CHAPTER 1

Cisco VoiceXML Features

Revised: October 2, 2009, OL-11175-01

This chapter describes Cisco VoiceXML features, and their implementation based on the *VoiceXML 2.1* W3C Candidate Recommendation (June 13, 2005) and contains the following sections:

Audience

This guide is intended primarily for developers writing VoiceXML applications using Cisco VoiceXML features. It describes Cisco VoiceXML features based on the *VoiceXML 2.1* W3C Candidate Recommendation (June 13, 2005) and must be used in conjunction with:

- *Cisco IOS Tcl and VoiceXML Application Guide* for your Cisco IOS release
- *Tcl IVR Version 2.0 Programmer's Guide*
- *VoiceXML 2.1* W3C Candidate Recommendation (June 13, 2005)

Note Cisco IOS Release 12.4(11)T is based on the W3C VoiceXML Version 2.0 Specification, (W3C Recommendation 16 March 2004). Cisco IOS Release 12.4(15)T adds support for Media Resource Control Protocol version 2 (MRCP v2) servers and *VoiceXML 2.1* W3C Candidate Recommendation (June 13, 2005).

Recommended Knowledge

We recommend you have the following knowledge before using this guide:

- For setting up the VoiceXML application environment:
  - Knowledge of VoiceXML
  - Experience with web application administration
  - Knowledge of languages and protocols such as HTML and HTTP
- For working with VoiceXML applications and writing VoiceXML documents:
  - Knowledge of web page development
  - Familiarity with the *VoiceXML 2.1* W3C Candidate Recommendation (June 13, 2005)
  - Knowledge of VoiceXML
- For configuring the Cisco VoiceXML-enabled gateway:
  - Experience with the prerequisite configuration of the Cisco voice gateway
  - Familiarity with Cisco IVR and VoIP functionality

Prerequisites

This section describes the prerequisites necessary to develop a VoiceXML application using Cisco VoiceXML features:

- VoiceXML Document Development, page 1-2
- Cisco Voice Gateway Requirements, page 1-3

VoiceXML Document Development

You must write a VoiceXML document using a web-authoring tool, defining your voice application. The document must be installed on a web or file server. A VoiceXML document can also call for the gateway to interact with various web applications (servlets and cgi executables) in which case you must also supply these web applications.

In general, HTTP is the preferred protocol for loading VoiceXML applications and audio prompts. The HTTP client code is implemented in Cisco IOS specifically for this purpose. The Cisco IOS File System (Cisco IFS) protocols (FTP, TFTP) were implemented for loading images, and saving and restoring configurations, so there are limits to the efficiency and number of concurrent loads. HTTP has been developed for efficiency over the Web; it has mechanisms to determine how long a file is considered valid in cache, and to determine if a cached version is still valid. With TFTP, the only way to determine if a cached version is valid is by reloading the entire file.

Pages which are loaded through a pointer within a document using TFTP are not cached on the gateway. TFTP should not be used for loading these dynamic documents. For example, the application attribute of the <vxml> tag and the next attribute in the <goto> tag should not use IFS protocols in the URI. These documents should use HTTP.

Note Place all VoiceXML documents behind a firewall.

For more information, see the following resources:

- *VoiceXML 2.1* W3C Candidate Recommendation (June 13, 2005)
- World Wide Web Consortium's Voice Browser Activities
- Media Resource Control Protocol version 2 (MRCP v2) (draft-ietf-speechsc-mrcpv2-10) (June 9, 2006)

Note Cisco VoiceXML features in Cisco IOS Release 12.2(11)T through 12.4(11)T releases are based on the *VoiceXML Version 2.0* W3C Recommendation (March 16, 2004).

Cisco Voice Gateway Requirements

For information on hardware and software requirements for the voice gateway, see the *Cisco IOS Tcl IVR and VoiceXML Application Guide*.

For information on configuring your external media server, see your vendor's documentation:

- Nuance Communications
- Loquendo S.p.A. ASR and TTS products

Note The Cisco IOS VoiceXML gateway using MRCP v2 was qualified with Nuance Recognizer 9.0.1, RealSpeak 4.5.0, and Nuance Speech Server 5.0.1.

Overview of Cisco VoiceXML Features

This section provides an overview of the Cisco VoiceXML features implemented in Cisco IOS Release 12.(2)11T.

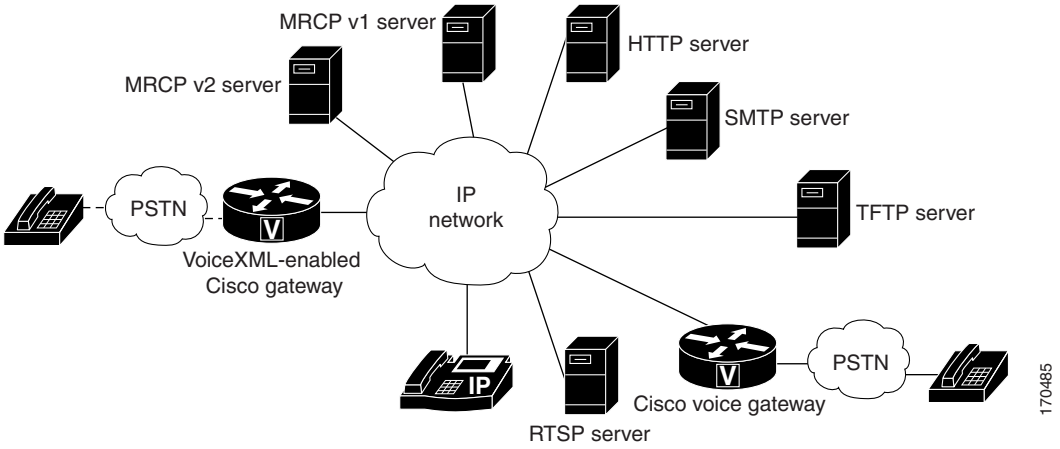
Note In Cisco IOS Release 12.4(15)T and later releases, VoiceXML 1.0 is not supported.

Applications written in Voice eXtensible Markup Language (VoiceXML) provide access through a voice browser to content and services over the telephone, just as Hypertext Markup Language (HTML) provides access through a web browser running on a PC. The universal accessibility of the telephone and its ease of use makes VoiceXML applications a powerful alternative to HTML for accessing the information and services of the World Wide Web.

Cisco VoiceXML provides a platform for interpreting VoiceXML documents. When a telephone call is made to the Cisco VoiceXML-enabled gateway, VoiceXML documents are downloaded from web servers, providing content and services to the caller, typically in the form of prerecorded audio in an IVR application. Customers can access online business applications over the telephone, providing for example, stock quotes, sports scores, or bank balances.

VoiceXML brings the advantages of web-based development and content delivery to voice applications. It is similar to HTML in its simplicity and in its presentation of information. Cisco VoiceXML is based on the *VoiceXML 2.1* W3C Candidate Recommendation (June 13, 2005) and is designed to provide web developers great flexibility and ease in implementing VoiceXML applications. Figure 1-1 shows components that can be configured as part of a VoiceXML application installed on a Cisco voice gateway.

Figure 1-1 Cisco VoiceXML Application Components



The following is an example of a call flow for a VoiceXML application:

Note In Cisco IOS Release 12.3, the recording feature is supported only on the Cisco AS5350XM and Cisco AS5400XM.

1. The caller dials a number and is connected through the PSTN or the IP network to a Cisco voice gateway that is configured as a VoiceXML-enabled gateway.

   For instance, the caller could be connected to a business providing sport scores over the telephone.

2. The Cisco voice gateway uses the caller's DNIS information and associates it with the appropriate VoiceXML document, residing on a web server (for example, an HTTP server).

   For example, this business uses a VoiceXML document on an HTTP server to provide baseball game scores.

3. The voice gateway runs the VoiceXML document and responds to the caller's input by playing the appropriate audio content.

An application might play a recorded prompt that asks the caller to press a specific dual tone multiple frequency (DTMF) key (for example, "Press 2 for the results of tonight's National League Playoff Game") to hear a spoken sport score ("Giants 4, Mets 0").

4. Cisco IOS VoiceXML could also transfer the caller to another party, perhaps customer service.

   For example, the application, after playing the score, might prompt the caller with the message: "If you sign up for a year's service now, you'll be entered in the drawing for two tickets to this year's World Series. Press 5 to contact one of our agents."

The following is an example of a call flow for VoiceXML application using recording:

1. The caller dials a number and is connected through the PSTN or the IP network to a Cisco voice gateway that is configured as a VoiceXML-enabled gateway.
2. The Cisco voice gateway uses the caller's DNIS information and associates it with the appropriate VoiceXML document, residing on a web server.
3. The voice gateway runs the VoiceXML document and responds to the caller's input by playing the appropriate prerecorded audio files.
4. The gateway executes the document, which prompts the user to record a voice message.
5. The message is recorded by the gateway and stored in local memory with the selected audio encoding.
6. After the recording is completed, the user can review the message or submit it by either pressing a specified key or hanging up the call.
7. When the user submits the voice mail message, the gateway's VoiceXML browser submits the voice message in .au or .wav file format to a specified URL using the HTTP POST method.
8. After receiving the message, the web server stores the message in the appropriate mailbox.
9. After successfully storing the voice message, the application instructs the media stream process to delete the local copy of the voice message.

Voice Store and Forward Feature

The Voice Store and Forward feature in Cisco IOS Release 12.2(11)T expands the capabilities of Cisco VoiceXML to include the input and processing of form field entries using recorded voice clips, instead of numeric input only. This feature uses the VoiceXML 2.0 <record> element to capture voice clips which can then be submitted to an external web server using Hypertext Transfer Protocol (HTTP) or Real Time Streaming Protocol (RTSP), or to a messaging server using Simple Mail Transfer Protocol (SMTP) for additional processing.

This recording feature can be used to collect caller names or addresses for call screening, product registration, or similar e-commerce applications, and for simple voice messaging, or any voice browser application where alphanumeric input using dual tone multifrequency (DTMF) is cumbersome or impractical.

The Voice Store and Forward feature supports speech recording and playout with a choice of four different media targets, including:

- Local memory—Intended for storing short-length speech clips, such as caller name or address, or a short voice message. Recordings can be submitted to HTTP server using POST method.
- RTSP—Intended for storing indefinite-length audio recordings. Voice recording is directly streamed to an external RTSP server using the recording URL specified by the user.
- HTTP—Recording is done by directly sending voice data to an HTTP server.

- SMTP—Recording is done by directly sending voice recordings to the SMTP server as e-mail audio attachments. This option supports the Mailto: URL.

The Voice Store and Forward feature enables voice messaging by dynamically switching a busy or no-answer voice call to a VoiceXML application. The voice gateway can operate in two modes:

- On-ramp mode—Incoming calls are handled by a VoiceXML document that lets callers record voice messages if the destination is busy or there is no answer. The on-ramp gateway saves all voice recordings as audio clip files to internal memory if space is available, or onto an external HTTP or RTSP server, or by directly streaming the voice message as an e-mail enclosure to an external ESMTP server. The audio clips are then converted into outbound ESMTP e-mail messages.
- Off-ramp mode—The external mail server sends an e-mail notification to the off-ramp gateway. The off-ramp gateway matches the DNIS in the e-mail header to a MMOIP dial peer, and places the call to the PSTN or IP phone user. When the call is answered, the gateway executes the VoiceXML application. The VoiceXML application is responsible for delivering the audio clip from the external media server to the outbound PSTN or IP destination using media play functions. The gateway does not support streaming the audio clip directly from the SMTP server.

Volume and Rate Control Feature

The volume of audio prompts can be adjusted during playback. Audio prompts that are played out from an HTTP server using the G.711 codec can also be speeded up or slowed down. A VoiceXML variable contains the rate and duration of the last prompt that was played.

The rate and volume of prompts is controlled by using specific attributes in the VoiceXML document. see the "Volume and Rate Control" section on page 1-16 for details.

Note This feature is a nonstandard extension to VoiceXML and is subject to nonbackward compatible changes in future versions.

ASR and TTS Features

Cisco IOS Release 12.4(15)T enables Cisco voice gateways to support automatic speech recognition (ASR) and text-to-speech (TTS) media services through Media Resource Control Protocol version 2 (MRCP v2) which uses Session Initiation Protocol (SIP) and Session Description Protocol (SDP) as session management protocols to create a session and set up media channels to the MRCP v2 server.

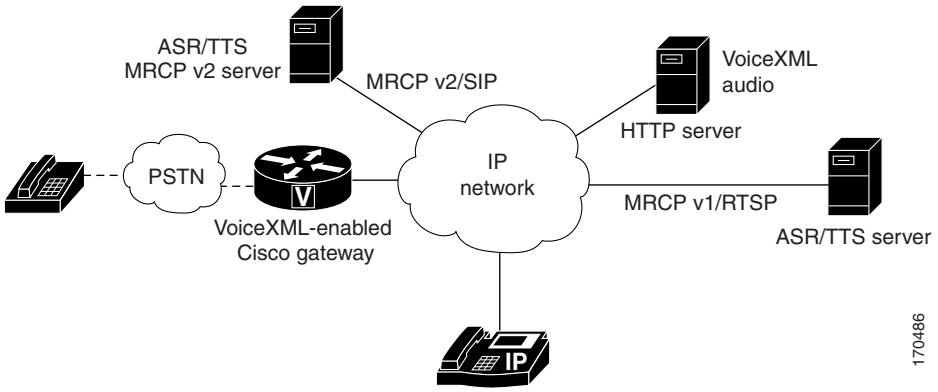
Cisco IOS Release 12.2(11)T introduces automatic speech recognition (ASR) and text-to-speech (TTS) capabilities for VoiceXML and Tcl IVR applications on Cisco voice gateways. This release also extends the Cisco VoiceXML interpreter to include support of some VoiceXML 2.0 features.

This release provides interfaces to ASR and TTS media servers using the Media Resource Control Protocol (MRCP), an application level protocol developed by Cisco and its ASR and TTS media server partner, Nuance Communications. MRCP is used to control media service resources such as speech synthesizers for TTS and speech recognizers for ASR. It provides a mechanism for client devices processing audio or video streams to control media resources or devices on external media servers. As shown in Figure 1-2, the Cisco gateway running an IVR application and the media servers providing TTS and ASR functionality maintain a client-server relationship through an RTSP connection; the RTSP client is the gateway and the RTSP server is the streaming media server providing ASR and TTS.

Note Cisco IOS Release 15.1(3)T introduces MRCPv1s command line interface to enhance the supported codec.When communicating with ASR and TTS server, the MRCPv1s command line interface enables the G.711(A-law) codec.

Note The Cisco IOS VoiceXML gateway using MRCP v2 was qualified with Nuance Recognizer 9.0.1, RealSpeak 4.5.0, and Nuance Speech Server 5.0.1.

Figure 1-2 A Network Implementing ASR and TTS



Tcl IVR 2.0 and VoiceXML Integration (Hybrid Applications) Feature

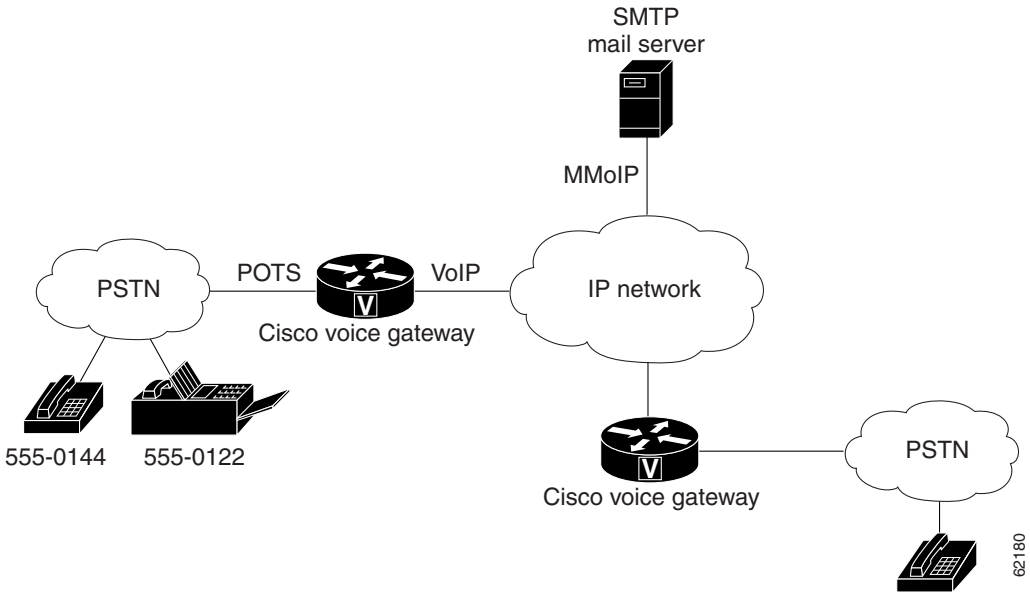
Tcl IVR 2.0 extensions allow Tcl applications to leverage support for ASR and TTS by invoking and managing VoiceXML dialogs within Tcl IVR scripts. This enables the implementation of hybrid applications using Tcl IVR for call control and VoiceXML for dialog management.

T.37 Store and Forward Fax Detection Feature

When a VoiceXML fax detection application is configured on the gateway, callers can dial a single number for both voice and fax calls. The gateway automatically detects that a call is a fax transmission by listening for CNG, the distinctive fax T.30 “calling” tone. When configured for fax detection, the

Cisco VoiceXML gateway continuously listens to incoming calls to determine which calls are voice or fax. The gateway then routes the calls to the appropriate application or media server, as shown in Figure 1-3.

Figure 1-3 Fax Detection on Cisco Gateway



Note In Cisco IOS Release 12.3, fax detection is supported only on the Cisco AS5350XM and Cisco AS5400XM.

