

A param tag with the name="event" attribute identifies the name of the user event and must always be present. All variables in the param tag are passed to the CCXML application.

A CCXML application event handler is required and must be developed.

**VXML - exit**

A VoiceXML dialog terminates when it executes the `<exit>` element or when it does not specify a successor dialog. The CCXML application receives an event dialog.exit.
The VoiceXML dialog can return data to the CCXML application by using the expr or namelist attribute of the <exit> element. These variables are made available to the CCXML application in the event values field.

### VXML - disconnect

The <disconnect> tag includes a namelist attribute, which is used to specify the variable names that are to be returned to a CCXML application via an ECAvaya MScript object.

### Setting SIP headers in bye

A VoiceXML application can customize the SIP request that is sent to the network to terminate a session, specifically, the SIP outgoing bye. The *hints* attribute of <disconnect> is used for this purpose.

The *hints* attribute specifies information used by the platform to customize the disconnect request. Currently, the only customization supported is the setting of SIP headers. The value of this attribute equates to an object that contains an array property of "headers"; for instance:

```javascript
<script>
    var hints = new Object();
    DiscHints.headers = new Object();
    DiscHints.headers['Reason'] = 'SIP ;cause=580 ;text="Precondition Failure"';
</script>
```

**Programming Example**

This example sets the Reason SIP headers.

```xml
... <script>
    var hints = new Object();
    DiscHints.headers = new Object();
    DiscHints.headers['Reason'] = 'SIP ;cause=580 ;text="Precondition Failure"';
</script> ...
</form>
```

### VXML - transfer

The <transfer> tag includes a type attribute, which is used to specify the type of transfer to be performed. Valid types are bridge, blind, and consultation.

An event handler must be developed in the CCXML application to manage any transfers sent from the VoiceXML application.

The VXML interpreter will, in a CCXML & VXML solution, use the CCXML interpreter to perform the transfer. In this case a CCXML application is required to be written to handle and execute the VXML applications transfer request.
The VXML object tag that can be used to send events to the CCXML application:
com.avaya.ivr.dialog.user

Setting SIP headers during transfer

A VoiceXML application can customize the SIP request that is sent to the network to initiate a transfer. Specifically the SIP outgoing invite (for bridge and two session consultation transfers) and the SIP outgoing refer (for blind and single session consultation transfers). The hints attribute of <transfer> is used for this purpose.

The hints attribute specifies information used by the platform to customize the transfer request. Currently, the only customization supported is the setting of SIP headers. The value of this attribute equates to an object that contains an array property of "headers"; for instance:

<script> var XferHints = new Object(); XferHints.headers = new Object();
XferHints.headers['Subject'] = 'Testing SIP Header Data'; XferHints.headers['User-to-User'] = 'Hello World'; XferHints.headers['Organization'] = 'Avaya'; </script>

Programming Example

This example sets the Subject, User-to-User, and Organization SIP headers:

... <var name="desturi" expr="'johndoe@avaya.com'"/>
<script> var XferHints = new Object(); XferHints.headers = new Object(); XferHints.headers['Subject'] = 'Testing SIP Header Data'; XferHints.headers['User-to-User'] = 'Hello World'; XferHints.headers['Organization'] = 'Avaya'; </script> ... <transfer name="mycall" destexpr="sip:+desturi" connecttimeout="60s" bridge="false" maxtime="20s" hints="XferHints"> ...
Chapter 8: Event handling

Events are triggers for actions in CCXML and VoiceXML applications.

Navigation

- CCXML events on page 101
- VoiceXML events on page 105

CCXML events

Events arrive from outside the CCXML application from the underlying telephony platform or from other sources including a dialog, conference, or external control.

An event handler must be developed and included in the CCXML application. CCXML processes events using an Event Handler Interpretation Algorithm (EHIA), which includes two components:

- `<eventprocessor>` - Defines a state variable indicating current state of the application.
- `<transition>` - Defines the processing required based on the current state and input event.