After a call is established, the VoiceXML application can play an audio prompt to the caller while waiting for CNG detection. CNG detection continues for the entire duration of the call, so it is possible that a caller could first be connected to a voice-mail server and leave a voice message, then start to transmit a fax and the application would automatically switch the call to the fax application. After the application detects whether a call is voice or fax, the gateway routes the call based on dial peers. The fax detection application requires at least two dial peers:

- Inbound POTS dial peer, for incoming calls from the PSTN
- Outbound MMoIP dial peer for store-and-forward fax, to send fax transmissions to an e-mail server

For information on configuring fax relay or store-and-forward fax, see the *Cisco IOS Fax and Modem Services over IP Application Guide*.

System Output

This section includes:

- Audio Playout, page 1-9

  Describes the playout methods for audio prompts and the types of audio file formats and codecs supported for a Cisco VoiceXML application.

- Volume and Rate Control, page 1-16

  Describes how to control the volume and rate of audio playout.

- Type-ahead Support, page 1-20

  Describes how a type-ahead buffer works in a VoiceXML application.

Audio Playout

Cisco VoiceXML supports the following types of prompts:

- Prerecorded Audio Playout
- Synthesized Audio Playout

Prerecorded Audio Prompts

Cisco VoiceXML supports HTTP, TFTP, RAM, RTSP, and flash memory as sources of prerecorded audio prompts for playout. The audio files can be played out in cached or buffered mode, or they can be streamed.

In cached mode, audio files are loaded into the gateway's memory and then streamed from the gateway to the specific call leg. The streaming occurs only after the entire audio file is loaded from an HTTP or TFTP server. The amount of memory required to store these audio files can be controlled by using the **ivr prompt memory** command. Caching can be performed only with HTTP and TFTP servers. RTSP servers can only conduct real-time streaming of audio files.

Streaming of audio files can be performed by both HTTP and RTSP servers; however there is a difference in the method of streaming. HTTP servers stream files to the gateway in chunks, using the transfer encoding method; RTSP servers stream files continuously to the gateway.

With an HTTP server, use the **ivr prompt streamed** command to enable the gateway to stream audio files during playout. HTTP prompts are streamed by default but they can be disabled by using the **no ivr prompt streamed http** command.

With an RTSP server, use the **rtsp client timeout connect** command to set the number of seconds allowed for the router to establish a TCP connection to an RTSP server. With Cisco IOS Release 15.0(1)M and later releases, use the cisco-maxtime attribute of the <prompt> element to control the playout time for RTSP live streaming. If cisco-maxtime is zero or has no value set, the RTSP stream is played indefinitely.

Cisco voice gateways support .au and .wav file formats, and 14 codecs for audio playout on the PSTN or IP side of a call leg. The VoiceXML document specifies the audio format at the time of recording. If the audio format is not specified, the default format of .au is used. The G.711 codec has an standard industry header that can be recognized by the voice gateway and third-party media players in the industry. However, the other codecs that are supported by Cisco voice gateways have specific Cisco headers that may have to be modified for playout on third-party media players.

For more information on supported codecs, and on how to modify the headers in a codec, see Appendix A, "Audio File Support" in the *Cisco IOS Tcl and VoiceXML Application Guide* for your Cisco IOS release.

For <audio> playout, the language CLI configuration is not used to locate the audio files. The VoiceXML document uses the full URI in the *src* attribute of the <audio> element. If *src* is a relative URL (just the filename), the *base* attribute of the <vxml> element is used to form a complete URI. If the *base* attribute is not present, the VoiceXML document uses the URI from where it loads as the *base*.

Note In Cisco IOS Release 12.4(11)T and later releases, if the <audio> element is under the <prompt> tag and if the <prompt> tag specifies a base URL, then the relative URL in the *src* attribute of the <audio> element will use the base attribute of the <prompt> tag to form a complete URI.

Multiple <audio> Playout

If you use multiple audio files within a single <prompt> tag, the document first downloads all the audio files before playing them to avoid playback silence between each audio download. However, if the document cannot load one or more of the audio files, none of the audio files are played. The caller will not hear any playback and the document throws an error event.

Tip You may not hear an audio playout if you load all the audio files into a single <prompt>. For best results, separate the audio files into multiple <prompt> tags. The number of audio files in a single <prompt> tag depends on the number of audio files and their individual sizes.

You may also want to preload all or most of the commonly used audio files onto the gateway, or save them in the gateway flash memory instead of the HTTP server.

If you use a single audio file within each< prompt> tag, the document downloads and plays each audio file one at a time. For example:

```
<prompt>
  <audio src= "http://1.2.16.1/audio/en_welcome.au"/>
</prompt>
<prompt>
  <audio src= "http://1.2.16.1/audio/noaudio.au"/>
</prompt>
```

If the document cannot load an audio file, the caller will hear the welcome prompt before the document throws an error event. Using multiple audio files within a single <prompt> tag works with RTSP because it is streamed; with HTTP, you must use a single audio file within each <prompt> tag.

Using multiple audio files within a single <prompt> tag is only intended for concatenated prompts where playback silence after each download is a concern. For example;

```
<prompt>
 <audio src="http://1.2.16.1/audio/you_have.au"/>
 <audio src="http://1.2.16.1/audio/3.au"/>
 <audio src="http://1.2.16.1/audio/emails.au"/>
</prompt>
```

In the above example, because the prompts are concatenated, they must all be embedded within a single <prompt> tag to ensure that the playback will be:

"You have 3 emails."

If you split these concatenated prompts by embedding each one within its own <prompt> tag, as shown below:

```
<prompt>
 <audio src="http://1.2.16.1/audio/you_have.au"/>
</prompt>
<prompt>
 <audio src="http://1.2.16.1/audio/3.au"/>
</prompt>
<prompt>
 <audio src="http://1.2.16.1/audio/emails.au"/>
```

```
</prompt>
```

The playback will be:

"You have <silence...> 3 <silence...> emails", which is inappropriate.

Tip Use multiple sources in a single <prompt> tag only for concatenated prompts. For different codec types, embed each audio file within a single <prompt> for successful playout.

Note In Cisco IOS Release 12.4(11)T and later releases, no error event will be thrown if the audio file cannot be loaded or played if you enter the **vxml version 2.0** command.

Dynamic Prompts Playout

Note In Cisco IOS Release 12.4(11)T and later releases, only the xml:lang attribute of the <prompt> element in a VoiceXML document can be set to select a specifc language.

Dynamic prompts consist of small audio files played out in sequence. The type of language and the TTS notation used in playout is defined by Tcl language modules or built-in language definitions. Cisco IOS software includes built-in modules for English, Chinese, and Spanish. To add support for a new language, you must configure a new Tcl language module on the voice gateway. For information on how to configure a language module, see *Enhanced Multi-Language Support for Cisco IOS Interactive Voice Response*.

The language and location of the audio files can specified in the VoiceXML document or they can be configured through the CLI by using the **call application voice language** and the **call application voice set-location** commands. If only one language is configured in the CLI, then that language is assumed for all documents that do not specify a language.

If the language and location is to be specified in the VoiceXML document, use the xml:lang attribute of the <prompt> element to select a specific language, and to point to the location of the recorded files.

Synthesized Audio Playout

Synthesized audio playout consists of dynamic prompt and text-to-speech (TTS) prompt playout. Dynamic prompts allow basic TTS operations such as playing dollar amounts, time, and dates. An external media server provided by a third-party is required to play TTS prompts. The type of language supported for TTS playout is dependent on the external media server. In the event of an external media server failure, the application developer can implement a backup server through the script. For details, see the "External Server Failure" section on page 1-13.

Text-to-Speech Prompts

A third-party external media server is required to play TTS prompts. Playout of TTS prompts is dependent on the languages supported on the external media server. A speech synthesis markup language (SSML) is used to allow the development of synthetic speech in web applications. SSML allows VoiceXML document writers to control aspects of speech output, such as emphasis, pitch, and volume. The media server receives the speech synthesis markup specified in the VoiceXML document, converts

it to audio, and streams it to the user through the Cisco voice gateway. SSML support is dependent on the media server. For example, the media server may not be able to support some audio URIs such as TFTP, RTSP, and flash.

For information on SSML, see the *Speech Synthesis Markup Language Specification*.

Mixed Audio and Text-to-Speech Prompts

Processing of prompts by a Cisco voice gateway depends on the control of the prompt. Prerecorded audio prompts load (or stream) through the Cisco voice gateway without any interaction from the TTS media server. By default, prerecorded prompts load through the voice gateway; however a user can force the voice gateway to direct the SSML to the external media server by modifying the script.
For example, using “alternate text” in <audio src= “audio-to-play”> alternate text </audio> forces the gateway to send the SSML to the media server for playout.

For a <prompt> with SSML, the markup (which includes text and prerecorded audio) is sent to the TTS media server by configuring the CLI or using the com.cisco.tts-server property. The level of SSML support, and the types of file formats and codecs supported, are dependent on the support capabilities of the external media server.

Examples

The following example is handled by the TTS media server:

```
<prompt> this is a sample tts text <break/> with ssml </prompt>
```

The following example is handled by the TTS media server because it contains SSML:

```
<prompt> this is a sample tts text <break/> with ssml and an audio src
<="welcome.wav"/></prompt>
```

The following example is handled by the voice gateway because it contains only .au and src.

```
<prompt><audio src="one.au"/><audio src="two.au"/></prompt>
```

The following example shows how the voice gateway is forced to play SSML from the media server and the audio source from the gateway.

```
<prompt> this is a sample tts text <break/> with ssml and audio src </prompt>
<prompt><audio src="welcome.wav"/></Prompt>
```

The limitations of TTS in the Cisco implementation of VoiceXML 2.0 are:

- The media server is required to support HTTP clients only. This requirement imposes a limitation on the data contained in a <prompt> element.
- If the <prompt> element contains a W3C speech markup, then the entire markup in that <prompt> element uses an external media server for a TTS operation, and, a single G.711 u-law codec for the entire audio stream. This <prompt> element can only contain an HTTP type URI. The codec of the audio stream is limited to G.711 u-law because the media server cannot switch codecs.
- If the <prompt> element contains <audio src> elements instead of a W3C speech markup, these <audio src> elements can contain individual Flash, TFTP, FTP, and RTSP URIs, or a mix of any of them. These URIs are played using streaming mechanisms available on Cisco voice gateways. The audio files can have different codecs and the system plays them out in sequence subject to the following limitations:
  - The digital signal processor (DSP) supports the specified codecs on the telephony side.
  - On the IP side, interactive voice response supports the specified codecs. See the *Cisco IOS Tcl IVR and VoiceXML Application Guide* for your Cisco IOS release for details.

- A URI in the <audio src> element can point to a block of speech markup data, but such support is dependent on, and conditional to the vendor's decision of where the <audio src> element can point.

External Server Failure

In a Cisco VoiceXML solution, the Cisco voice gateway interacts with external media servers for automatic speech recognition, text-to-speech, and streamed recording. If an external media server fails to work, a backup server is desirable to take over the function of the failed media server. A typical approach is to use a content switch that is transparent to the VoiceXML application developer. However, this approach may not be very desirable because of the cost increase.

Cisco VoiceXML provides an alternate solution to external server failure where the application developer can handle server failure directly through the script. A backup external media server must be provided by the third party server vendor, and it must be initially configured on the voice gateway. In the event of a server failure, the VoiceXML interpreter throws an error event which is caught by the VoiceXML application, and the information is passed to a server side application.

The server side application is a web application written as a Java servlet or a Hypertext Preprocessor (PHP) script. For example, an error.noresource event is thrown if the external ASR server fails to operate. The VoiceXML script passes this error event information to a server side web application using the <submit> element. The server side application assigns a new backup media server using the <property> element in all subsequent VoiceXML pages that are generated dynamically.

The original VoiceXML document is regenerated and returned to the VoiceXML interpreter for execution. The regenerated VoiceXML document is similar to the original document with the exception of pointing to the new backup media server instead of the failed server.

Note In Cisco IOS Release 12.4(11)T and later releases, the error.noresource event is thrown if an external ASR or TTS server fails to operate. The old event com.cisco.media.resource.unavailable is supported for backward compatibility.

Example

This example shows how to recover from media server and HTTP server failures.

color.vxml

```
<?xml version="1.0"?>
<vxml version="2.0" xml:lang="en-US" application="root.vxml">

<!-- color.vxml
This is the main VoiceXML document invoked when the user calls into the gateway.
This VoiceXML document plays a text-to-speech prompt and waits for user input.
If the user's input matches the defined grammar, a prompt is played to indicate his
selection.
In the form, "colorSelection", the DOCUMENT_STATE and FORM_STATE are used to track the
execution state of this application. When an error event is caught in the root.vxml
document, these variables are submitted to the application server. This allows the
application server to generate a dynamic VoiceXML document according to the state of
execution of this application.-- >

    <property name="fetchtimeout" value="5s"/>
    <property name="timeout" value="5s"/>
    <property name="interdigittimeout" value="3s"/>
    <property name="termchar" value="#"/>
    <property name="bargein" value="true"/>
```

```
    <form id="colorSelection">
  <block>
    <assign name="DOCUMENT_STATE" expr="'color.vxml'"/>
        <assign name="FORM_STATE" expr="'colorSelection'"/>
  </block>

      <field name="color">
        <prompt bargein="true">
       Please select one of the following colors. Say red, blue or green
        </prompt>

         <grammar version="1.0" xml:lang="en-US" root="colors">
          <rule id="colors" scope="public">
            <one-of>
              <item>red</item>
              <item>blue</item>
              <item>green</item>
            </one-of>
         </rule>
        </grammar>

         <filled>
              <prompt>You have selected <value expr="color"/></prompt>
         </filled>
      </field>
  </form>
</vxml>
```

root.vxml

```
<?xml version="1.0"?>
<vxml version="2.0">
<!-- This is the root.vxml document which defines the catch handlers, the default
properties and application variables.
The error.noresource event handler is designed to handle two possible scenarios.
The first scenario is a VoiceXML application trying to play a text-to-speech (TTS) prompt
and collect user input for recognition. In this scenario, two error.noresource events are
thrown.
The first error event is thrown because of a TTS failure, the second event is thrown
because of an ASR failure. When the first event is caught, an error counter is
incremented. This counter is cleared when the second event is thrown.
The second scenario is a VoiceXML application trying to play a TTS prompt or trying to
conduct a recognition. In this case, only one error.noresource event is thrown.
The two forms, “dummyForm” and “resetMediaServer” handle the two scenarios.-- >

  <var name="WEB_SERVER" expr="'http://httpServer1/'"/>
  <var name="FORM_STATE" expr="'ROOT'"/>
  <var name="DOCUMENT_STATE" expr="'ROOT'"/>
  <var name="ERROR_EVENT" expr=""/>
  <var name="ERROR_COUNTER" expr="0"/>

  <catch event="error.noresource">
    <log> media resources unavailable</log>
    <assign name="ERROR_EVENT" expr="'MEDIA_RESOURCE_UNAVAILABLE'"/>
    <if cond = "ERROR_COUNTER == 0">
        <assign name="ERROR_COUNTER" expr="1"/>
        <goto next="#dummyForm"/>
    <else/>
      <assign name="ERROR_COUNTER" expr="0"/>
      <goto next="#resetMediaServer"/>
    </if>
  </catch>
```

```
<!-- This form is to handle the first error.noresource event thrown because of a failure
to do Text To Speech -->

    <form id="dummyForm">
     <block>
          <prompt><audio src="silence.au"/></prompt>
          <goto next="#resetMediaServer"/>
     </block>
    </form>

    <form id="resetMediaServer">
      <block>
     <log>Document state is <value expr="DOCUMENT_STATE"/> </log>
     <log>Form state is <value expr="FORM_STATE"/> </log>
      <submit expr="WEB_SERVER+'handleError.php'" method="get" namelist=" ERROR_EVENT
DOCUMENT_STATE FORM_STATE"/>
      </block>
    </form>

    <catch event="error.badfetch">
      <assign name="ERROR_EVENT" expr="'BAD_FETCH'"/>
      <assign name="WEB_SERVER" expr="'http://mediaServer2/'"/>
      <prompt>We are having technical difficulties. </prompt> <reprompt/>
      <submit expr="WEB_SERVER+'handleError.php'" method="get" namelist=" ERROR_EVENT
DOCUMENT_STATE FORM_STATE WEB_SERVER ERROR_COUNTER"/>
    </catch>

    <catch event="nomatch">
      <assign name="ERROR_EVENT" expr="'NOMATCH'"/>
      <prompt>I did not get that. Please try again</prompt><reprompt/>
    </catch>

    <catch event="noinput">
      <assign name="ERROR_EVENT" expr="'NOINPUT'"/>
      <prompt>I did not hear you.Please try again</prompt><reprompt/>
    </catch>

   <catch event ="noinput nomatch error" count="3">
      <assign name="ERROR_EVENT" expr="'FINAL_TRY'"/>
      <prompt>Sorry,please try again later</prompt>
     <exit/>
   </catch>
</vxml>
```

handleError.php

```
<vxml version="2.0" xml:lang="en-US" application="root.vxml">

<!-- This PHP script handles the different error events and generates a dynamic VoiceXML
document in response to a request by the root.vxml document. When a media server error is
detected, the PHP script sets the properties for a backup media server.
When a HTTP server is detected, the PHP script sets the global variable for the HTTP
server. -- >

<form id="handleError">
<?php
     if($ERROR_EVENT==MEDIA_RESOURCE_UNAVAILABLE){
       if(($FORM_STATE) && ($DOCUMENT_STATE)) {
        echo("
            <property name=\"com.cisco.asr-server\"
value=\"rtsp://mediaServer2/recognizer\"/>
            <property name=\"com.cisco.tts-server\"
value=\"rtsp://mediaServer2/synthesizer\"/>
```

```
<block>
  <log> Assign to a new media server </log>
</block>
       ");
      }
     } elseif($ERROR_EVENT==BAD_FETCH){
        echo("
            <var name=\"WEB_SERVER\" expr=\"'http://backupServer/'\"/>
            <block>
              <log> Assign a backup web server </log>
            </block>
       ");
     } else{
       echo("
           <block>
             <log> Fail to assign a new media server </log>
           </block>
       ");
     }
  ?>

        <property name="fetchtimeout" value="5s"/>
        <property name="timeout" value="5s"/>
        <property name="interdigittimeout" value="3s"/>
        <property name="termchar" value="#"/>
        <property name="bargein" value="true"/>

        <block>
          <assign name="DOCUMENT_STATE" expr="'color.vxml'"/>
          <assign name="FORM_STATE" expr="'colorSelection'"/>
        </block>

        <field name="color">
          <prompt bargein="true">
          Please select one of the following colors. Say red, blue or green
          </prompt>

           <grammar version="1.0" xml:lang="en-US" root="colors">
             <rule id="colors" scope="public">
               <one-of>
                 <item>red</item>
                 <item>blue</item>
                 <item>green</item>
               </one-of>
             </rule>
           </grammar>

            <filled>
                 <prompt>You have selected <value expr="color"/></prompt>
            </filled>
        </field>
   </form>
</vxml>
```

Volume and Rate Control

The attribute cisco-vcrprompt of the <prompt> element is used to control volume and the attribute cisco-rate specifies the rate of prompt playout. To control the playout volume or the rate of playout, set the cisco-vcrprompt attribute to TRUE before you use the <cisco-vcrcontrol> element. To enable volume

and rate control, the <prompt> element uses two attributes cisco-vcrprompt and cisco-rate. The element <cisco-vcrcontrol> allows you to control the playback volume or the playout rate. For details on elements and their attributes, see <vcrcontrol> in Table B-1 in Appendix B, "Cisco VoiceXML Elements: Reference Table.". For information on implementing grammar for volume and rate control, see the "Implementing Grammar for Volume and Rate Control" section on page 1-25.