Connection events

<table>
<thead>
<tr>
<th>Event</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection.accept.failed</td>
<td>Asynchronous event</td>
</tr>
<tr>
<td>connection.alerting</td>
<td>Asynchronous event</td>
</tr>
<tr>
<td>connection.connected</td>
<td>Confirmation of a <code>&lt;accept&gt;</code> or <code>&lt;createcall&gt;</code> element</td>
</tr>
<tr>
<td>connection.disconnected</td>
<td>Confirmation of a <code>&lt;disconnect&gt;</code> element or an asynchronous event</td>
</tr>
<tr>
<td>connection.failed</td>
<td>Any call control tag or asynchronous event</td>
</tr>
<tr>
<td>connection.merge.failed</td>
<td>Failure indication for a <code>&lt;merge&gt;</code> element</td>
</tr>
</tbody>
</table>
### Event handling

<table>
<thead>
<tr>
<th>Event</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection.merged</td>
<td>Confirmation of a <code>&lt;merge&gt;</code> element</td>
</tr>
<tr>
<td>connection.progressing</td>
<td>Confirmation of a <code>&lt;createcall&gt;</code> element or an asynchronous event</td>
</tr>
<tr>
<td>connection.redirected</td>
<td>Confirmation of a <code>&lt;redirect&gt;</code> element</td>
</tr>
<tr>
<td>connection.reject.failed</td>
<td>Failure indication for a <code>&lt;reject&gt;</code> element</td>
</tr>
<tr>
<td>connection.signal</td>
<td>Asynchronous event</td>
</tr>
<tr>
<td>error.connection</td>
<td>Asynchronous event</td>
</tr>
<tr>
<td>error.connection.wrongstate</td>
<td>Any call control tag</td>
</tr>
</tbody>
</table>

### Conference events

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conference.created</td>
<td>Confirmation of a <code>&lt;createconference&gt;</code> element</td>
</tr>
<tr>
<td>conference.destroyed</td>
<td>Confirmation of a <code>&lt;destroyconference&gt;</code> element</td>
</tr>
<tr>
<td>conference.joined</td>
<td>Confirmation of a <code>&lt;join&gt;</code> element</td>
</tr>
<tr>
<td>conference.unjoined</td>
<td>Confirmation of an <code>&lt;unjoin&gt;</code> element</td>
</tr>
<tr>
<td>error.conference</td>
<td>Asynchronous event</td>
</tr>
<tr>
<td>error.conference.create</td>
<td>Failure indication for a <code>&lt;createconference&gt;</code> element</td>
</tr>
<tr>
<td>error.conference.destroy</td>
<td>Failure indication for a <code>&lt;destroyconference&gt;</code> element</td>
</tr>
<tr>
<td>error.conference.join</td>
<td>Failure indication for a <code>&lt;join&gt;</code> element</td>
</tr>
<tr>
<td>error.conference.unjoin</td>
<td>Failure indication for a <code>&lt;unjoin&gt;</code> element</td>
</tr>
</tbody>
</table>
### Dialog events

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialog.disconnect</td>
<td>Asynchronous event from a dialog</td>
</tr>
<tr>
<td>dialog.exit</td>
<td>Asynchronous event from a dialog</td>
</tr>
<tr>
<td>dialog.prepared</td>
<td>Confirmation of a <code>&lt;dialogprepare&gt;</code> element</td>
</tr>
<tr>
<td>dialog.started</td>
<td>Confirmation of a <code>&lt;dialogstart&gt;</code> element</td>
</tr>
<tr>
<td>dialog.terminatetransfer</td>
<td>Asynchronous event from a dialog</td>
</tr>
<tr>
<td>dialog.transfer</td>
<td>Asynchronous event from a dialog</td>
</tr>
<tr>
<td>dialog.user.*</td>
<td>Asynchronous event from a dialog</td>
</tr>
<tr>
<td>error.dialog</td>
<td>Asynchronous event from a dialog</td>
</tr>
<tr>
<td>error.dialog.notprepared</td>
<td>Failure indication for a <code>&lt;dialogprepare&gt;</code> element</td>
</tr>
<tr>
<td>error.dialog.notstarted</td>
<td>Failure indication for a <code>&lt;dialogstart&gt;</code> element</td>
</tr>
</tbody>
</table>