Note Rate control is not supported for prompts containing TTS or RTSP audio files.

You can control the volume and rate of audio prompt playout as described in the following sections:

Volume Control

You can control the playout volume by setting the output attenuation of the DSP. Volume control is supported only for PSTN ports, and has a range of -14 to +16 dB. The volume level, set in discrete steps of +1 or -1 dB, is valid for the entire duration of a call, unless you change it in the middle of a call. The new volume level is effective from the point at which you make the change. However, if you use the <transfer> tag, the volume level is reset to the default level. Volume control works across all codecs that are supported by Cisco's implementation of VoiceXML 2.1. It also works for IFS, RTSP, HTTP, TTS, recorded prompt playout, and dynamic prompts.

To enable volume control, set cisco-vcrprompt to TRUE and set volume as the value for the action attribute. Use the step attribute to change the playout volume. The step value for volume control is indicated as follows:

- For the Cisco AS5300, the step values range from -14 to +16 dB, where each step represents 1dB.
- For the Cisco AS5350 and the Cisco AS5400, the step values range from -14 to 14 dB, where each step represents 1 dB.
- For the Cisco AS5350XM and the Cisco AS5400XM, the step values range from -14 to 14 dB, where each step represents 1 dB.
- N or + N indicates an increase in volume by N steps.
- -N indicates a decrease in volume by N steps.
- 0 indicates a reset to the configured normal (default) volume.

Note

- If you specify a step value that is out of range, the value chosen will be the maximum or minimum volume within the allowed range, depending on the value specified by you.
- Characters, expressions, and variables are not allowed for a step value.
- Volume control is not supported on the Cisco 3660 router.

- The step value is relative to the default volume configured for the applicable port. The volume for a port is configured as output attenuation using the **output attenuation** command. To configure output attenuation, see the Cisco IOS Voice Configuration Library and *Cisco IOS Voice Command Reference* for details.

On the Cisco AS5400, prompt playout is attenuated from the recorded volume. Since the default output attenuation is 0dB on the Cisco AS5400, record the prompts at a higher volume and then attenuate them in a normal manner.

In the example below, if a caller enters 1, it raises the volume by 2dB. If the caller enters 2, it reduces the volume to the minimum level. If the caller enters 3, it returns the volume to the default level.

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

```
<vxml version="2.0">
<form scope="document" id="get_msg”>
        <cisco-vcrcontrol dtmf="1" action="volume" step="+2"/>
        <cisco-vcrcontrol dtmf="2" action="volume" step="-99"/>
        <cisco-vcrcontrol dtmf="3" action="volume" step="0"/>
        <cisco-vcrcontrol dtmf="4" action="rate" step="+2"/>
        <cisco-vcrcontrol dtmf="5" action="rate" step="-2"/>
        <cisco-vcrcontrol dtmf="6" action="rate" step="0"/>
   <block>
        <prompt cisco-vcrprompt="true" bargein="false" cisco-rate="1">
           <audio src="http://msgserver/YourMessages/NextMsg.au"/>
        </prompt>
   </block>
</form>
```

Rate Control

You can control the playout rate through the step attribute of the <cisco-vcrcontrol> element, allowing you to speed up or slow down the playout rate of an audio prompt. Rate control is implemented by dropping or duplicating packets for the defined rate, and is applicable only to HTTP based prompts for chunk and RAM based playout. Only the G.711 codec is supported.

To enable rate control, set cisco-vcrprompt to TRUE, and set rate as the value for the action attribute. For the attribute cisco-rate, use a value in the range of -4 to +4 to set the absolute playout rate. To set a step change in the playout rate, use the attribute step.

The step value for rate control is indicated as follows:

- Range of step value is from -4 to +4.
- N or + N indicates fast-forward by N steps.
- -N indicates a slowdown by N steps.
- 0 indicates a reset to normal (default) speed.

Note

- If you specify a step value that is out of range, the value chosen will be the maximum or minimum speed within the allowed range, depending on the value specified by you. For example, if you specify the value +6, the value chosen will be +4. If you specify the value -5, the value chosen will be -4.
- Characters, expressions, and variables are not allowed for a step value.

Table 1-1 indicates step values for rate control.

Table 1-1 Step Values for Rate Control

Step Values for Rate Control	Description
Rate = 1	6 packets sent 1 packet dropped
Rate = 2	4 packets sent 1 packet dropped
Rate = 3	3 packets sent 1 packet dropped

Table 1-1 Step Values for Rate Control (continued)

Step Values for Rate Control	Description
Rate = 4	Fastest rate 2 packets sent 1 packet dropped
Rate = -1	6 packets sent 1 packet duplicated
Rate = -2	4 packets sent 1 packet duplicated
Rate = -3	3 packets sent 1 packet duplicated
Rate = -4	Slowest rate 2 packets sent 1 packet duplicated

Prompt Timing

To hold information about the last prompt that is played, use the Cisco specific application variable application.lastprompt$. It holds information about the last prompt that is played through the read-only variables application.lastprompt$.duration and application.lastprompt$.lastrate. For details on Cisco specific application variables, see the "Application Variables" section on page 1-48.

Information on timing for the last prompt is set to the application.com.cisco.lastprompt$ variable when:

- The correct input is collected,

  or

- A nomatch event is thrown,

  or

- A noinput event is thrown,

  or

- Media play fails.

Examples

In the example below, when <submit> tries to use the application variable lastprompt$, the timing information for prompt A is not defined because prompt A is still queued for playing when <submit> is executed. In this case, the lastprompt$ variable has value "undefined", or it may still contain timing information of the previous prompt from the last input collection.

```
<block>
    <prompt A/>
    <submit application.com.cisco.lastprompt$.../>
```

In the example below, a prompt is queued before a <goto> statement.

- The lastprompt$ variable resets to "undefined" and does not target prompt B, if <goto> transitions to a different application.

- The lastprompt$ variable targets prompt B, if <goto> transitions to the same application with a new VoiceXML document in the same application, and if no prompts are available for input collection for the new VoiceXML document.
- The lastprompt$ variable does not target prompt B if <goto> transitions to the same application, and prompts are available for input collection for the next VoiceXML document within the application.

```
<block>
<prompt B/>
<goto next="next.vxml"/>
```

If a digit is handled by an item that has volume and rate control, it is unavailable to a field or menu that is active at the same time.

Type-ahead Support

Cisco VoiceXML includes a type-ahead buffer that holds DTMF digits collected from the caller. When the VoiceXML form interpretation algorithm collects user DTMF input, it uses the digits from this buffer before waiting for further input. For example, if the caller knows ahead of time that the document will prompt for account, PIN, and destination, and ask the caller to listen to the welcome message, then the caller can enter all the digits in advance without waiting.

In the <prompt> element, a set of audio tags with bargein enabled is played out as a group. The group of prompts is interrupted when the digit is received by the gateway, independent of when the digit is pulled out of the type-ahead buffer. A bargeinable prompt will not start playout if there are digits in the type-ahead buffer. This means multiple prompts may not be played because of a single digit entering the type-ahead buffer.

For a simple example, if a noninterruptable prompt (bargein=FALSE) is played, the set of digits received during that playout are put into the type-ahead buffer without interrupting the prompt. If the next field plays out a prompt with bargein=TRUE, that prompt will not play if there are digits in the type-ahead buffer.

For a more complex example, consider a document that plays an interruptible prompt without collecting digits. After the document transitions to another document, the new document plays an interruptible prompt while collecting a single digit, and finally plays a third interruptible prompt. If the caller enters a digit during the first prompt, it interrupts that prompt. When the second document is loaded, the initial interruptible prompt is not played because there is a digit in the type-ahead buffer. The third prompt is played because the digit collection consumed the digit from the type-ahead buffer.

A prompt with volume and rate control can have bargein=FALSE. For bargein=TRUE, the digits that are not handled or ignored by <cisco-vcrcontrol> stop prompt playout. If the field collects digits against a pattern with bargein enabled and <cisco-vcrcontrol> active, the prompt is not interrupted until the entire pattern is matched.

Buffer Control and Flushing

The VoiceXML specification does not provide information on explicit control over the type-ahead buffer. The type-ahead buffer is always enabled when a call comes in. It is flushed if a <grammar> tag in a field uses a *regexp* that does not match any digits in the buffer. This causes a nomatch event each time digits pulled from the buffer do not match the regular expression. By default, the typeahead buffer is automatically flushed when processing a nomatch event or a <reprompt> element. To override this default behavior, set the com.cisco.autoflush property to false where this change is required.

If the call is transferred to a third party with *long-#* enabled, the type-ahead buffer is flushed looking for the long-# digit in the type-ahead buffer. Therefore, a *long-#* entered during call setup will disconnect the call as soon as the setup completes and the system checks the buffer. Digits from the buffer are not played out toward the third party.

cisco-typeaheadflush Attribute for <prompt>

The default value of cisco-typeaheadflush is false. A false value means that the typeahead buffer is not flushed after the prompt plays out. If the prompt is bargeinable, the digit which barges in is not flushed.

com.cisco.autoflush Property

The type-ahead buffer is flushed by default for a nomatch or reprompt event. If the autoflush property is set to *false*, the type-ahead buffer is not flushed. For example, consider a document that has a nonbargeinable prompt followed by a collection pattern of 125. If a caller enters 1234 during the prompt, the document tries to collect 125 based on the collection pattern. The document reads 1, 2, 3, then gets a nomatch and flushes the 4.

If there is a nomatch event and audio is played inside it, the flush occurs immediately after the nomatch. The system handles the event which means that the caller can re-enter digits during the prompt playout.

Type-ahead Buffer with <goto> or <transfer> to an Application

Tcl applications do not utilize the type-ahead buffer, so calls handed from Tcl applications do not support type-ahead until the call is being handed from the VoiceXML application. If the call is handed between VoiceXML documents either using <goto> or using the <transfer> tag with an outbound VoiceXML application, the type-ahead buffer is active. It is not flushed during the handoff.

Type-ahead Interaction with Barge-In

A bargeinable prompt will not start playout if there are digits in the type-ahead buffer. See the "Prerecorded Audio Prompts" section on page 1-9 for more details on this operation.

If bargein occurs during any prompt in a sequence, all subsequent prompts are not played, even those prompts whose bargein attribute is set to false. In the following example, if a digit is received during the first prompt playout, the second and third prompts will not play.

```
<field name="field1" >
 <grammar type="application/grammar+regex">.*</grammar>

 <prompt bargein="false">
 <audio src="press1_2.au"/>
 </prompt>

 <prompt bargein="true" >
 <audio src="long_2m.au"/>
 </prompt>

 <prompt bargein="false">
 <audio src="en_welcome.au"/>
 </prompt>
.
.
.
</field>
```

A prompt with volume and rate control can have a bargein value of false. Digits that are ignored will stop prompt playout if the bargein value is true. Digits that are not used for volume and rate control are put in the typeahead buffer. If a field collects digits against a pattern with bargein enabled and, volume and rate control active, the prompt is uninterrupted until the entire pattern is matched.

The typeahead buffer is always flushed if there are digits in the typeahead buffer when playout starts for a prompt with volume and rate control.

User Input

VoiceXML allows two types of user input, voice and DTMF. Cisco VoiceXML accepts both types of input and processes them either through an external media server or through the voice gateway itself, depending on the type of input and grammar used to collect the input. For user input recognition based on DTMF, the voice gateway is capable of recognizing that input without the help of any external media recognition devices. Regex is the only grammar format supported for DTMF based recognition. Recognition of user input such as ASR or TTS involves an external media server working in conjunction with the voice gateway. The grammar formats required to process ASR and TTS based input are dependent on the support provided by the media server vendor. All external media servers are required to support at least the W3C XML grammar format. Media server vendors may support other standard or proprietary grammar formats such as Nuance's GSL grammar format.

Note The regex grammar used for DTMF input is a Cisco grammar and is not supported by vendors on their media servers.

Voice Input

To handle voice input, an external media server is required to work with Cisco voice gateway. The media server conducts automatic speech recognition (ASR) and communicates the interpretation back to the VoiceXML interpreter, running on the voice gateway, for processing.

Cisco VoiceXML only supports W3C XML grammar for speech recognition. The user can dynamically change the media server for the next ASR by setting the Cisco specific VoiceXML property "com.cisco.asr-server" in the VoiceXML script. For example, the statement,

<property name= "com.cisco.asr-servervalue"= "rtsp://asr-server/recognizer"/> sets "rtsp://asr-server/recognizer" as the external media server for the next ASR, and continues with the same setting until the property is set again with a different server.

DTMF Input

To handle DTMF input, Cisco VoiceXML uses either an external media server or the Cisco voice gateway, which is capable of handling all DTMF applications.

For an external media server to collect DTMF input, use W3C XML grammar. For a Cisco voice gateway to collect DTMF input, use Cisco specific DTMF grammar.

Note If you use the external media server and a Cisco voice gateway to collect input in the same application, the VoiceXML interpreter throws an error.badfetch event.

The user can dynamically change the media server for the next DTMF recognition by setting the Cisco proprietary VoiceXML property *com.cisco.asr-server* in the VoiceXML script.

Grammar Support

Cisco VoiceXML supports two types of grammar, W3C XML grammar and Cisco specific DTMF grammar. W3C XML grammar is used to collect both voice and DTMF input. Cisco specific DTMF grammar is only used to collect DTMF input.

Note If you use both grammars in the same application, the VoiceXML interpreter throws an error.badfetch event.

For all grammars that are automatically generated by the VoiceXML interpreter from <choice>, <option> and builtin, W3C XML grammar is the default. However, if at least one specific Cisco DTMF grammar is used in the application, all the automatically generated grammars are in Cisco specific DTMF grammar format.

XML Grammar

Cisco's VoiceXML interpreter supports the standard W3C XML grammar format. For user input using W3C XML grammar, an external media server is required. The interpreter does not process grammars; it checks the grammar for syntax. The external media server processes the grammar. Deviations from W3C XML grammar introduced by the media server are imposed on Cisco's implementation of VoiceXML. For example, with the current media server integrated with Cisco VoiceXML, the deviations are:

- The external media server only supports HTTP clients. Because of this restriction, the *src* attribute only contains an HTTP URI for referencing external grammar.
- TFTP, FTP, flash, or other types of URI are not supported.

For more information on W3C XML grammar format, see *Speech Recognition Grammar Specification for the W3C Speech Interface Framework* (W3C Working Draft, 20 August 2001).

Cisco DTMF Grammar

The Cisco dual tone multiple frequency (DTMF) grammar supported is a regular expression (Regex) grammar. The media type of the grammar is *application/grammar+regex*. Cisco DTMF grammar has the following limitations:

- It supports inline grammar only.
- It cannot be used as a form grammar.
- It cannot be given a document scope when used as a menu grammar.

Cisco DTMF grammar supports the following metacharacters:

Metacharacter	Description
.	Matches any single character. For example, Cisco DTMF grammar with a regular expression <grammar type= “application/grammar+regex”>1408.....</grammar> matches a seven digit phone number with the leading area code 1408.
\	The quoting character. It removes any special meaning from the following character and treats it as an ordinary character. For example, <grammar type= “application/grammar+regex”>*</grammar> matches a literal asterisk (star) key, not the asterisk repetition operator.

Repetition Operators

?	Matches zero or one occurrence of the character or regular expression immediately preceding. For example, <grammar type= “application/grammar+regex”>408?</grammar> matches 40, 4088, 40888, 408123, 4083456. The match occurs for 408, 4088, 40888 and also for 408 followed by other extra digits that occur after 408.
+	Matches one or more occurrences of the character or regular expression immediately preceding. For example, <grammar type= “application/grammar+regex”>408+</grammar> matches 408, 4088, 40888, 408123, 408883456. The match occurs not only for 408, 4088, 40888 but also for 408 followed by other extra digits that occur after 408.
*	Matches zero or more occurrences of the character immediately preceding. For example, <grammar type= “application/grammar+regex”>408*</grammar> matches 40, 4088, 40888, 408123, 4083456. The match occurs not only for 408, 4088, 40888 but also for 408 followed by other extra digits that occur after 408.

Only the previously listed metacharacters are supported. When an unsupported metacharacter is used, no error will be triggered. However, input recognition will produce unexpected results.

In addition to matching the original pattern, the DTMF grammar matches the original pattern followed by extra digits. Matching of extra digits occurs only if the repetition operators are at the end of a pattern.

Regular expression for DTMF grammar allows you to use only empty spaces instead of the operator | to join characters.

For example:

- To join * and .+ use an empty space instead of the operator |. See the following example:

  <grammar type= “application/grammar+regex”>* .+</grammar>

The <field> builtin types digits and number accept any nondigit input. A nomatch event is not generated.

Implementing Grammar for Volume and Rate Control

The element <cisco-vcrcontrol> is used for volume and rate control of an audio prompt playout. For more information on using <cisco-vcrcontrol>, see the element <cisco-vcrcontrol> in Table B-1 in Appendix B, "Cisco VoiceXML Elements: Reference Table" and the "Volume and Rate Control" section on page 1-16. In-line DTMF grammar is used. External and ASR grammar is not supported.

Note The dtmf attribute of <cisco-vcrcontrol> does not take a regular expression. It takes only a single digit.

Applicable volume and rate control grammars are active when the interpreter plays a prompt <cisco-vcrcontrol> turned on. Volume and rate control can be applied to any prompt inside a VoiceXML document. For a prompt inside <field> or <menu>, volume and rate control grammar takes precedence over DTMF input grammar if both are active at the same time.

Volume and rate control grammar activates in the following order:

1. Grammar for the form item
2. Grammar for the form
3. Grammar for the document
4. Grammar for the root document

The grammar closest to the applicable item is activated.

Some dependencies for the scope of volume and rate control grammar are:

- Volume and rate control grammars for a form item are always scoped to its form item.
- Volume and rate control grammars for form are assigned a dialog scope by default. This enables the grammars to be active only when the user is in the form.
- Volume and rate control grammars assigned to a document scope are active when the user is in the document and in the applicable leaf documents.
- Volume and rate control grammars for menu are assigned a dialog scope by default, and are active only when the user is in the document and in the applicable leaf documents.
- For prompts in the event handler, the scope of the original event applies to volume and rate control grammar.
- For prompts in the <filled> element:
  - The scope of the field item applies to volume and rate control grammar if the <filled> element is inside a field item.
  - The scope of the form applies to volume and rate control grammar if the <filled> element is inside a form.

Recording Support

This section contains the following information about recording support:

- Properties for <record>, page 1-30
- Typical Call Flow Using Recording, page 1-31

For recording, the following four recording destinations are supported:

- HTTP
- RAM
- RTSP
- SMTP

The recording is stored as a VoiceXML field item variable and can be referenced in the VoiceXML application document. The VoiceXML browser plays an audio file for a specific URI (<audio src=filename>) in a VoiceXML document and the recorded input can be played back using <value> or by submitting it to the server with <submit method= “post+enctype”= “multipart/form-data”/> saved as an audio file.

Note In Cisco IOS Release 12.4(11)T and later releases, use the <audio> element to play back the recorded input. The <value> element is supported for backward compatibility.

Recognition of grammars during recording is supported in Cisco IOS Release 12.4(15)T and later releases. On the POTS call leg, recording with ASR grammar is supported in the G.711 u-law codec only. On the VoIP call leg, recording with ASR grammar is not supported.

Note In Cisco IOS Release 12.3, a VoiceXML application can submit a RAM recording to an HTTP server using the POST method with an enctype of “audiobasic.” Later releases of Cisco IOS software, however, only support the “multipart/form-data” type for <submit>. A VoiceXML document written to support the “audiobasic” MIME type for <submit> in Cisco IOS Release 12.3 is not supported in later releases.

RTSP, SMTP and HTTP recording is done by streaming the voice data to the specified external server. The “cisco-dest” attribute is used to specify the target destination. If “cisco-dest” is not specified, the default target destination is RAM.

Using cisco-dest

The attribute “cisco-dest” is a specific Cisco attribute that points to a URL specifying a recording destination. The three recording choices are:

- HTTP recording: <record cisco-dest = “http://”>

  HTTP recording is done by streaming voice data to an HTTP server. The <record> name variable is not applicable. The HTTP URL must point to an application on the server which accepts the data and writes it into a file. Playback can be achieved by referring to the HTTP URL specified in <record>.

- RTSP recording: <record cisco-dest = “rtsp://”>

  Specifying the recording URI causes the message to be streamed directly to an RTSP server. Here, the <record> name variable is not applicable. The RTSP URL must point to an application on the server which accepts the data and writes it into a file. Playback can be achieved by referring to the RTSP URL specified in <record>.

  Recording status is returned by shadow variable name$.status (see Shadow Variables, page 1-28).

- SMTP recording: <record cisco-dest = “mailto://”>

The message streams in real time to an SMTP server.

To write a VoiceXML document user interface for a simple voice mail system, use the <record> element with the attribute cisco-dest.

You can use <record cisco-dest> to record a voice message and deposit it at an external server for later retrieval. This extension works with another Cisco extension of <record> so that it can interoperate with the .au file format and with PC clients.

Recording ends when:

- The user presses any DTMF key, or
- Final silence is detected, or
- There is a match in record grammar, or
- The maximum recording time or size is reached.

Table 1-2 gives information on Cisco-specific applications of the attributes for <record>:

Table 1-2 <record> Attributes

Attribute	Use
name	Variable name of RAM recording.
beep	Not supported. **Note** In Cisco IOS Release 12.4(15)T and later releases, the beep attribute is supported if the **vxml version 2.0** command is entered. There is no codec limitation on the PSTN call leg. The G.711 u-law, G.711 a-law, G.729r8, G.723r53, and G.723r63 codecs only are supported on the VoIP call leg.
maxtime	Maximum time in seconds for recording duration. The recording is cut to maxtime if it exceeds the maxtime specified. If this attribute is missing, the default is 30 seconds.
finalsilence	Allows a gateway to terminate a recording after a defined length of silence. The gateway waits for a few seconds before terminating the recording. The final silence feature is disabled by default. It is also disabled for a value of zero. Final silence is enabled by using the finalsilence attribute. Final silence is not supported for RTSP recording. **Note** Voice activation detection (VAD) must be enabled on the VoIP dial peer when final silence detection is needed to terminate a voice recording. See the *Cisco IOS Tcl and VoiceXML Application Guide* for details.
type	MIME type of recording which describes the audio file format and codec type. Codec type is specified as a parameter in name=value format based on RFC1341 “Content-Type.” File format only supports audio/basic. If this attribute is missing, the default is “audio/basic; codec=G.711ulaw”. **Note** In Cisco IOS Release 12.4(11)T and later releases, the supported file formats are audio/basic, audio/x-alaw-basic, and audio/x-wav.
dtmfterm	“true” or “false”; if true, detection of any DTMF key terminates the recording. If this attribute is missing, the default value is “true.”
cisco-dest	Recording URL: HTTP, RTSP, SMTP or “mailto.” If missing, the default is RAM.

Table 1-2 <record> Attributes (continued)

Attribute	Use
cisco-codec	Codec type of recording. If this attribute is missing, the default is "G.711 ulaw."
cisco-recordbeep	"true" or "false". When the value is set to "true," a beep sound is sent back to the user every x seconds during recording. The number of seconds between beeps is specified in the cisco-recordbeepinterval attribute. **Note** In Cisco IOS Release 12.4(15)T and later releases, all codecs are supported on the PSTN call leg. The G.711 u-law, G.711 a-law, G.729r8, G.723r53, and G.723r63 codecs only are supported on the VoIP call leg.
cisco-recordbeepinterval	Time in seconds to specify the interval between beeps for the cisco-recordbeep attribute. The default value is 3 seconds; the minimum value is 2 seconds. If a value smaller than the minimum is specified, it will be changed to the minimum value. **Note** This attribute is supported in Cisco IOS Release 12.4(15)T and later releases.

Shadow Variables

Standard shadow variables from VoiceXML 2.1 are supported, including the following:

- **name$.duration**: Duration of the recording in milliseconds
- **name$.size**: Size of the recording in bytes
- **name$.termchar**: If *dtmfterm* is true, this shadow variable holds the pressed key.
- **name$.maxtime**: Boolean which is true if the recording was terminated because the maxtime duration was reached
- **name$.recording**: Reference to the recording, or undefined if no audio is collected
- **name$.recordingsize**: Size of the recording in bytes, or undefined if no audio is collected
- **name$.recordingduration**: Duration of the recording in milliseconds, or undefined if no audio is collected.

RAM Recording

You can play a RAM recording by referring to its variable name and using <prompt> and <audio expr= "nnn"/> together where "expr" is the recording name variable.

Note In versions earlier than VoiceXML 2.0, RAM recording can be played using <prompt> and <value expr="nnn"/> only. In VoiceXML 2.0 and later versions, use <prompt> and <audio expr="nnn"/> to play RAM recordings.

Here is an example of playing a recording:

```
<vxml>
<form>
<record name="myrec" beep="false" maxtime="10s" dtmfterm="true"
type="audio/basic;codec=g711ulaw">
</record>
<block>
<prompt> Your recording is <audio expr="myrec" /></prompt>
```

```
</block>
</form>
</vxml>
```

The RAM recording can then be saved to a server, as shown in the following example, where the Record.php script saves the file and returns a VoiceXML document.

```
<block>
<submit next="http://myserver/mypath/Record.php"
namelist="myrec" method="post" enctype="multipart/form-data"/>
</block>
```

Playing recording by referring to the RTSP, HTTP, or SMTP recording variable in <value expr> results in an error; an error.semantic event is thrown and an error message “cannot playback a streaming recording” is generated.

The other types of recording (RTSP, HTTP and SMTP) cannot be played back by referring to the <record> name variable because they are directly streamed to the external server.

For RTSP and HTTP, the recording can be played back by <audio src= “nnn”/> where “src” specifies the recording URL.

HTTP Recording

For HTTP recording, a PHP script must reside on the server. It is used to copy the recorded audio files into one of the server directories.

Here is an example of a VoiceXML 2.0 script for HTTP recording (nonstreaming mode) and the PHP servlet that is used to upload the audio files into a server directory.

Example

```
<form>
<record cisco-dest="http://goa/php/recordhttp.php?audiofile=httprecordaudio.au"
dtmfterm="true"/>
<block>
Your recording is <audio src="http://goa/httpaudio/httprecordaudio.au"/>
</block>
</form>
```

In this example of HTTP recording, the PHP script http://goa/php/recordhttp.php?audiofile=httprecordaudio.au copies the audio file into the http://goa/httpaudio/httprecordaudio.au file.

Exception Handling

In addition to supporting standard error types in VoiceXML 2.1, the following application-specific error types are supported:

- error.badfetch.rtsp.nnn

  This event is thrown when a RTSP server communication error is generated; nnn is the corresponding protocol error code.

- error.unsupported.format

  This event is thrown when the user specifies an unsupported audio format, codec type, or recording destination URL. If the codec specified by the user mismatches the codec type that is configured for VoIP, the VoIP configured codec is used for the recording and a warning message is generated to notify the user of the codec being used.

Note In Cisco IOS Release 12.4(11)T and later releases, the following error events are supported:
error.badfetch.http.response_code
error.badfetch.protocol.response_code

Properties for <record>

You can use properties to specify the default attribute values for <record>. See the "Property" section on page 1-55 and Table 1-9 for a list of Cisco specific properties that are supported for <record>. These properties can be specified at <vxml>, <form>, and <record> level, and allow the user flexibility in specifying a default value. The Cisco implementation of VoiceXML 2.1 supports message disposition notification (MDN) and delivery status notification (DSN).

The MDN address is specified by the following specific Cisco VoiceXML Properties:

- com.cisco.mta.send.mdn_request

  Setting com.cisco.mta.send.mdn_request to TRUE sends the MDN request.

- com.cisco.mta.send.mdn_hostname and com.cisco.mta.send.mdn_username

  If these properties are not specified, the MDN address is composed by configuring the **mta send return-receipt-to hostname** and **mta send return-receipt-to username** Cisco IOS commands. If these commands are not configured, the MDN address is composed by configuring the **mta send postmaster email-address** command. If this command is not configured, the **mta send mail-from** command is used. For information on configuring these Cisco IOS commands, see the *Cisco IOS Tcl IVR and VoiceXML Application Guide* for your Cisco IOS release and the *Cisco IOS Fax and Modem Services over IP Application Guide*.

DSN is specified by the following specific Cisco VoiceXML properties:

- com.cisco.mta.send.dsn_delay
- com.cisco.mta.send.dsn_success
- com.cisco.mta.send.dsn_failure

The subject header is specified by the com.cisco.mta.send.subject property. If this property is not defined, the subject field of the e-mail header is set by using the **mta send subject** Cisco IOS command.

The following specific Cisco properties are also supported for platform-specific settings for SMTP:

- com.cisco.mta.send.from_username
- com.cisco.mta.send.from_hostname
- com.cisco.mta.send.server
- com.cisco.mta.send.origin_prefix

Note

- The properties used to specify MDN and DSN have equivalent Cisco IOS commands. If the specific Cisco VoiceXML properties and their equivalent Cisco IOS commands are specified simultaneously, the VoiceXML properties take precedence over the CLI.
- For MDN, the username and the hostname must be specified to form a valid e-mail address.
- The scope of each property conforms to the VoiceXML Version 2.0. Each property can be defined at the application root document level, document <vxml> level, dialog<form> or <menu> level, or the form item <record> level.

Typical Call Flow Using Recording

The following call flow illustrates recording:

1. The voice gateway receives a setup indication and hands the call to a generic VoiceXML document.
2. The generic VoiceXML document requests a customized VoiceXML document from the VoiceXML server based on incoming call information (ANI, DNSI, RDN, and so on).
3. The VoiceXML server creates a customized VoiceXML document based on the call information provided a nd sends it to the gateway.
4. The gateway executes the document, which prompts the user to record a voice message.
5. The gateway records the message and stores it in local RAM with G.723.1 and G.711 u-law encoding.
6. When recording is complete, the user is prompted to review the message, or to submit the message to the VoiceXML server. It is also possible that the user can simply hang up.
7. When the user submits the voice mail message, the gateway's VoiceXML browser submits the voice message as an .au file to a specified URL using the HTTP POST method.

   The VoiceXML application must be able to handle the case the user hanging up before choosing to submit the voice message.
8. When the VoiceXML server receives the message, it stores the message in the appropriate mailbox.
9. When the voice message is successfully stored, the application instructs the media stream process to delete the local copy of the voice message.
10. The VoiceXML document disconnects the call.

<cisco-data> Element

The <cisco-data> element allows an application to load data from a server for handling a call. It allows voice applications to send and receive information from an external server when answering and transferring calls.

For call center applications, the VoiceXML <transfer> element is limited because it can only inform the HTTP server if the call does not connect because of:

- Busy tone,

  or
- No response from the called party,

or if the call disconnects because of:

- Connection hangup,

  or
- Maxtime limitation.

VoiceXML 2.0 does not support an event when the <transfer> connection is made and the two parties start talking to each other. However, this limitation can be overcome by using a Cisco VoiceXML hybrid script in which the Tcl script performing the transfer can use the <cisco-data> element to send a GET or POST HTTP request to a web server.

The VoiceXML interpreter running on a gateway uses the <cisco-data> element to post data to a server, and receives that data back from the server without transitioning to a new VoiceXML document. The <cisco-data> element occurs as executable content, or as a child of <form> or <vxml>. It has the same scoping rules as the <var> element. If a <data> element has the same name as a variable declared in the same scope, that variable is assigned a value retrieved by the <data> element.

If the *name* attribute is specified and the data is not retrieved, the VoiceXML interpreter throws an error *event error.badfetch.http.nnn* to indicate a fetch failure.

Note

- Headers from HTTP responses are not available to a VoiceXML script.
- The <cisco-data> element supports only text messages. It fetches strings from a server and does not handle binary data. If binary or null data is encountered in a response, the information returned to the VoiceXML interpreter is truncated. For example, "My data is <binary data> received on Tuesday" is truncated to "My data is."
- Data received is truncated if it exceeds 20 KB.

The attributes of <cisco-data> are:

- src

  The URI specifying the location of data.

- name

  The name of the variable that contains the retrieved data. If this field is deleted, data can only be sent to the server, not retrieved.

- expr

  The URI specifying the location of the data. The URI is dynamically determined.

- method

  Similar to the method attribute of the <submit> element in the VoiceXML 2.0 specification.

- namelist

  Similar to the namelist attribute of the <submit> element in the VoiceXML 2.0 specification.

- enctype

  Similar to the enctype attribute of the <submit> element in the VoiceXML 2.0 specification.

- fetchhint

  Similar to the fetchhint attribute of the <submit> element in the VoiceXML 2.0 specification.

- fetchtimeout

  Similar to the fetchtimeout attribute of the <submit> element in the VoiceXML 2.0 specification.

- fetchaudio

  Similar to the fetchaudio attribute of the <submit> element in the VoiceXML 2.0 specification.

- maxage

  Indicates that the document is willing to use content whose age is no greater than the specified time in seconds. Compare to max-age in HTTP 1.1, RFC 2616. The document is not willing to use stale content, unless maxstale is also provided. If not specified, a value derived from the innermost relevant maxage property, if present, is used.

- maxstale

  Indicates that the document is willing to use content that has exceeded its expiration time. Compare to max-stale in HTTP 1.1, RFC 2616. If maxstale is assigned a value, then the document is willing to accept content that has exceeded its expiration time by no more than the specified number of seconds. If not specified, a value derived from the innermost relevant maxstale property, if present, is used.

DTD for <cisco-data>

```
<!ELEMENT cisco-data EMPTY>
<!ATTRLIST cisco-data
       src              %uri                 #IMPLIED
       name           NMTOKEN     #IMPLIED
       expr            %expression;    #IMPLIED
       %cache.attrs;
       %submit.attrs; >
```

Example

This example shows the use of <cisco-data> in posting information to a server and receiving that data back from the server.