### Document Control Flow and Execution

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccxml.created</td>
<td>Confirmation of a <code>&lt;createccxml&gt;</code> element</td>
</tr>
<tr>
<td>ccxml.exit</td>
<td>Confirmation of an <code>&lt;exit&gt;</code> element</td>
</tr>
<tr>
<td>ccxml.kill</td>
<td>Asynchronous event from CCXML app or platform</td>
</tr>
<tr>
<td>ccxml.loaded</td>
<td>Asynchronous event</td>
</tr>
<tr>
<td>error.createccxml</td>
<td>Failure indication for a <code>&lt;createccxml&gt;</code> element</td>
</tr>
<tr>
<td>error.fetch</td>
<td>Failure indication for a <code>&lt;fetch&gt;</code> or <code>&lt;createccxml&gt;</code> element</td>
</tr>
</tbody>
</table>
Event Handling

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetch.done</td>
<td>Confirmation of an &lt;fetch&gt; element</td>
</tr>
<tr>
<td>cancel.successful</td>
<td>Confirmation of a &lt;cancel&gt; element</td>
</tr>
<tr>
<td>error.move</td>
<td>Failure indication for a &lt;move&gt; element</td>
</tr>
<tr>
<td>error.send.failed</td>
<td>Failure indication for a &lt;send&gt; element</td>
</tr>
<tr>
<td>error.send.targettypeinvalid</td>
<td>Failure indication for a &lt;send&gt; element</td>
</tr>
<tr>
<td>error.send.targetunavailable</td>
<td>Failure indication for a &lt;send&gt; element</td>
</tr>
<tr>
<td>move.successful</td>
<td>Confirmation of a &lt;move&gt; element</td>
</tr>
<tr>
<td>send.successful</td>
<td>Confirmation of an &lt;send&gt; element</td>
</tr>
<tr>
<td>error.notallowed</td>
<td>Failure indication for a &lt;cancel&gt; element</td>
</tr>
</tbody>
</table>

SIP Mappings for CCXML Call Control Failures

The following table illustrates the correlation to SIP errors when a <createcall>, <redirect>, or <merge> request fails. The reason attribute for the connection.failed, error.merge and error.redirect events is set according to the mappings outlined below. This information is made available to a CCXML application via "event$.reason".

<table>
<thead>
<tr>
<th>SIP Code</th>
<th>CCXML (Merge/Redirect/Createcall) Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>404: Not Found</td>
<td>Not found</td>
</tr>
<tr>
<td>405: Method Not Allowed</td>
<td>Feature not supported</td>
</tr>
<tr>
<td>480: Temporarily Unavailable</td>
<td>Service temporarily unavailable</td>
</tr>
<tr>
<td>486: Busy Here 600: Busy Everywhere</td>
<td>Remote end is busy</td>
</tr>
</tbody>
</table>
### SIP Code

<table>
<thead>
<tr>
<th>SIP Code</th>
<th>CCXML (Merge/Redirect/Createcall)</th>
</tr>
</thead>
<tbody>
<tr>
<td>408: Request Timeout 487: Request Cancelled</td>
<td>Remote end did not answer</td>
</tr>
<tr>
<td>503: No Service</td>
<td>No service is available</td>
</tr>
<tr>
<td>504: Server Timeout</td>
<td>Network is busy</td>
</tr>
<tr>
<td>603: Declined</td>
<td>Request rejected</td>
</tr>
<tr>
<td>Other 3xx/4xx/5xx/6xx</td>
<td>Internal system failure</td>
</tr>
</tbody>
</table>

### Generic Error Event

<table>
<thead>
<tr>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>error.semantic</td>
</tr>
<tr>
<td>error.unsupported</td>
</tr>
</tbody>
</table>

### VoiceXML events

The VoiceXML interpreter detects predefined events based on errors, telephone disconnects, or user requests. When an event is thrown, the associated event handler, if it exists, is invoked. Then execution resumes in the element that executed when the event was thrown.

An application can define additional events and use a `<throw>` element to return an event of a specified kind.

An application can catch and respond to an event in an event handler. A `<catch>` element is a general-purpose event handler which uses the event attribute to specify the type of event it handles.