In this example:

- Variables a and b are assigned values. The variables and values are sent as avpairs in the body of the HTTP message.
- The HTTP server that receives the data sent to it is identified by the src attribute.
- The namelist attribute identifies the list of variables that are sent to the HTTP server.
- The HTTP method is the POST method.
- The received data is the body of the message received in the response from the server, and is stored in the MyData variable.
- The HTTP server does not support multipart data. In the <cisco-data> element, the data assigned to the variable name can be submitted back to the server for processing. However, Cisco IOS release 12.3T only supports the “multipart/form-data” type for <submit>. The enctype must be *enctype= “multipart/form-data*” if <cisco-data> is to be submitted back to the server.

```
<vxml version="2.0">
    <form>
        <var name="a" expr="123"/>
        <var name="b" expr="456"/>
        <block>
            <cisco-data
                src="http://townsend.cisco.com/cgi-bin/rama.tcl"
                name="MyData"
                method="post"
                namelist="a b">
            </cisco-data>
        </block>
        <block>
           <log> Yes. DATA received successfully.
               <value expr="MyData"/>
           </log>
    </block>
    </form>
</vxml>
```

Transfer Support

Note In Cisco IOS Release 12.4(11)T and later releases, if the **vxml version 2.0** command is entered, the behavior of blind transfer conforms with VoiceXML 2.0 specifications and does not behave like consultation transfer.

The <transfer> element can place a call to a destination for the caller. This tag triggers the outbound dial peer selection and call setup.

Recognition of ASR and regular expression grammars during transfer is supported in Cisco IOS Release 12.4(15)T and later releases. A call handled between VoiceXML documents using the <transfer> tag with grammar can support the G.711 u-law codec only on the VoIP dial peer.

Note If an invalid number is used for a transfer element, the interpreter throws an error.connection.baddestination event.

Cisco Longpound Attribute

The most common case is to connect a caller through the Cisco gateway to a third party over any network the gateway supports. For this case, Cisco has implemented a cisco-longpound attribute for <transfer>, *long-#*, to allow the caller to terminate the connection by holding the pound (#) key down for more than a second. The Boolean value of this attribute specifies if *long-#* is used to terminate a transfer.

In the case of a bridging transfer, holding down the # key for more than a second resumes the session with the interpreter.

Note In releases earlier than Cisco IOS Release 12.4(15)T, any <grammar> inside a <transfer> element is ignored.

Cisco-newguid Attribute

If you are a VoiceXML document writer, you can use the cisco-newguid attribute to specify that the call requested by the <transfer> tag will be made using a new globally unique identification (GUID) instead of inheriting the GUID of the incoming call leg. The use of this attribute is significant when a single inbound call generates multiple outbound calls (for example, in a debit card application, the user can make several calls in a single session).

For more information on GUID, see VSA number 24 (h323-conf-id=value) in Table 2 of the *RADIUS Vendor-Specific Attributes Voice Implementation Guide*.

You can specify the cisco-newguid attribute if:

- You need to identify the different calls that were made when reconciling your billing records,

or

- If two back-to-back calls are made to the same terminating gateway. In this case, if the same GUID is used for two back-to-back calls, and if the terminating gateway receives a request for setting up the second call before it finishes tearing down the first call, the duplicate GUID causes the terminating gateway to reject the second call.

The cisco-newguid attribute resolves to a boolean value with a default value of false.

For example:

```
<transfer...cisco-newguid="true">
  !assign a boolean value directly

<transfer...cisco-newguid="new_guid">
  !new_guid is a variable that contains the boolean value "true" or "false".

<transfer ... cisco-newguid="call_count > 0">
  !where call_count is a number variable tracking the number of calls made so far for this
session.
```

Continuous Fax Detection and Transfer

Cisco VoiceXML supports continuous CNG tone detection during all phases of a call. While the application is listening for CNG tones, VoiceXML scripts are being loaded and executed to play prompts, record voice, process DTMF input, etc.

When the VoiceXML script is running, the application continuously listens for CNG tones. When CNG is detected, a CNG tone detection event is passed to the VoiceXML application which in turn generates the VoiceXML event **com.cisco.fax.cng**.

This event is specific to Cisco's VoiceXML implementation. The VoiceXML document has to define a catch event handler to process this fax event. It may choose to respond to the fax and send it through T.37 Store & Forward by a blind transfer invoked by **<transfer dest=”fax://[dnis-to-match-dp]” bridge=”false” cisco-mailtoaddress=”[email-id]”>** in a block outside the catch event handler of **com.cisco.fax.cng**.

Note Only a blind transfer and setting the bridge attribute to false (**bridge=”FALSE''**) allows the VoiceXML document to send the fax through T.37 Store and Forward.

Fax Mailto Addressing

Cisco VoiceXML allows full control of e-mail addresses by providing the cisco-mailtoaddress attribute in the <transfer> element and the e macro in the MMoIP dial peer. The cisco-mailtoaddress variable in the transfer element replaces the e in the mailto address. In the VoiceXML script, the e-mail address can be constructed in the cisco-mailtoadress attribute through one of the following:

- DNIS
- RDNIS
- A combination of DNIS and RDNIS
- A number generated during the session (for example, dtmf entry) for direct calling by the document.
- A string that represents a valid e-mail address.

The VoiceXML document defines an e-mail address through the cisco-mailtoaddress variable in the <transfer> element.

Note Make sure that you do not pass @domain again in the cisco-mailtoaddress attribute in the VoiceXML script if it is already configured in **sessiontarget mailto** in MMoIP dial peer.

The VoiceXML application maps the cisco-mailtoaddress attribute to the e-mail address only if the e macro is configured in the MMoIP dial peer. By default, if the cisco-mailtoaddress variable is not specified in the transfer element, the VoiceXML application maps the DNIS to e. The attribute cisco-mailtoaddress is optional in the <transfer> element. If the attribute is not specified, the VoiceXML application maps the e-mail ID to DNIS. Cisco VoiceXML is also capable of setting the MMoIP session mail-to e variable to DNIS, RDNIS, or a string that represents a valid e-mail address.

e is capable of accepting any of the following e-mail addresses:

- user@domain.com
- user
- 1234@domain.com (where 1234 is DNIS)
- 5678@domain.com (where 5678 is RDNIS)
- 12345678@domain.com (where 12345678 is DNIS+RDNIS)
- user1!domain1@domain2.com
- FAX= 12345678@domain.com (T.33 subaddressing strings are also accepted.)

Note If e is not specified in the **session target mailto** command, and the cisco-mailtoaddress attribute is specified in the transfer element of the fax detection document, then whatever is specified in the MMoIP dial peer takes precedence; the cisco-mailtoaddress attribute is ignored.

Example

This example shows a Cisco VoiceXML script (VoiceXML 2.0) used for CNG detection.

```
<?xml version="1.0"?>
<vxml version="2.0" base="tftp://audio directory path/">

<catch event="com.cisco.fax.cng">
        <log>!!!!!!! Got com.cisco.fax.cng !!!!!! </log>
        <goto next="#transferToFax"/>
</catch>

<catch event="nomatch noinput">
        <goto next="#transferToPhone"/>
</catch>

<catch event="telephone.disconnect.transfer">
        <log>!!!!!!! Got telephone.disconnect.transfer </log>
</catch>

<catch event="telephone.disconnect.hangup">
        <log>!!!!!!! Got telephone.disconnect.hangup </log>
</catch>

<catch event="error.badfetch">
       <log>!!!!!!! Got ERROR.BADFETCH !!!!!! </log>
</catch>


<var name="phone_num"/>
<var name="mydur"/>
<var name="defaultPhoneNum" expr="session.telephone.dnis"/>
<var name="junkvar"/>

<property name="timeout" value="9s"/>
```

```
    <form id="main">
        <field name="get_phone_num" type="number">
        <dtmf type="regex">.......</dtmf>

        <prompt bargein="true"> <audio src="audio/menu.au"/> </prompt>


        <filled>
            <assign name="phone_num" expr="get_phone_num"/>
            <log>The number collected is <value expr="phone_num"/></log>
            <goto next="#transferToFax"/>
        </filled>
        </field>
</form>

<form id="transferToFax">
    <block>
        <assign name="phone_num" expr="5550112"/>
        <log>Transferring to <value expr="phone_num"/></log>
    </block>
        <transfer name="mycall"   destexpr="'fax://'+ phone_num"  bridge="false"
connecttimeout="15s" maxtime="180s" cisco-longpound ="true" cisco-mailtoaddress="user
name" >
        <filled>
            <assign name="mydur" expr="mycall$.duration"/>
            <log>The value in mycall is <value expr="mycall"/></log>
            <log>Duration of call is <value expr="mydur"/></log>
        </filled>
        </transfer>
</form>

<form id="transferToPhone">
    <block>
        <assign name="phone_num" expr="5550112"/>
        <log>Transferring to <value expr="phone_num"/></log>
    </block>
        <transfer name="mycall"  destexpr="'tel: '+ phone_num" bridge="true"
connecttimeout="150s" maxtime="180s" cisco-longpound ="true" >
        <filled>
            <assign name="mydur" expr="mycall$.duration"/>
            <log>The value in mycall is <value expr="mycall"/></log>
            <log>Duration of call is <value expr="mydur"/></log>
        </filled>
        </transfer>
</form>

<form id="transferToFax">
        <property name="timeout" value="500s"/>
        <field name="get_phone_num" type="number">
        <dtmf type="regex">.......</dtmf>

        </field>

</form>

</vxml>
```

Transfer Form Item Variable

The transfer form item variable is used to store the outcome of the transfer attempt. In VoiceXML 2.0, the possible values are: “busy”, “noanswer”, “network_busy”, “near_end_disconnect”, “far_end_disconnect”, and “network_disconnect.”

For a blind transfer, the value is “success” if the transfer was successful. The other possible values are “busy”, “noanswer” and “network_busy.”

Outcomes not described in the values provided in the specification are declared “unknown.”

Cisco Extensions for <transfer>

Cisco VoiceXML supports specific Cisco telephony attributes listed in Table 1-3 in addition to supporting the standard VoiceXML attributes “dest” and “connecttimeout.”

Table 1-3 Cisco Telephony Attributes for <transfer>

Cisco Attribute	Description
cisco-username	RADIUS username. It can have a maximum string length of 32 characters. Longer strings are truncated. If cisco-username is missing, the calling number attribute is used.
cisco-ani	Calling (caller’s) number. The telephone number format is tel: <E.164 number>, where <E.164 number> can have a maximum string length of 32 digits. For example: cisco-ani= “tel: 5550134”
cisco-aniexpr	Calling number with subfield expr.
cisco-anitype	Calling number with subfield type.
cisco-aniplan	Calling number with subfield plan.
cisco-anipi	Calling number with subfield pi.
cisco-anisi	Calling number with subfield si.
cisco-carrierid-source	The source carrier ID for an outgoing call.
cisco-carrierid-target	The target carrier ID for an outgoing call.
cisco-desttype	Called number with subfield type.
cisco-destplan	Called number with subfield plan.
cisco-rdn	Redirecting number. The telephone number format is: “tel: <E.164 number>”. The <E.164 number> can have a maximum string length of 32 digits. For example: cisco-ani= “tel: 5550134”
cisco-rdnexpr	Redirecting number with subfield expr
cisco-rdntype	Redirecting number with subfield type
cisco-rdnplan	Redirecting number with subfield plan
cisco-rdnpi	Redirecting number with subfield pi
cisco-rdnsi	Redirecting number with subfield si

Table 1-3 Cisco Telephony Attributes for <transfer> (continued)

Cisco Attribute	Description
cisco-redirectreason	Reason for call redirection
cisco-mailtoaddress	Destination e-mail ID
long-#	Cisco Longpound Attribute

When the <transfer> element uses specific Cisco attributes to set signaling information in a transferred call, the voice gateway transfers the signaling information to remote equipment. If the gateway handling the incoming call contains a VoiceXML document, the signaling information is also available to the VoiceXML document as a set of session variables. Table 1-4 lists attributes of the <transfer> element mapped to specific Cisco VoiceXML variables for an incoming call.

Table 1-4 Cisco <transfer> Attributes Mapped to VoiceXML Session Variables

Cisco Attribute	VoiceXML Session Variable
cisco-ani	session.telephone.ani
cisco-aniexpr	session.telephone.ani
cisco-rdn	session.telephone.rdnis
cisco-rdnexpr	session.telephone.rdnis
cisco-redirectreason	session.telephone.redirect_reason
cisco-anitype	com.cisco.ani_noa
cisco-aniplan	com.cisco.ani_npi
cisco-anipi	com.cisco.ani_pi
cisco-anisi	com.cisco.ani_pi
cisco-rdntype	com.cisco.rgn_noa
cisco-rdnplan	com.cisco.rgn_npi
cisco-rdnpi	com.cisco.rgn_pi
cisco-rdnsi	com.cisco.rgn_si
cisco-desttype	com.cisco.dnis_noa
cisco-destplan	com.cisco.dnis_npi

The example below shows some of the specific Cisco attributes included in the <form> element:

```
<?xml version="1.0">
<vxml version="2.0">
    <form id = "transfer">
        <var name="mydur"/>
            <block>
                <audio src="http://myserver/welcome.au"/>
            </block>
            <transfer name="mycall" dest="tel: 18005550134"
                connecttimeout="30s" bridge="true"
                cisco-username="1234-5678-ABCD-WXYZ"
                cisco-aniexpr="'tel: ' + (4085260000 + 1234)"
                cisco-anitype="0" cisco-aniplan="1"
                cisco-anipi="0" cisco-anisi="0"
                cisco-rdn="tel: 4085261000"
```

```
                    cisco-rdntype="0" cisco-rdnplan="1"
                    cisco-rdnpi="0" cisco-rdnsi="0"
                    cisco-redirectreason="10"
                    cisco-desttype="0"
                    cisco-destplan="1">
                    <audio src="http://anyname.anycompany.com/jdoe/spacemusic.au"/>
                    <filled>
                        <!-- call should pass and
                                mycall$duration has new val -->
                        <assign name="mydur" expr="mycall$.duration = 99999"/>
                        <if cond="mycall$.duration == '11111111'">
                        </if>
                    </filled>
                </transfer>
            <block>
            <if cond="mycall$.duration">
            </if>
            <goto next="#transfer2"/>
            </block>
        </form>
        <form id = "transfer2">
            <block>
            <if cond="mycall$.duration">
            </if>
            </block>
        </form>
</vxml>
```

Definitions of Subfields For ANI, DNIS, and RDNIS

The subfields for ANI, DNIS, and RDNIS are defined below. For more details, see *ITU-T Q.931, ISDN User Network Interface Layer 3 Specification for Basic Call Control*.

Note

- Subfields must be present with their associated Q.931 information element (IE). If the subfields are present without their associated IE, an error event error.semantic is thrown.
- IE subfields do not propogate from the originating to the terminating gateway for calls over SIP.
- IEs and their subfields do not propogate over CAS because of limitations in R2 signaling.

Type of Number

3 bits for ANI, DNIS, and RDNIS

Field values	0 = Unknown 1 = International number 2 = National number 3 = Network specific number 4 = Subscriber number 6 = Abbreviated number

Numbering Plan Identification

4 bits for ANI, DNIS, and RDNIS

Field values	0 = Unknown 1 = ISDN/telephony numbering plan 3 = Data numbering plan 4 = Telex numbering plan 8 = National standard numbring plan 9 = Private numbering plan

Presentation Indicator

2 bits for ANI and RDNIS

Field values	0 = Unknown 1 = Presentation restricted 2 = Number not available because of interworking

Screening Indicator

2 bits for ANI and RDNIS

Field values	0 = User-provided, not screened 1 = User-provided, verified and passed 2 = User-provided, verified and failed 3 = Network provided

Reason for Redirection

4 bits for RDNIS

Field values	0 = Unknown 1 = Call forwarding busy or call DTE busy 2 = Call forwarding no reply 4 = Call deflection 9 = Called DTE out of order 10 = Call forwarding by the called DTE 15 = Call forwarding unconditional or systematic redirection

Precedence Rules for Setting Parameters

The tables shown below specify the precedence used in setting up parameters for the second call leg. The corresponding parameters may or may not be present in the <transfer> element or property. In each table, the precedence is arranged from the highest to the lowest as the condition in each row is checked for the parameters to be set for the second call leg.

In the tables below:

- cisco-ani represents any one attribute of the cisco-ani and cisco-aniexpr attribute pair. Only one of the two attributes can be present at a time.
- cisco-rdn represents any one attribute of the cisco-rdn and cisco-rdnexpr attribute pair. Only one of the two attributes can be present at a time.

RADIUS Username

If the RADIUS username is present, use its value cisco-username. If it is not present, use cisco-ani.

Calling Number

If the calling number is present, use these values:	The resultant values are:
cisco-ani with subfields	• cisco-ani with subfields. • cisco-anipi and cisco-anisi as a pair[1].
cisco-ani without subfields	cisco-ani subfield = incoming leg
subfields without cisco-ani	An error event, error.semantic is thrown.
None	Calling number = incoming leg subfield = incoming leg

1. cisco-anipi and cisco-anisi must be provided as a pair. Providing only one of these attributes may result in the loss of an attribute on the terminating gateway. No error events are thrown.

Called Number

If the called number is present, use these values:	The resultant values are:
dest with subfields	dest with subfields
dest without subfields	dest subfields = incoming leg
None	Call is not transferred.

Redirecting Number

If the redirecting number is present, use these values:	The resultant values are:
cisco-rdn with subfields	• cisco-rdn with subfields. • cisco-rdntype and cisco-rdnplan as a pair.[1] • cisco-rdnpi and cisco-rdnsi as a pair with cisco-rdntype and cisco-rdnplan.[2]
cisco-rdn without subfields	cisco-rdn subfield = incoming leg
subfields without cisco-rdn	An error event error.semantic is thrown.
None—With RDNIS information in the first call leg.	Redirecting number = incoming leg subfield = incoming leg
None—Without RDNIS information in the first call leg.	• Redirecting number= incoming leg called number. • cisco-rdntype = cisco-desttype • cisco-rdnplan = cisco-destplan • redirectreason = 10 (originating gateway only)

1. cisco-rdntype and cisco-rdnplan must be provided as a pair. Providing only one of these attributes may result in the loss of an attribute on the terminating gateway. No error events are thrown.

2. cisco-rdnpi and cisco-rdnsi must be provided as a pair. They must be provided only when cisco-rdntype and cisco-rdnplan are present. Failure to follow this guideline may result in a loss of attributes on the terminating gateway. No error events are thrown.

Control Flow and Scripting

This section describes mechanisms such as variables and events. that are used to control dialog flow.

Variables

VoiceXML 2.0 defines a set of standard JavaScript variables under the following groups:

- Supported Session Variables: Available to an entire VoiceXML session
- Application Variables: Available throughout a particular application within a session
- Dialog Variables: Available in a particular dialog within an application
- Event Variables: Available only in event handlers

Supported Session Variables

Cisco VoiceXML supports the standard session variables in the *VoiceXML Version 2.0* W3C Recommendation (March 16, 2004) and the extended session variables listed in Table 1-5.

Table 1-5 Cisco Extended Session Variables

Variable	Description
session.telephone.nas_port_id	Network universal gateway port number
session.telephone.rdnis	Redirect dialed number identification services
session.telephone.redirect_reason	Redirect reason
session.handoff_string	Handoff string from the previous application
session.mail.to	Holds the TO: header of the mail
session.mail.from	Holds the FROM: header of the mail
session.mail.subject	SUBJECT: header of the mail
session.mail.messageid	Message-Id: header of the mail
session.mail.headerinfo	All of the remaining SMTP headers, such as reply-to, if specified
com.cisco.fax.cng	CNG detection event

Table 1-5 Cisco Extended Session Variables (continued)

com.cisco.params	Used in a hybrid application, it is a session variable of the form com.cisco.params.xxxx, that is used by the VoiceXML session to access parameters passed from the Tcl script to the VoiceXML dialog. A Tcl array containing these parameters is passed from the calling Tcl application to the VoiceXML dialog. xxxx is the index in the array. See the **leg vxmldialog** verb in the *Tcl IVR Version 2.0 Programmer's Guide*. **Example:** Consider an array called paramarray, containing a parameter list, in the form of a string of variables, which is passed by the Tcl script to the VoiceXML dialog. When the VoiceXML dialog starts execution, the VoiceXML session can access com.cisco.params after it is initialized. • If paramarray contains: • paramarray(message) = "Hello World" • paramarray(totalusers) = 4 • paramarray(callingpartyname) = "ram://callingpartyname" com.cisco.params is initialized to contain: • com.cisco.params.message = "Hello World" • com.cisco.params.totalusers = 4 • com.cisco.params.callingpartyname = (an audio file object referring to "ram://callingpartyname")
com.cisco.ani_pi	Calling party number presentation indicator
com.cisco.ani_si	Calling party number screening indicator
com.cisco.ani_noa	Calling party number nature of address
com.cisco.ani_npi	Calling party number numbering plan indicator
com.cisco.carrierid.source	Defines the source ID for an incoming call.
com.cisco.carrierid.target	Defines the target carrier ID for an incoming call.
com.cisco.dnis_noa	Called party number nature of address
com.cisco.dnis_npi	Called party number numbering plan indicator
com.cisco.carrierid.source	Defines the source carrier ID for an incoming call.
com.cisco.carrierid.target	Defines the target carrier ID for an incoming call.
session.telephone.com.cisco.handoff.dnis	Returns the DNIS that is set by the inbound application in the <transfer> element (leg setup). The handoff attribute is always returned to the specific outbound application that receives the handoff call from the inbound application.

Table 1-5 Cisco Extended Session Variables (continued)

session.telephone.com.cisco.handoff.ani	Returns the ANI set by the inbound application in the <transfer> element (leg setup). The handoff attribute is always returned to the specific outbound application that receives the handoff call from the inbound application.
session.telephone.com.cisco.handoff.argstring	Replaces the existing variable "session.telephone.com.cisco.handoff_string"
session.telephone.com.cisco.handoff.proto_headers ['<attribute-name>']	Returns the value of the requested header that is handed off to an outbound application.
session.telephone.com.cisco.callid	Contains the call ID of the incoming call leg which can be reused for later calls. It is displayed in the output debug message. Only one call ID value exists for each VoiceXML document. When a call is connected during a <transfer> operation, the outbound call leg has a separate call ID which is not accessible to the VoiceXML script.
session.telephone.com.cisco.guid	Contains the globally unique ID of the call. It is unique between multiple gateways and is displayed in the output debug message.
session.connection.protocol.name	Identifies the connection protocol used.
session.connection.protocol.version	Identifies the version of the connection protocol used.
session.connection.aai	Provides application-to-application information passed during connection setup.
session.connection.redirect	This variable is an array representing the connection redirection paths. The first element is the original called number, the last element is the last redirected number. Each element of the array contains a URI, PI (presentation information), SI (screening information), and reason property. The reason property can be: • unknown • user busy • no reply • deflection during alerting • deflection immediate response • mobile subscriber not reachable
session.connection.originator	This variable directly references either the local or remote property.

For more information on signaling protocols applicable to the use of session variables, see the following documents:

- *E.164: Telephone Network and ISDN Operation, Numbering, Routing, and Mobile Service*

- *Q.931: Digital Subscriber Signaling System No.1 (DSS 1)— ISDN user-network interface layer 3 specification for basic call control*
- *H.323: Packet based Multimedia Communication Systems*
- *SIP: Session Initiation Protocol*

The session variables have the following values:

The session.telephone.ani_pi values are:

- Presentation allowed
- Presentation restricted
- Number lost because of networking
- Reserved value

The session.telephone.ani_si values are:

- USR provided unscreened
- USR provided screening passed
- USR provided screening failed
- Network provided

The session.telephone.ani_plan values are:

- Unknown
- ISDN telephony E.164
- Data X.121
- Telex F.69
- National
- Private
- Reserved value

The session.telephone.ani_type values are:

- Unknown
- International
- National
- Network specific
- Subscriber
- Abbreviated
- Reserved value

The session.telephone.rdnis_pi values are:

- Presentation allowed
- Presentation restricted
- Number lost because of networking
- Reserved value

The session.telephone.rdnis_si values are:

- USR provided screening passed

- USR provided screening failed
- Network provided

The session.telephone.rdnis_type values are:

- Unknown
- International
- National
- Network specific
- Subscriber
- Abbreviated
- Reserved value

The session.telephone.rdnis_plan values are:

- Unknown
- ISDN telephony E.164
- Data X.121
- Telex F.69
- National
- Private
- Reserved value

The session.telephone.dnis_type values are:

- Unknown
- International
- National
- Network specific
- Subscriber
- Abbreviated
- Reserved value

The session.telephone.dnis_plan values are:

- Unknown
- ISDN telephony E.164
- Data X.121
- Telex F.69
- National
- Private
- Reserved value

The session.telephone.redirect_reason values are a string set by the signaling protocol (redirect_reason).

The session.telephone.handoff_string values are a string set by the application (handoff_string).

The session.telephone.redirect_count values are a number. Values retrieved between 0 and 7.

The session.telephone.nas_port_id values are for the following interface types:

- FXO—FXO, followed by the port name
- FXS—FXS, followed by the port name
- E&M—ENM, followed by the port name
- ISDN—ISDN, followed by the port name

For example, a value for an ISDN interface is ISDN 7/1:D.

The session.telephone.iidigits values are:

- Unknown
- ISDN telephony E.164
- Data X.121
- Telex F.69
- National
- Private
- Reserved value

Application Variables

Cisco VoiceXML supports the standard application variable application.lastresult$. For more information on this variable, see the *VoiceXML 2.1* W3C Candidate Recommendation (June 13, 2005). In addition to the standard variable, Cisco IOS Release 12.2(11)T introduces specific Cisco application variables, listed in Table 1-6.

A specific Cisco application variable, application.lastprompt$ holds information about the last prompt that was played through the read-only variables listed in Table 1-6.

Table 1-6 Specific Cisco Application Variables

Variable	Description
application.com.cisco.lastprompt$.duration	• Playout duration of the last prompt before it is interrupted by a bargein. It uses the application scoped Cisco specific variable com.cisco.lastprompt$ to keep the information of the prompt played last. • The duration, in milliseconds, is from the start of the entire set of audio files being played within the <prompt> element. **Note** If the prompt is played in its entirety without a bargein, the duration of playout should be equal to the duration of playout for the entire prompt. • For a prompt that contains TTS or RTSP audio files, the duration of playout is unknown and the value of the variable is -1. • In Cisco IOS Release 15.0(1)M and later releases, for a prompt that contains RTSP audio files, the duration of playout is determined by the value of the *cisco-maxtime* attribute of the <prompt> element. If *cisco-maxtime* is zero or has no value set, the RTSP stream is played indefinitely.

Table 1-6 Specific Cisco Application Variables

Variable	Description
application.com.cisco.lastprompt$.lastrate	• Playout rate of the last prompt. • The rate is an absolute value in the range from +4 to -4. • If rate control is not supported for a prompt, the value of the variable is 0. • The value of this variable is set to undefined when an application root document is loaded.

Dialog Variables

In VoiceXML 2.0, all dialog variables are represented as shadow variables.

Cisco VoiceXML supports all standard shadow variables including:

- name$.duration: Duration of the recording in milliseconds
- name$.size: Size of the recording in bytes
- name$.termchar: If *dtmfterm* is true, this shadow variable holds the pressed key.
- name$.maxtime: Boolean which is true if the recording was terminated because the maxtime duration was reached
- name$.recording: Reference to the recording, or undefined if no audio is collected
- name$.recordingsize: Size of the recording in bytes, or undefined if no audio is collected
- name$.recordingduration: Duration of the recording in milliseconds, or undefined if no audio is collected

Event Variables

Cisco VoiceXML does not support standard event variables.

Event Handling

This section discusses:

- The Fax Event Handler— A catch event handler to process the fax detection event.
- Events and Errors— A list of events thrown by the platform.

Fax Event Handler

The VoiceXML document has to define a catch event handler to process the fax detection event. This definition should normally be done in the application root document so it can remain active as other VoiceXML documents are loaded and executed during the current session.

The VoiceXML specification does not allow transfer elements in VoiceXML catch blocks. To get around this, use a goto tag with an associated form id block, where the transfer element resides as shown in the following example:

```
<catch event="com.cisco.fax.cng">
       <!-Can't have a transfer tag inside a catch handler…..-->
```

```
            <goto next="#transferToFax"/>
    </catch>
….
    ….
<form id="transferToFax">
        <transfer dest=" fax://4085550134" bridge=false cisco-mailtoaddress =
        "'email-id'">
    </form>
```

Events and Errors

Cisco VoiceXML supports the events and errors listed in Table 1-7.

Table 1-7 Events and Errors Supported by Cisco VoiceXML

Error Event	Description
error.unsupported	If you use regex grammar with a version 2.0 VoiceXML document or, if you specify DTMF grammar without specifying the DTMF option and do not configure the ASR server, then the interpreter throws an error.unsupported event. When a platform does not support a specific sequence of audio URI or codecs, or prompt elements, the interpreter throws an error.unsupported event. **Note** A VoiceXML document throws an error.unsupported event even if you have pure DTMF grammar.
error.condition.baddestination	When an invalid number is used for a transfer element, the interpreter throws an error.condition.baddestination event.
error.badfetch	When a media server cannot reach or fetch a URI that is specified in the speech markup or in the grammar, the interpreter throws an error.badfetch event.
error.unsupported.language	If the media server is configured but does not support the requested language, then the platform throws an error.unsupported.language event.
error.unsupported.format	If the media server is configured but cannot process the grammar or speech format that is used by the document, an error.unsupported.format event is generated.

Table 1-7 Events and Errors Supported by Cisco VoiceXML (continued)

Error Event	Description
disconnect.com.cisco.handoff	If the destination application does not return the call leg but disconnects it, the VoiceXML interpreter throws a disconnect.com.cisco.handoff event to the VoiceXML application and the call is handed off to another application using the <object> element. If the return flag is FALSE, indicating a nonreturnable handoff, then the VoiceXML interpreter throws a disconnect.com.cisco.handoff event to the VoiceXML application.
error.com.cisco.handoff.failure	If there is a failure in handing off an application; for example if the application is not preconfigured or built-in, or if it cannot be loaded, then the interpreter throws an error.com.cisco.handoff.failure event to the VoiceXML application.
error.unsupported.object	If the platform specific function com.cisco.callhandoff is not supported, the interpreter throws an error.unsupported.object event to the VoiceXML application. **Note** In VoiceXML 2.0 and later versions, this event is replaced by error.unsupported.objectname.
error.com.cisco.aaa.authenticate.failure	This error event is generated if there are server errors, for example, when the RADIUS server is unreachable.
error.com.cisco.aaa.authorize.failure	This error event is generated if there are server errors, for example, when the RADIUS server is unreachable.
error.noauthorization	This error event is generated when the application tries to perform an operation that is not authorized by the platform.
error.noresource	This error event is generated when a run-time error occurred because a requested platform resource was not available during execution and when playing prerecorded files from the router fails. In versions earlier than VoiceXML 2.0, the equivalent events are error.com.cisco.media.resource.unavailable, error.com.cisco.resource.failure.asr, error.com.cisco.resource.failure.tts, and error.badfetch (for failure playing prerecorded files from the router). These events are supported for backward compatibility.

Table 1-7 Events and Errors Supported by Cisco VoiceXML (continued)

Error Event	Description
error.unsupported.objectname	This error event is generated when the platform does not support the object. **Note** In versions earlier than VoiceXML 2.0, the equivalent event is error.unsupported.object, which is supported for backward compatibility.
connection.disconnect.hangup	This event is generated when the user hangs up. In versions earlier than VoiceXML 2.0, the equivalent event is telephone.disconnect.hangup, which is supported for backward compatibility.
connection.disconnect.transfer	This event is generated when the user is transferred unconditionally to another line and will not return. In versions earlier than VoiceXML 2.0, the equivalent event is telephone.disconnect.transfer, which is supported for backward compatibility.

Cisco VoiceXML provides a set of default event handlers if no appropriate event handler is specified in the VoiceXML document or in the root document; these are summarized in Table 1-8.

Note For event types using default event handlers, no audio is provided.

Table 1-8 Event Types Using Default Event Handlers

Event Type	Action
cancel	Does not reprompt
error	Exits interpreter
error.noauthorization	Exits interpreter
exit	Exits interpreter
help	Reprompts
noinput	Reprompts
nomatch	Reprompts
telephone.disconnect	Exits interpreter
All others	Exits interpreter

Note If the behavior of a catch handler is specified as an error event in a VoiceXML document, the VoiceXML interpreter does not exit after executing the catch handler; it allows the developer to use another VoiceXML document. For example, you can play a different prompt on a different server instead of exiting the interpreter.

JavaScript Support

Cisco VoiceXML supports ECMAScript Specification 3.0 (*Standard ECMA-262, ECMAScript Language Specification*, 3rd edition, August 1998, available at http://www.ecma.ch/), approximately equivalent to JavaScript 1.5 with the following exceptions:

- Built-in type: buffer
- The following function properties of the math object: acos(), asin(), atan(), atan2(), cos(), exp(), log(), sin(), sqrt(), tan(), toExponential(), toFixed(), toPrecision().

Use of these unsupported features causes the VoiceXML interpreter to throw an error.semantic event.

Cisco VoiceXML supports floating point operations with a limited number range:

- MIN_VALUE = 5e-32
- MAX_VALUE = 1.7976931348623157e+31

The size limit for JavaScript scripts is 65K in Cisco IOS Release 12.4(12), Cisco IOS Release 12.4(12)T, and later releases. Javascript scripts consume memory and CPU resources. To avoid performance problems, make sure JavaScript scripts contain only the minimum required code.

Environment and Resources

This section consists of:

Resource Fetching and Caching

The Cisco implementation of VoiceXML 2.1 supports resource fetching and caching. For more information, see the *VoiceXML 2.1* W3C Candidate Recommendation (June 13, 2005).

Fetching

The resources fetched by the VoiceXML interpreter context are:

- VoiceXML documents
- Prerecorded audio files
- JavaScript scripts
- Speech and DTMF grammars

Cisco VoiceXML supports the following attributes that govern fetching of content associated with a URI:

- The fetchtimeout attribute specifies the waiting time for a fetch.
- The fetchhint attribute defines the time when the interpreter retrieves content from the server.
- The fetchaudio attribute specifies the URI of the audio to be played while the document is being fetched.

Note Cisco VoiceXML does not support the fetchhint properties.

The protocols supported for fetching are:

- Hypertext Transfer Protocol (HTTP)
- Real-Time Streaming Protocol (RTSP)
- Trivial File Transfer Protocol (TFTP)

Caching

Cisco VoiceXML does not have its own caching mechanism. It relies on the HTTP protocol to provide caching.

Note Only the HTTP protocol has caching capabilities.

Cisco VoiceXML does not allow the user to control caching.

Property

Cisco <property> extensions are listed in Table 1-9:

Table 1-9 Cisco < property> Extensions

Property	Description
com.cisco.tts-server	Allows the document to specify an external media server for text-to-speech operations. The media server is specified in the form of an URI, and is used in all consecutive ASR operations until the next media server is specified. An external media server specified by a property in the script takes precedence over being specified by a command through the CLI. It can be defined for: • An entire application or document at the <vxml> level, • A specific dialog at the form or menu level, or • A specific form item. The media server's URI can be formatted for Media Resource Control Protocol version 1 (MRCP v1) which uses Real Time Streaming Protocol (RTSP) or MRCP v2, which uses Session Initiation Protocol (SIP), for example: <property name="com.cisco.tts-server" value="rtsp://tts-server1/synthesizer" /> <property name="com.cisco.tts -server" value="sip:mresources@mediaserver.com" /> There are two ways to specify an external media server for TTS and ASR operations: • **ivr tts-server** and **ivr asr-server** commands—Media server sessions are created for each call to IVR applications, regardless of whether an application needs to talk to the media server. • com.cisco.tt-server and com.cisco.asr-server <property> extensions—Media server sessions are created for each call to that application. If only a small number of applications require TTS/ASR media sessions, you should use the <property> extensions within those applications to define the external media server URL in the VoiceXML script. For information on identifying TTS or ASR servers through the **ivr tts-server** and **ivr asr-server** commands, see the *Cisco IOS Tcl IVR and VoiceXML Application Guide*.
com.cisco.asr-server	Allows a document to specify an external media server for automatic speech recognition operations. The media server is specified in the form of an URI, and is used in all consecutive ASR operations until the next media server is specified. An external media server specified by a property in the script takes precedence over being specified by a command through the CLI. The media server's URI can be formatted for Media Resource Control Protocol version 1 (MRCP v1) which uses RTSP or MRCP v2, which uses SIP, for example: <property name="com.cisco.asr-server" value="rtsp://asr-server1/synthesizer" /> <property name="com.cisco.asr -server" value="sip:mresources@mediaserver.com" />
com.cisco.asr-builtin-grammar	Allows the document to specify a base URI for the external media server for ASR operations.