When an event is thrown, the associated event handler, if it exists, starts. If the handler did not cause the application to terminate, execution resumes in the element that is executed when the event was thrown.

### Predefined events

The following standard events are predefined:
<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exit</td>
<td>The user asked to exit.</td>
</tr>
<tr>
<td>help</td>
<td>The user asked for help.</td>
</tr>
<tr>
<td>noinput</td>
<td>The user did not provide timely input.</td>
</tr>
<tr>
<td>nomatch</td>
<td>The user did not provide input understood by the application grammar.</td>
</tr>
<tr>
<td>cancel</td>
<td>The user asked to cancel the current prompt.</td>
</tr>
<tr>
<td>connection.disconnect.hangup</td>
<td>The user hung up.</td>
</tr>
<tr>
<td>connection.disconnect.transfer</td>
<td>The user’s call was transferred.</td>
</tr>
<tr>
<td>error.badfetch</td>
<td>An error occurred while the interpreter was fetching the document or resources.</td>
</tr>
<tr>
<td>error.noauthorization</td>
<td>The user is not authorized to perform the requested action.</td>
</tr>
<tr>
<td>error.semantic</td>
<td>A run-time error occurred in the VoiceXML code.</td>
</tr>
<tr>
<td>error.connection.baddestination</td>
<td>The destination URI for an outbound telephone call was invalid.</td>
</tr>
<tr>
<td>error.connection.noauthorization</td>
<td>An unauthorized outbound telephone call was attempted.</td>
</tr>
<tr>
<td>error.connection.noresource</td>
<td>A telephone resource is unavailable.</td>
</tr>
<tr>
<td>error.noresource</td>
<td>An audio input or output resource is unavailable.</td>
</tr>
<tr>
<td>error.unsupported.format</td>
<td>The requested resource format is not supported.</td>
</tr>
<tr>
<td>error.unsupported.element</td>
<td>The requested element is not supported.</td>
</tr>
<tr>
<td>error.badfetch.http.response_code</td>
<td>An error occurred while the interpreter was fetching the document or resources. The interpreter must use a detailed event type to determine which response code was encountered.</td>
</tr>
<tr>
<td>error.unsupported.language</td>
<td>The language is not supported.</td>
</tr>
</tbody>
</table>
**Application defined event handlers**

A VoiceXML application can provide custom event handlers that override the default handlers. An element in which an event can be thrown inherits event handlers defined in its ancestor elements.

An application defines additional events if a tag throws an event which specifies a non predefined event. An application can use a `<catch>` element to catch and handle an application-defined event.

- Variable references are resolved relative to the scope of the element where the event was thrown.
- URL references are resolved relative to the document from which the event was thrown.

Form items contain event counters that throw responses if the same event is thrown multiple times. These event counters are reset each time the form is invoked. When an event occurs, you can use the counter to select applicable event handlers:

- All handlers in the scope in which the event occurred and its containing scopes are considered.
- A handler for the event is eligible if its count attribute is less than or equal to the event counter.
- Those eligible handlers with the highest count are selected as applicable (more than one handler can have the same highest count).
- The applicable handlers are ordered by scope, with the innermost handlers first.

Within a scope, handlers are examined in the order in which they occur in the VoiceXML document. The first applicable handler in is selected to handle the event. Event handlers are configured to catch all events with a specified prefix. However, the interpreter selects a handler based on count, scope, and document order only. A specific handler does not take precedence.

Within an event handler:

- The `_event` variable contains the name of the event currently handled.
- The `_message` variable contains the message string that provides additional information about the event. If no message was supplied when the event was thrown, the `_message` variable is undefined.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error.unsupported.objectname</td>
<td>The object is not supported.</td>
</tr>
</tbody>
</table>
Event handling
Chapter 9: Deployment

This section covers what you need to configure when deploying your applications.

Navigation

Deploy applications on page 109

Deploy applications

All applications need to be configured as part of their deployment. For more information on deploying applications, see Avaya Media Server Administration - Application Management, NN44471-601.