Table 1-9 Cisco < property> Extensions (continued)

Property	Description
com.cisco.media-logging-id	• Allows the document to specify a unique ID for that specific instance of the application. This ID is passed on to the external media server to allow it to associate session logs generated for TTS and ASR operations with that ID. • If this property is not set, the Cisco voice gateway automatically generates this ID for that specific call or instance of the VoiceXML interpreter and uses it for the length of the call or session. • If a document chooses to override this ID with a new one, the new ID applies to the rest of the session or until the time when another document chooses to set a new ID.
com.cisco.autoflush	Prevents default flushing of the typeahead buffer by setting the property to FALSE.
com.cisco.record_finalsilence	Specifies the interval of silence that indicates the end of speech.
com.cisco.record_maxtime	Specifies the maximum duration of the recording.
com.cisco.record_type	Specifies the MIME format of the resulting recording.
com.cisco.mta.send.mdn_hostname	Specifies the hostname of the receiving e-mail address for message delivery notification (MDN).
com.cisco.mta.send.mdn_username	Specifies the username of the receiving e-mail address for message delivery notification.
com.cisco.mta.send.dsn_delay	Specifies a delay in the message delivery.
com.cisco.mta.send.dsn_failure	Specifies a failure in the message delivery.
com.cisco.mta.send.dsn_success	Specifies successful delivery of the message.
com.cisco.mta.send.from_hostname	Specifies the hostname from the originating address of the e-mail.
com.cisco.mta.send.from_username	Specifies the username from the originating address of the e-mail.
com.cisco.mta.send.origin_prefix	Specifies prefix header from the originating e-mail.
com.cisco.mta.send.server	Specifies the destination server.
com.cisco.mta.send.subject	Defines the text that appears in the “subject” field of the e-mail.
bargeintype	Specifies the type of bargein: speech or hotword.
markname	The name of the mark last executed by the SSML processor before bargein or the end of audio playback occurred.
marktime	The number of milliseconds that elapsed since the last mark was executed by the SSML processor until bargein or the end of audio playback occurred.
recordutterance	To enable recording during recognition, set the value of the recordutterance property to true. Recording during recognition is not supported for the <transfer> and <record> elements. **Note** In VoiceXML 2.1 and later versions, the recordutterance property is not supported for the <transfer> and <record> elements.

In Cisco IOS Release 12.4(11)T and later releases, an error event error.semantic is thrown if a property value is found to be illegal.

Default Values and Ranges

This section describes values for the VoiceXML timeouts: fetchtimeout, timeout, interdigittimeout, connecttimeout, and maxtime. For all of these, a negative value is invalid and 0 means default.

The maximum supported value for each timeout described in this section is 2147483 seconds. The duration string after fetchtimeout could be arbitrary. When a timeout is converted into a real value, it could assume only a valid positive integer value, ranging from 0 to 2147483647 milliseconds, that is, 596523 hours or about 68 years.

The VoiceXML timeouts are listed in Table 1-10. For more information, see *VoiceXML 2.1* W3C Candidate Recommendation (June 13, 2005). For information on configuring default values through the CLI, see the *Cisco IOS Tcl and VoiceXML Application Guide* for your Cisco IOS release.

Table 1-10 VoiceXML Timeouts

VoiceXML Timeout	Description
fetchtimeout	The fetch timeout for loading a file. For this timeout, 0 or the default means ten seconds. The default can be reset through the CLI.
timeout	The initial timeout for collecting digits. For this timeout, 0 or the default means ten seconds.
incompletetimeout	The required length of silence following user speech after which a recognizer finalizes a result. **Note** The behavior of the completetimeout property is not supported. However, the value of the completetimeout property is parsed. The larger of the completetimeout and incompletetimeout values is used as the value of incompletetimeout.
interdigittimeout	The maximum interval between digits. For this timeout, 0 or the default means ten seconds. If interdigittimeout is not specified, the port configuration value is used.
connecttimeout	The timeout for connecting a call. For this timeout, 0 or the default can be a maximum of sixty eight years.
maxtime	The maximum call duration. For this timeout, 0 or the default can be a maximum value of sixty eight years.

Call Handoff

To understand call handoff, it is important to understand the concept of an application instance. In the Cisco IOS interactive voice response (IVR) infrastructure, an application instance is an entity that executes the application code, and receives, creates, and manages one or more call legs together to setup a call or to deliver a service to a user. The application instance owns and controls these call legs, and

receives and manages all events associated with these call legs. Most applications will use a single application instance to deliver the services of a single call. A stand alone VoiceXML session acts as an application instance.

The term *call handoff* is used to describe the act of transferring complete control of a call leg from one application instance to another. When a call leg is handed off, all future events associated with that call leg are received and handled by the target application instance. There are different types of call handoff, and they occur depending on whether the call leg needs to return to the application instance that is the source of the handoff operation. A normal call handoff is similar to a goto statement with no automatic memory of a return address where the target application instance cannot return the call leg back to the source. During the handoff of a call leg, any legs conferenced to that call leg are also handed off. During a handoff or a handoff return operation, an application instance can pass parameters as argument strings.

The call handoff functionality allows a developer to write applications that may interact with each other for various reasons; for example, to use or leverage functionality in existing applications, or to modularize a larger application into smaller application segments and use the handoff mechanism to coordinate and communicate between them. During call handoff, only one application instance controls the call leg and receives events. Hybrid scripting is a recommended alternative for developers implementing applications that use VoiceXML and Tcl IVR 2.0. For more information on hybrid scripting, see the "Hybrid Applications" section on page 1-115.

The call handoff functionality in Cisco VoiceXML is similar to the call handoff initiated by the **handoff appl** and **handoff callappl** verbs in Tcl IVR 2.0. For more information on the Tcl IVR 2.0 verbs, see the *Tcl IVR Version 2.0 Programmer's Guide*.

Call Handoff in Cisco VoiceXML

Call handoff in Cisco VoiceXML is invoked through the <object> element as follows:

```
<object
        name= "ResultVariableName"
        classid= "builtin://com.cisco.callhandoff"
    <param name= "return" expr="<Boolean value>"/>
    <param name= "app-uri" expr="<URI>"/>
    <param name= "arg-string" expr="a string"/>
</object>
```

Call handoff can take place between any combination of VoiceXML and Tcl IVR applications, and is of two types:

- Handoff application— A one way handoff where the invoking VoiceXML session hands off the call leg to the destination application without the call leg returning to it. To enable this one way handoff, the parameter "return" is set to false.

  ```
  <param name= "return" expr="<false>"/>
  ```

  Before the handoff takes place, both applications must be configured on the gateway. After the handoff is successful, a handoff event telephone.disconnect.com.cisco.callhandoff is thrown at the invoking session, and the VoiceXML document continues executing without a call leg, similar to a <transfer> with bridging set to false.

- Handoff call application— In a handoff call application, the invoking VoiceXML session hands off the specified call leg to the destination application, and waits for the call leg to be returned to it. To allow the call leg to return, the parameter name "return" is set to true.

  ```
  <param name= "return" expr="<true>"/>
  ```

  If the return flag is true, the VoiceXML interpreter execution is blocked until the handoff is complete. The interpreter waits for the called application to return with the call leg. During this handoff, no grammars can be active on the leg because the VoiceXML application instance has no

control on the call leg. If the destination application does not return the leg but disconnects it, the VoiceXML interpreter is released from being blocked, throws a telephone.disconnect.com.cisco.handoff event to the VoiceXML application, and continues executing without a call leg, similar to a <transfer> with bridging set to false. If the target application returns the call leg, the field variable is filled with the handoff completion status and the form continues to be processed. If the target application returns the call leg with additional parameters, they can be accessed through the field variable of the <object> element and the field variable is filled with a complex object.

- ResultVariableName.status is filled with the handoff status value of true for a successful handoff and return operation, and false for a failure. ResultVariableName.argstring is filled with the argument string returned by the target application. arg-string is an optional parameter name that is used to pass parameters to the target application during a handoff. The target application instance has access to the argument string when it starts up. A VoiceXML application can access the handoff arg-string that is passed from the source of a handoff operation as a session variable session.telephone.handoff_string.
- A return value of false indicates a nonreturnable handoff. If no return value is written into the script, the default value is false.

Table 1-11 lists the attributes and parameters for the call handoff script shown above.

Table 1-11 Attributes and Parameters for Call Handoff

Attribute or Parameter	Description
name (attribute)	A field item variable name that contains the results of the handoff operation. It contains the subelement ResultVariableName.argstring in the form of a string that is returned to the initiating application.
classid (attribute)	An attribute that specifies the platform object to be invoked to initiate the handoff operation. It also invokes the RADIUS authorization command.
return (parameter)	• An optional parameter in the form of a boolean value of true or false. • If the value is true, the initiating application waits for the call leg to be returned to it by the destination application. • If the value is false, the handoff is nonreturnable. • If the value is not specified, the default is false, where the handoff is nonreturnable.
app-uri (parameter)	A mandatory parameter, it is a URI that specifies the destination application. The URI is of the form builtin://appname, where appname is the name of the destination application.
arg-string (parameter)	An optional parameter that specifies a string or list of strings to be passed to the destination application.

Events and Errors in Call Handoff

- telephone.disconnect.com.cisco.callhandoff
  - This event is thrown in a handoff application indicating a nonreturnable handoff.
  - This event is thrown in a handoff call application, if the destination application disconnects the call leg instead of returning it.
- error.com.cisco.callhandoff.failure

  This error event is thrown in both types of call handoff if there is an error in handing off the call leg to the destination application. For example, an error occurs if the destination application is not configured or built into the voice gateway.
- error.unsupported.object

  This error event is thrown if the specific platform variable com.cisco.callhandoff is not supported.

Example

The call handoff example shown here has a return value of true.

```
<var name="appName" expr="'builtin://coapp'"/>
!coapp is the name of the application configured on the gateway using the call application
voice command.
   <object name="myhandoff" classid="builtin://com.cisco.callhandoff">
        <param name="return" expr="true"/>
!For a return value of false, expr="false".
        <param name="app-uri" expr="appName"/>
        <param name="arg-string" expr="'arg 1'" />
   </object>
```

Authentication and Authorization

Cisco VoiceXML implements the **aaa authorize** and **aaa authenticate** commands through the <object> element. The implementation is very similar to the use of **aaa authorize** and **aaa authenticate** in Tcl IVR 2.0, where authentication and authorization requests are sent to a RADIUS (or TACACS) server.

Some of the main differences between authentication and authorization are:

- Authentication validates the identity of the user.
- Authorization authorizes what the user can do.
- Authentication is valid without authorization.
- Authorization is not valid without authentication.

RADIUS is a distributed client-server system protocol that secures networks against unauthorized access. In Cisco's implementation of RADIUS, clients run on Cisco routers, access servers, and gateways, and send authentication requests to a central RADIUS server that contains user authentication and network service access information.

For information on how to configure Cisco IOS security features on your network device, see the *Cisco IOS Security Configuration Guide, Release 12.4T*, and the *Cisco IOS Security Command Reference Guide, Release 12.4T*.

Authentication

In Cisco VoiceXML:

- An authentication request containing the account number and password is first sent to the RADIUS server through the <object> element. The attribute classid specifies authentication through the Cisco extension com.cisco.aaa.authenticate and is of the form:

  ```
  classid="builtin://com.cisco.aaa.authenticate"
  ```

- Vendor specific attributes (VSAs) may also be sent by the application to the RADIUS server. These VSAs are represented as parameters in the script with a valuetype “com.cisco.datatype.list.” For a list of VSAs and how to use them, see the *Vendor Specific Attributes Voice Implementation Guide*.
- The RADIUS server sends the results of the authentication and applicable VSAs back to the voice gateway in the form of a field item variable name:

  ```
  name="ResultVariableName"
  ```

  It contains the authentication result (represented as pass or fail) and any vendor specific attributes that are sent by the billing server as part of the authentication response.
- To choose a specific RADIUS server group for authentication, use the server-tag to identify that server group. The name of that group corresponds to the server group name configured through the CLI.
- The VoiceXML interpreter is blocked until authentication is complete.

Table 1-12 summarizes the attributes and parameters of the authentication object.

The authentication object is accessed as follows:

```
<object
        name="ResultVariableName"
        classid="builtin://com.cisco.aaa.authenticate"
    <param name="account" expr="user_account_num"/>
    <param name="password" expr="user_passwd"/>
    <param name="server-tag" expr="server_tag"/>
    <param name="key1" expr="value1" valuetype="com.cisco.datatype.list"/>
    <param name="key2" expr="value2" valuetype="com.cisco.datatype.list"/>
     .
     .
     <param name="keyN" expr="valueN" valuetype="com.cisco.datatype.list"/>
</object>
```

Table 1-12 Attribute and Parameter Descriptions for the Authentication Object

Attribute or Parameter	Description
name (attribute)	It is a field item variable name that contains two subobjects: • *ResultVariableName.result*— It contains the results of the authentication operation as a pass or fail. • *ResultVariableName.attributes*— It contains vendor specific attributes (VSAs) that are returned by the RADIUS server as part of the authentication response. The *attributes* object is a two dimensional array indexed by the VSA name and an instance number. For example, if the RADIUS server returns the VSA "h323-credit-amount" with a value of 282.25, the object [h323-credit-amount][0] contains the value 282.25. • If the VoiceXML script accesses VSAs that are not returned in the authentication operation, the variable returns undefined. For more information on VSAs, see the *RADIUS Vendor Specific Attributes Voice Implementation Guide*.
classid (attribute)	It specifies the authentication operation through the Cisco extension *com.cisco.aaa.authenticate*.
account (parameter)	A mandatory parameter that specifies the account number for authentication.
password (parameter)	A mandatory parameter that specifies the password for the account.
server-tag (parameter)	• An optional parameter that specifies the RADIUS server group to be used for authentication. • The name of that group corresponds to the server group name configured through the CLI.
key 1 to key N: VSA name (parameter)	• These optional parameters are VSAs that are sent to the RADIUS server as a part of the authentication request. These VSA parameters are recognized as their valuetype attribute com.cisco.datatype.list. These VSAs are sent in the authentication and authorization messages. For a list of VSAs and how to use them, see the *RADIUS Vendor Specific Attributes Voice Implementation Guide*. • A VSA can occur multiple times if there is a need to send multiple instances of the same VSA to the RADIUS server.

Events and Errors in Authentication

- error.com.cisco.aaa.authenticate.failure

  This error event is thrown if the voice gateway does not connect with the RADIUS server and authentication fails. This error event is also thrown if there is a server error after the voice gateway connects with the RADIUS server.

- error.unsupported.object

  This error event is thrown if the specific platform function com.cisco.authentication is not supported.

Authorization

In Cisco VoiceXML:

- An authorization request containing the account number and password is first sent to the RADIUS server through the <object> element. The attribute classid specifies authorization through the specific Cisco extension com.cisco.aaa.authorize. It is of the form:

  ```
  classid="builtin://com.cisco.aaa.authorize"
  ```

- The ANI and the DNIS of the call being authorized is included in the authorization command sent to the RADIUS server.
- Vendor specific attributes (VSAs) may also be sent by the application to the RADIUS server. These VSAs are represented as parameters in the script with a valuetype “com.cisco.datatype.list.” For a list of VSAs , see the *Vendor Specific Attributes Voice Implementation Guide*.
- The RADIUS server sends the results of the authorization back to the voice gateway in the form of a field item variable name:

  ```
  name= "ResultVariableName"
  ```

  It contains the authorization result (represented as pass or fail) and any vendor specific attributes that are sent by the billing server as part of the authorization response.
- To choose a specific RADIUS server group for authorization, use the server-tag to identify that server group. The name of that group corresponds to the server group name configured through the CLI.
- The VoiceXML interpreter is blocked until authorization is complete.

The authorization object is accessed as follows:

```
<object>
        name="ResultVariableName"
        classid="builtin://com.cisco.aaa.authorize"
    <param name="account" expr="user_account_num"/>
    <param name="password" expr="user_passwd"/>
    <param name="ani" expr="calling_num"/>
    <param name="dnis" expr="called_num"/>
    <param name="server-tag" expr="server_tag"/>
    <param name="key1" expr="value1" valuetype="com.cisco.datatype.list"/>
    <param name="key2" expr="value2" valuetype="com.cisco.datatype.list/>
     .
     .
     <param name="keyN" expr="valueN" valuetype= "com.cisco.datatype.list"/>
</object>
```

Table 1-13 summarizes the attributes and parameters of the authorization object.

Table 1-13 Attribute and Parameter Descriptions for the Authorization Object

Attribute or Parameter	Description
name (attribute)	It is a field item variable name that contains two subobjects: • ResultVariableName.result— It returns the results of the authorization operation as a pass or fail. • ResultVariableName.attributes— It contains vendor specific attributes (VSAs) that are sent by the billing server as part of the authorization response. For example, if the RADIUS server returns the VSA "h323-credit-amount" with a value of 282.25, the object [h323-credit-amount][0] contains the value 282.25. • If the VoiceXML script accesses VSAs that are not returned in the authorization operation, the variable returns undefined. For more information on VSAs, see the *RADIUS Vendor Specific Attributes Voice Implementation Guide*.
classid (attribute)	It specifies the authorization operation through the specific Cisco extension com.cisco.aaa.authorize.
account (parameter)	A mandatory parameter that specifies the account number for authorization.
password (parameter)	A mandatory parameter that specifies the password for the account.
ani (parameter)	A mandatory parameter that specifies the ANI of the call being authorized.
dnis (parameter)	A mandatory parameter that specifies the DNIS of the call being authorized.
server-tag (parameter)	• An optional parameter that specifies the RADIUS server group to be used for authorization. • The name of that group corresponds to the server group name configured through the CLI.
key 1 to key N: VSA name (parameter)	• These optional parameters are VSAs that are sent to the RADIUS server as a part of the authorization request. They have a valuetype of "com.cisco.datatype.list." For a list of VSAs, see the *RADIUS Vendor Specific Attributes Voice Implementation Guide*. • A VSA can occur multiple times if there is a need to send multiple instances of the same VSA to the RADIUS server.