The following configurations are required when deploying an application:

- Licenses - The Avaya MS requires a license to run an application
- SIP trusted nodes - SIP Trusted nodes are pasted into the SIP Nodes and Routes page of Element Manager
- MRCP pools
- Custom applications
- Common application configurations
- Advanced application configurations
Chapter 10: Reporting

The following reporting tools are used in applications and interpreters.

Navigation

- **Counters** on page 111
- **Application statistics** on page 113
- **Application defined system counters** on page 119

---

Counters

Counters are used for recording events on the system.

---

### Global CCXML Counters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FetchTotalRequests</td>
<td>Total <code>&lt;fetch&gt;</code> requests issued.</td>
</tr>
<tr>
<td>FetchParseFailures</td>
<td>Count number of times a <code>&lt;fetch&gt;</code> failed due to a document parse error</td>
</tr>
<tr>
<td>FetchTimeouts</td>
<td>Count number of times a <code>&lt;fetch&gt;</code> failed due to a timeout (408)</td>
</tr>
<tr>
<td>FetchNotFound</td>
<td>Count number of times a <code>&lt;fetch&gt;</code> failed file not found (404)</td>
</tr>
<tr>
<td>CreateccxmlTotalRequests</td>
<td>Total <code>&lt;createccxml&gt;</code> requests issued.</td>
</tr>
<tr>
<td>CreateccxmlParseFailures</td>
<td>Count number of times a <code>&lt;createccxml&gt;</code> failed due to a document parse error</td>
</tr>
<tr>
<td>CreateccxmlTimeouts</td>
<td>Count number of times a <code>&lt;createccxml&gt;</code> failed due to a timeout (408)</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CreateccxmlNotFound</td>
<td>Count number of times a <code>&lt;createccxml&gt;</code> failed file not found (404)</td>
</tr>
<tr>
<td>TotalDialogStarts</td>
<td>Total <code>&lt;dialogstart&gt;</code> requests received</td>
</tr>
<tr>
<td>DialogStartFailures</td>
<td>Count <code>&lt;dialogstart&gt;</code> failures</td>
</tr>
<tr>
<td>TotalConferencesCreated</td>
<td>Total <code>&lt;createconference&gt;</code> requests received</td>
</tr>
<tr>
<td>FailedConferencecreates</td>
<td>Count <code>&lt;createconference&gt;</code> failures</td>
</tr>
<tr>
<td>TotalJoinRequests</td>
<td>Total <code>&lt;join&gt;</code> requests received</td>
</tr>
<tr>
<td>FailedJoins</td>
<td>Count <code>&lt;join&gt;</code> failures</td>
</tr>
<tr>
<td>TotalInboundCalls</td>
<td>Number of inbound calls received (connection.alerting)</td>
</tr>
<tr>
<td>InboundCallFailures</td>
<td>Number of Inbound calls that failed to transition to a connected state</td>
</tr>
<tr>
<td></td>
<td>(connection.connected) after issuing an <code>&lt;accept&gt;</code></td>
</tr>
<tr>
<td>CallsRejected</td>
<td>Number of calls that were rejected in an alerting state</td>
</tr>
<tr>
<td></td>
<td>(connection.alerting)</td>
</tr>
<tr>
<td>TotalOutboundCalls</td>
<td>Number of outbound calls attempts <code>&lt;createcall&gt;</code></td>
</tr>
<tr>
<td>OutboundCallFailures</td>
<td>Number of Outbound calls that failed to transition to a connected state</td>
</tr>
<tr>
<td></td>
<td>(connection.connected)</td>
</tr>
</tbody>
</table>

### Session CCXML Counters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SessionFetchTotalRequests</td>
<td>Total <code>&lt;fetch&gt;</code> requests issued.</td>
</tr>
<tr>
<td>SessionFetchFailures</td>
<td>Count number of times a <code>&lt;fetch&gt;</code> failed due to a document parse error</td>
</tr>
<tr>
<td>SessionCreateccxmlTotalRequests</td>
<td>Total <code>&lt;createccxml&gt;</code> requests issued.</td>
</tr>
</tbody>
</table>
Parameter Description
---
SessonCreateccxmlFailures Count number of times a `<createccxml>` failed due to a document parse error
SessonTotalDialogStarts Total `<dialogstart>` requests received
SessonDialogStartFailures Count `<dialogstart>` failures
SessonTotalConferencesCreated Total `<createconference>` requests received
SessonFailedConferencecreates Count `<createconference>` failures
SessonTotalJoinRequests Total `<join>` requests received
SessonFailedJoins Count `<join>` failures

---

**Application statistics**

VoiceXML interpreter counters (VXML 2.1 extension) are used for various types of events/conditions that occur during the life of the component (process). The counters used in this implementation are listed under the following categories: **General counters** on page 113, **Transfer counters** on page 114, **Fetch/Parse counters** on page 115, **Input counters** on page 117, and **Output counters** on page 118.

---

**General counters**

<table>
<thead>
<tr>
<th>Counter Name</th>
<th>Description</th>
<th>System</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>NbrCalls</td>
<td>Total number of calls received by the VoiceXML interpreter.</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>NbrEvtNoResource</td>
<td>Total number of error.noresource events thrown.</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>NbrEvtErrNoAuth</td>
<td>Total number of error.noauthorization events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtErrorObject</td>
<td>Total number of error.object events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
### Transfer counters

<table>
<thead>
<tr>
<th>Counter Name</th>
<th>Description</th>
<th>System</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>NbrTransfers</td>
<td>Total number of transfer attempts.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrBlindTransfers</td>
<td>Total number of blind transfer attempts.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrBridgedTransfers</td>
<td>Total number of bridge transfer attempts.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrConsultTransfers</td>
<td>Total number of consultation transfer attempts.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrTransferFails</td>
<td>Total number of transfer failures.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrXferFailCalleeBusy</td>
<td>Total number of transfer failures due to the callee being busy.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrXferFailNetBusy</td>
<td>Total number of transfer failures due to the network being busy.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrXferFailNoAnswer</td>
<td>Total number of transfer failures due to the callee not answering within the time specified by the connecttimeout attribute.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtConnNoAuth</td>
<td>Total number of error.connection.noauthorization events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Counter Name</td>
<td>Description</td>
<td>System</td>
<td>Session</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>NbrEvtConnBadDest</td>
<td>Total number of error.connection.baddestination events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtConnNoRoute</td>
<td>Total number of error.connection.noroute events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtConnNoRsrc</td>
<td>Total number of error.connection.noresource events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtUnsupportBlind</td>
<td>Total number of error.unsupported.transfer.blind events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtUnsupportBridge</td>
<td>Total number of error.unsupported.transfer.bridge events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtUnsupportURI</td>
<td>Total number of error.unsupported.uri events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

**Fetch/Parse counters**

<table>
<thead>
<tr>
<th>Counter Name</th>
<th>Description</th>
<th>System</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>NbrFetchAttempts</td>
<td>Total number of attempts to fetch a VoiceXML document</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtBadFetches</td>
<td>Total number of error.badfetch events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtBadFetchURI</td>
<td>Total number of error.badfetch.bad events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtBadFetchAppURI</td>
<td>Total number of error.badfetch.application.uri events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtBadFetchDialog</td>
<td>Total number of error.badfetch.baddialog events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Counter Name</td>
<td>Description</td>
<td>System</td>
<td>Session</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>NbrEvtBadFetchHttpErr</td>
<td>Total number of error.badfetch.http.# events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrDiskCacheHits</td>
<td>Total number of disk cache hits.</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>NbrDiskCacheMiss</td>
<td>Total number of disk cache misses.</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>NbrMemoryCacheHits</td>
<td>Total number of memory cache hits.</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>NbrMemoryCacheMiss</td>
<td>Total number of memory cache misses.</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>NbrEvtSemanticErrors</td>
<td>Total number of error.semantic.events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtSemErrECMA</td>
<td>Total number of error.semantic.ecmascript events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtSemErrBadThrow</td>
<td>Total number of error.semantic.no_event_in_throw events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtSemErrNoGram</td>
<td>Total number of error.semantic.nogrammars events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtUnsuppFormat</td>
<td>Total number of error.unsupported.format events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtUnsuppObj</td>
<td>Total number of error.unsupported.object name events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtUnsuppLang</td>
<td>Total number of error.unsupported.language events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtUnsuppBuiltin</td>
<td>Total number of error.unsupported.builtin events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrPosts</td>
<td>Total requests with method set to post.</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
### Input counters

<table>
<thead>
<tr>
<th>Counter Name</th>
<th>Description</th>
<th>System</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nbrgets</td>
<td>Total requests with method set to get or undefined, default is a get method.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrSpeechInputs</td>
<td>Total number of speech inputs.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrDTMFInputs</td>
<td>Total number of DTMF inputs.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrFailedInputs</td>
<td>Total number of failed inputs.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtNoInputs</td>
<td>Total number of noinput events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtNoMatches</td>
<td>Total number of nomatch events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtMaxSpeechTimeouts</td>
<td>Total number of maxspeechtimeout events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrUserCancelReq</td>
<td>The number of pre-defined cancel events thrown. This occurs when the user speaks ‘cancel’ and the universal cancel grammar is active.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrUserExitReq</td>
<td>The number of pre-defined exit events thrown. This occurs when the user speaks ‘exit’ and the universal exit grammar is active.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrUsrHelpReq</td>
<td>The number of pre-defined help events thrown. This occurs when the user speaks ‘help’ and the universal help grammar is active.</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
### Output counters

<table>
<thead>
<tr>
<th>Counter Name</th>
<th>Description</th>
<th>System</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>NbrSpeechInputs</td>
<td>Total number of speech inputs.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrDTMFInputs</td>
<td>Total number of DTMF inputs.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrFailedInputs</td>
<td>Total number of failed inputs.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtNoInputs</td>
<td>Total number of noinput events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtNoMatches</td>
<td>Total number of nomatch events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtMaxSpeechTimeouts</td>
<td>Total number of maxspeechtimeout events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrUserCancelReq</td>
<td>The number of pre-defined cancel events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Counter Name</td>
<td>Description</td>
<td>System</td>
<td>Session</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>NbrUserExitReq</td>
<td>The number of pre-defined exit events thrown. This occurs when the user speaks 'exit' and the universal exit grammar is active.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrUsrHelpReq</td>
<td>The number of pre-defined help events thrown. This occurs when the user speaks 'help' and the universal help grammar is active.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtBadGrammar</td>
<td>Total number of error.badgrammar events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtBadInlineGrammar</td>
<td>Total number of error.grammar.inlined events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtBadChoiceGrammar</td>
<td>Total number of error.grammar.choice events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtErrorRecog</td>
<td>Total number of error.recognition events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrEvtErrorRecord</td>
<td>Total number of error.record events thrown.</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>NbrRecordings</td>
<td>Total number of recordings requested.</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

**Application defined system counters**

VoiceXML applications can keep track of client-side counters and stats. This is achieved via an Avaya specific extension of the `<object>` tag (classid of "com.avaya.ivr.statincr").

The following VoiceXML code demonstrates how existing applications maintain application level counters and stats:

```xml
Application defined system counters

VoiceXML and CCXML Application Programming

3 Dec 2010 119"
The above code increments the statistical counter “balance” within the static group “TransactionType”. This interface is being deprecated, however it continues to be supported for backwards compatibility.

A new interface is introduced with this feature to allow VoiceXML applications to keep track of client-side counters and stats. This new interface takes advantage of features available with SpiderMonkey 1.6 (open source JavaScript interpreter) to allow VoiceXML applications to extend a predefined JavaScript object with their own counters and utilize standard JavaScript assignment and increment/decrement operators on these custom counters. These custom counters, when updated, result in the VoiceXML interpreter creating/updating the counters in the database resident on the Avaya MS platform, which generates the reports. The following sample JavaScript illustrates how a VoiceXML application updates the same counter from the above example which used <object>:

```javascript
OM.TransactionType.balance++;<script/>
```

The following represents the method to increment the same counter in VXML:

```xml
<assign name="OM.TransactionType.balance" expr="OM.TransactionType.balance++"/>
```

**Important:**

In the above examples the object OM is a special object that is exposed by the VoiceXML interpreter.
Chapter 11: Code validation

Several tools are available to assist you in validating the code in your applications.

Navigation

- Syntax validation on page 121
- Utilities on page 121
- Grammar validation on page 122

Syntax validation

Syntax validation is handled differently for VoiceXML and CCXML.

- Use the validatedoc utility to validate VoiceXML syntax
- Compile CCXML to validate CCXML syntax

Utilities

The following utilities are available for validating your VoiceXML applications.

Use validatedoc to retrieve a specified document and verify that it is a valid VoiceXML document. If VALID is returned, your document will run inside the VoiceXML interpreter. If a failure occurs, check your document or your Web application that generated the document.

**Important:**
This utility does not check for logic errors or javascript errors.

Use httpreq to make an http request as specified by the input file. Your application can use this program to verify that you can reach your Web server.

The httpreq program is used as follows:

```
httpreq [-h hostname] [-p port] [-f inputfile] [-H] -h hostname (localhost) -p port (80) -f inputfile (httpreq.txt) -H Show this message
```

The input file has the following format:

```
GET / HTTP/1.1 Host: MyHostname UserAgent: httpreq Connection: close
```
Grammar validation

Refer to your vendor documentation for steps on validating grammar.
Chapter 12: Logging errors and troubleshooting code

This section covers how errors are logged and tools available for troubleshooting applications.

Navigation

- Logging on page 123
- Events and alarms on page 124

Logging

A variety of trace and log files are available for determining the nature of problems in an application.

Trace files

Trace files are created for each component of the Avaya MS platform.

The Trace File Size property, found in Element Manager (Configuration>Logging>Debug) defines the maximum size of the trace file (in bytes). Once the current trace file reaches the maximum size it is rolled out with a .bak extension and a new trace file is created. The Trace File History property specifies the number of rotated trace files that should be kept before discarding the oldest trace file.

The following key trace files are created by the platform:

<table>
<thead>
<tr>
<th>Trace File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>confmpDebug.txt</td>
<td>Conference Media Processor trace logging</td>
</tr>
<tr>
<td>cstoreDebug.txt</td>
<td>Content Store trace logging</td>
</tr>
<tr>
<td>ivrmpDebug.txt</td>
<td>IVR Media Processing trace logging</td>
</tr>
<tr>
<td>reporterDebug.txt</td>
<td>Reporter trace logging</td>
</tr>
<tr>
<td>scDebug</td>
<td>Session Controller trace logging</td>
</tr>
<tr>
<td>sipmcDebug.txt</td>
<td>SIP Multimedia Conductor trace logging</td>
</tr>
</tbody>
</table>
Logging errors and troubleshooting code

<table>
<thead>
<tr>
<th>Trace File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>soapserverDebug.txt</td>
<td>Soap Server trace logging</td>
</tr>
<tr>
<td>srp.log srp_state.log</td>
<td>Startup and Recovery Process trace logging</td>
</tr>
<tr>
<td>streamsourceDebug.txt</td>
<td>Stream Source trace logging</td>
</tr>
<tr>
<td>vxmlAppDebug.txt</td>
<td>VXML application trace logging generated by the &lt;log&gt; tag.</td>
</tr>
<tr>
<td>vxmlIBug.txt</td>
<td>VoiceXML Browser trace logging</td>
</tr>
<tr>
<td>ccxmlApp.txt</td>
<td>CCXML application trace logging generated by the &lt;log&gt; tag.</td>
</tr>
<tr>
<td>ccxmlIBug.txt</td>
<td>VoiceXML Browser trace logging</td>
</tr>
</tbody>
</table>

---

Events and alarms

Events and alarms are tools for informing you of errors in your applications.

CCXML alarms

No alarms are defined for the CCXML function.

VoiceXML events

The VoiceXML interpreter can generate two events.

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>12801</td>
<td>Supports legacy applications that are written with the object tag classid:&quot;com.avaya.ivr.alarm&quot; Legacy application (user defined) alarm</td>
</tr>
<tr>
<td>12800</td>
<td>SSL Connection: self signed certificate in certificate chain SSL Connection: mismatched Common Name or mismatched critical subjectAltName</td>
</tr>
</tbody>
</table>
VoiceXML alarms

The VoiceXML Interpreter currently has no alarms.
Logging errors and troubleshooting code