Events and Errors in Authorization

- error.com.cisco.aaa.authorize.failure

  This error event is thrown if the voice gateway does not connect with the RADIUS server and authorization fails. This error event is also thrown if there is a server error after the voice gateway connects with the RADIUS server.

- error.unsupported.object

  This error event is thrown if the specific platform function com.cisco.aaa.authorization is not supported.

SIP and H.323 Support

Cisco IOS Release 12.3(T) allows VoiceXML applications to pass and receive URI and SIP header information in call setup messages, specify headers that are sent in SIP INVITE or H.323 setup messages, and initiate blind call transfers.

SIP and TEL URL Support

This feature allows VoiceXML applications to place a call to a destination address containing a URI, and to pass and receive URI and SIP header information in a call setup message for inbound and outbound calls. For Cisco IOS configuration tasks related to this feature, see the following documents:

- *Cisco IOS Tcl and VoiceXML Application Guide*
- *SIP Header/URL Support and Subscribe/Notify for External Triggers*

Limitations

Limitations of SIP and TEL URL Support in Cisco VoiceXML are:

- The destination URI string length cannot exceed 1024 bytes. The VoiceXML application throws an error if the document places a call to a URI whose string length exceeds 1024 bytes.
- For a call going out on an H.323 network, the destination URI has a maximum string length of 512 bytes.
- Headers in abbreviated formats are always converted to their full formats before being stored and are not received by the VoiceXML application. For example, an application requesting the session variable session.com.cisco.proto_headers[‘t’] does not receive the value of the “To:” header field.
- Restricted SIP headers for outbound calls: The following SIP headers cannot be overwritten:
  - Call-ID
  - Supported
  - Require
  - Min-SE
  - Session-Expires
  - Max-Forward
  - CSeq

  Excluding this list of headers, a VoiceXML application is allowed to pass any header, including extended and nonstandard headers.
- Limitations for passing protocol headers are:
  - 20 KB is the maximum memory allocation for a call passing headers.
  - Each header avpair has a maximum limit of 256 characters. The application throws an error if a VoiceXML document passes a header avpair exceeding 256 characters, or if the memory exceeds 20 KB bytes.

Reserved Characters

Character	Escape Sequence
&	&amp;
<	&lt;
>	&gt
‘	&apos;
“	&quot;

For example, *sip:joe@big.com?Subject=Hello&ampPriority=Urgent* must be rewritten as:

sip:joe@big.com?Subject=Hello&Priority=Urgent.

If the URL in a header contains a reserved character from this list, the header must be rewritten with the equivalent escape sequence.

Passing Headers in Voice Messages

In Cisco IOS Release 12.3 T, Cisco VoiceXML allows a VoiceXML document to specify headers that are sent in the SIP INVITE or H.323 setup message. Voice applications can use headers in SIP *invite* messages to pass information about a call to an application on another server. For example, an account number can be passed in a SIP header if the caller entered an account number and the application transfers the call to another application on another platform. A typical scenario is an airline application where a call comes into an airline application, and then needs to be transferred to a hotel reservation application. If the voice browser cannot load VoiceXML documents directly from the HTTP servers of the airline and the hotel, the voice application can use headers in SIP messages to transfer call information. That call can then be transferred to a hotel reservation application that is being hosted by a different service provider.

The script writer enters the header in the destination URI after the ? as part of the SIP URL, or after the ; as part of the TEL:URL. Headers embedded in SIP URLs are separated by *&amp*. Headers embedded in TEL URLs are separated by a semicolon (;).

Note The URL in TEL:URL can only be an E.164 number. For example, tel:5550123 or tel:+1-408-555-1023.

Example

In this SIP:URL, “sip:joe@big.com?Subject=Car Rental&ampPriority=urgent&ampAccount=1234567890” the header *Subject=Car Rental&ampPriority=urgent&ampAccount=1234567890* is embedded into the SIP URL after the question mark ? .

In this TEL: URL, “tel:5550123;Subject=Car Rental;Priority=urgent;Account=1234567890” the header *Subject=Car Rental;Priority=urgent;Account=1234567890* is embedded in the TEL URL after the semicolon ; .

Headers in Outbound Calls

Cisco VoiceXML allows access to headers in only the following messages:

- SIP INVITE
- Subscribe
- Notify
- H.323 Setup

Passing Headers from <transfer> to SIP INVITE

A VoiceXML document appends the header value-pairs as part of the *destexpr* (or *dest*) attribute of the <transfer> element.

In the example shown here, the subject, priority, and X-ReferenceNumber headers are embedded in the *destexpr* URI and are sent out in the SIP INVITE message.

```
<transfer name="mycall"
    destexpr="'sip:joe@big.com?Subject=Hotel Reservation&ampPriority=urgent&amp
    X-ReferenceNumber=1234567890'"
    ...
</transfer>
```

The SIP INVITE message that is transferred out is shown below. The headers that are passed from the VoiceXML application are Subject: Hotel Reservation, Priority: urgent, and X-ReferenceNumber: 1234567890.

```
INVITE sip:joe@1.100.7.32 SIP/2.0
Via: SIP/2.0/UDP 192.168.6.121:5060
From: 4085550134@192.168.6.121
To: joe@big.com
Call-ID: c2943000-e0563-2a1ce-2e323931@192.168.6.21
Subject: Hotel Reservation
Priority: urgent
X-ReferenceNumber: 1234567890
```

Note If the outbound call leg is not SIP or H.323, all headers that are transferred out from the VoiceXML application are ignored.

VoiceXML Handoff String

In Cisco IOS Release 12.3T, a <transfer> attribute *cisco-handoffexpr* is used to hand off a string to an outbound application.

Example

The example shown here demonstrates the use of the *cisco-handoffexpr* in the handoff string of the <transfer> element.

```
<transfer name="mycall"
        destexpr="'sip:joe@big.com?Subject=Car
Rental&ampPriority=urgent&ampAccount='+callerID"
        bridge="true" connecttimeout="15s" maxtime="180s" cisco-longpound ="true"
        cisco-handoffexpr="'my handoff string goes here'"
...
</transfer>
```

In Cisco IOS release 12.2(11)T, if the <transfer> element is used to transfer calls between VoiceXML applications, the script writer can embed ? and a string after the destination number as shown below.

```
<transfer name = "mycall"
        dest ="tel: 5550124?reference-prompt=http://1.10.21.7/audio/hello.au">
```

The string was forwarded to the receiving VoiceXML application which received reference-prompt=http://1.10.21.7/audio/hello.au in *session.handoff_string*.

Cisco IOS Release 12.3T allows this method of handing off a string to an outbound application; however, the use of the *cisco-handoffexpr* attribute is recommended to hand off the string.

<transfer> DTD

```
<!ELEMENT transfer (%audio; | %event.handler; | filled | %input;| prompt | property)* >
<!ATTLIST transfer
%item.attrs;
    ...
    cisco-handoffexpr%expression;#IMPLIED>
```

PSTN Outbound Calls

- Outbound calls to a SIP:URL destination cannot be routed to a PSTN leg.
- Outbound calls to a TEL:URL can be routed to a PSTN leg. The number is extracted from the URL and used as the destination number. The rest of the URL is ignored.
- Outbound header information in the future-extension field of a TEL:URL is ignored by a PSTN leg.

Headers in Inbound Calls

For incoming calls, Cisco VoiceXML allows a VoiceXML application to receive headers in session variables.

VoiceXML Document Receiving Headers

In Cisco VoiceXML, headers are received in a VoiceXML document by using session variables of the following syntax:

session.com.cisco.proto_headers[‘<attribute name>’]

where, *session.com.cisco.proto_headers* is a JavaScript object.

<attribute name> is one of the following:

- A standard SIP header name such as To, From, Subject.
- Any ASCII string limited to 256 characters.
- The predefined string name, *request-URI*; *request-URI* is the requested URI in a SIP message.

Examples

1. com.cisco.proto_headers[‘request-URI’]

   Returns the request-URI in the incoming INVITE message.

2. com.cisco.proto_headers[‘To’]

   Returns the “To” header value in the incoming INVITE message.

3. com.cisco.proto_headers

   Returns all headers concatenated into a single string with an '&' to separate each header avpair. For example "To=Joe@big.com&ampSubject=Hotel Reservation&amp...".

If the DNIS or ANI includes a SIP URI, information is returned in the following session variables:

- session.telephone.dnis

  Returns the requested URI received in the INVITE message which may or may not be an E.164 number.

- session.telephone.ani

  Returns the "From" header field received in the INVITE message which may or may not be an E.164 number.

The example below demonstrates the use of Cisco session variables in the <submit> element.

```
<submit next="http://Hotel-X/Reservation/servlets/getInfo" method="post"
    namelist="session.com.cisco.proto_headers['To']
      session.com.cisco.proto_headers['From']
      session.com.cisco.proto_headers['Subject']
      session.com.cisco.proto_headers['X-ReferenceNumber']/>
```

The namelist format submitted to the server appears as shown below:

```
session.com.cisco.proto_headers['To']=Joe@big.com&amp \
session.com.cisco.proto_headers['From']=4085550134@192.168.6.121&amp \
session.com.cisco.proto_headers['Subject']=HotelReservation&amp \
sesion.com.cisco.proto_headers['X-ReferenceNumber']=1234567890
```

If the specified variable name is a header received in a SIP message, its value is received and submitted to the web server *http://Hotel-X/Reservation/servlets/getInfo*. If the requested header name does not exist, the session variable contains an empty string. For example, if the header *To* does not exist, the string is *To*= with nothing after the = sign.

Retrieving DNIS, ANI, and Headers in Outbound Applications

If a call is handed off to an outbound application, that application retrieves all headers handed off to it from the previous application, and it also retrieves headers from the incoming call leg. The session variable *session.telephone.com.cisco.handoff.proto_headers['<attribute name>']* is used to retrieve the handoff headers. The session variable *session.telephone.com.cisco.handoff.dnis* returns the DNIS and the ANI set by the inbound application in the <transfer> element. They are returned to the specific outbound application that received the handoff call from the inbound application. The session variable *session.telephone.dnis* always returns the DNIS and the ANI from the original incoming call leg.

Example: Passing a SIP URL with Headers Using SIP

This example demonstrates two features of Cisco VoiceXML:

- The ability to place a call from an application to a destination SIP URL.
- Passing protocol headers between applications.

See the *Cisco IOS Tcl and VoiceXML Application Guide* for the originating gateway (OGW) call flow and Cisco IOS configuration.

Originating Gateway

In this example, a Tcl script is used on the OGW. The Tcl script prompts the caller for an account number. It calls the URL *sip:elmo@sip.tgw.com* and after receiving the account number, passes it in the SIP header named *accountinfo*. Other static headers such as Subject, To, From, and Priority are also passed by the Tcl script either independently or as part of the URL.

Terminating Gateway

In this example, a Tcl or VoiceXML script is used on the terminating gateway (TGW). See the *Cisco IOS Tcl and VoiceXML Application Guide* for the originating gateway (OGW) call flow and Cisco IOS configuration.

The Tcl script plays the account number and the prompt "*The number is*" both of which are received from the OGW. Other headers received are displayed in debug messages such as **debug voip ivr script**.

In this example, the VoiceXML script get_headers.vxml on the terminating gateway plays the account number and the prompt "*The number is*," both of which are received from the OGW. Other headers received are displayed in debug messages such as **debug voip application vxml puts**.

get_headers.vxml

```
<?xml version="1.0"?>
<vxml version="2.0">
<form>
    <block>
    <log> get_headers: ani is
        <value expr="session.telephone.ani"/>
    </log>
    <log> get_headers: dnis is
        <value expr="session.telephone.dnis"/>
    </log>
    <log> get_headers: header Subject is
        <value expr="session.com.cisco.proto_headers['Subject']"/>
    </log>
    <log> get_headers: header Priority is
        <value expr="session.com.cisco.proto_headers['Priority']"/>
    </log>
    <log> get_headers: header From is
        <value expr="session.com.cisco.proto_headers['From']"/>
    </log>
    <log> get_headers: header To is
        <value expr="session.com.cisco.proto_headers['To']"/>
    </log>
    <log> get_headers: header Via is
        <value expr="session.com.cisco.proto_headers['Via']"/>
    </log>
    <log> get_headers: tsp is
        <value expr="session.com.cisco.proto_headers['tsp']"/>
    </log>
    <log> get_headers: phone-context is
        <value expr="session.com.cisco.proto_headers['phone-context']"/>
    </log>
    <log> get_headers: Non-Exist is
        <value expr="session.com.cisco.proto_headers['Non-Exist']"/>
    </log>
    <log> get_headers: request-URI is
        <value expr="session.com.cisco.proto_headers['request-URI']"/>
    </log>
    <log> get_headers: All headers are
        <value expr="session.com.cisco.proto_headers['']"/>
    </log>
```

```
    <log> get_headers: header AccountInfo is
        <value expr="session.com.cisco.proto_headers['AccountInfo']"/>
    </log>
    <prompt>
        <audio src="tftp://townsend.cisco.com/audio/num_is.au"/>
        <value expr="session.com.cisco.proto_headers['AccountInfo']" class="digits"
mode="recorded"
            recsrc="tftp://servername/location/audio/en/"/>
    </prompt>
    </block>
</form>
</vxml>
```

Example: Passing a TEL URL with Headers Using H.323

This example demonstrates two features of Cisco VoiceXML:

- The ability to place a call from an application to a destination TEL URL.
- Passing protocol headers between applications.

Originating Gateway

See the *Cisco IOS Tcl and VoiceXML Application Guide* for the originating gateway (OGW) call flow and Cisco IOS configuration.

In this example, a VoiceXML script tel_headers.vxml on the OGW prompts the caller for an account number, and after receiving it, calls the TEL URL *"tel:7671234;phone-context=408;tsp=lalaland.com;Subject=HelloTelVXML;*

To=oscar@abc.com;From=nobody;Priority=urgent'+';AccountInfo='+acctInfo".

The header *accountinfo* is passed to the leg setup along with other standard and user-defined headers in the destination URL.

tel_headers.vxml

```
<vxml version="2.0">
    <form>
        <var name="mydur" expr="0"/>
        <field name="acctInfo" type="digits">
            <prompt bargein="true">
                <audio src="http://townsend.cisco.com/vxml/audio/enterAccount.au"/>
            </prompt>
        </field>
        <transfer name="mycall"

destexpr="'tel:7671234;phone-context=408;tsp=lalaland.com;Subject=HelloTelVXML;\
                To=oscar@abc.com;From=nobody;Priority=urgent'+';AccountInfo='+acctInfo"
                connecttimeout="30s" bridge="true"/>
        <filled>
                <assign name="mydur" expr="mycall$.duration"/>
                <log> ORIG OUTBOUND APP </log>
                <log>    The duration of the call is
                                <value expr="mydur"/>
                </log>
        </filled>
    </form>
</vxml>
```

Terminating Gateway

See the *Cisco IOS Tcl and VoiceXML Application Guide* for your Cisco IOS release for the terminating gateway (TGW) call flow and Cisco IOS configuration.

In this example:

- A Tcl or VoiceXML script is used on the terminating gateway.
- The Tcl script plays the account number received from the OGW and the prompt *"The number is."* Other headers received are displayed in debug messages such as **debug voip ivr script**.
- The VoiceXML script get_headers.vxml on the terminating gateway plays the account number and the prompt "*The number is*", both of which are received from the OGW. Other headers received are displayed in debug messages such as **debug voip application vxml puts**.

See the *Cisco IOS Debug Command Reference, Release 12.4T* for more information on debug commands.

SIP Blind Call Transfer

Cisco IOS releases after 12.2(11)T allow VoiceXML applications to initiate a blind call transfer using the Refer method in SIP.

For information on SIP blind call transfers, see *SIP Call Transfer and Call Forwarding Suplementary Services*.

GTD Manipulation, Cisco IOS Release 12.2(11)T

Note If you are using Cisco IOS Release 12.3 or a later release, go to the "GTD Manipulation, Cisco IOS Release 12.3" section on page 1-100.

GTD Parameters and Fields Mapped to VoiceXML Variables

The ISUP signaling message set used in SS7 networks contains information that is used for call establishment, routing and billing functions. To help transport these messages from SS7 networks (using ISUP based messages) to VoIP networks (using H.323 and SIP based messages), ISUP messages and parameters are represented in generic transparency descriptor (GTD) format and transported by the underlying call signaling messages to each node transited by the call.

Table 1-14 describes the GTD parameters and Table 1-15 maps the GTD parameters and fields to VoiceXML variables.

Table 1-14 GTD Parameters

GTD Parameter	Name	Description
RGN	Redirecting number	The number of the endpoint from which the call is re-directed.
RNI	Redirection information	Sent in the forward direction. Includes redirecting indicator, redirect reason, and redirect count.
OCN	Original called number	In the case of multiple redirections it contains the number of the endpoint at which the first redirection occured.
RNN	Redirection number	The number of the endpoint to which the call is redirected.
RNR	Redirection number restriction	Indicates whether the redirection number may be presented by the caller.
CDI	Call diversion information	Sent in the backward direction. Indicates the reason for the diversion.
GNO	Generic notification indicator	Indicates call status for supplementary services.
CNN	Connected number	Final connected number
GEA	Generic address	Contains additional addresses indentified by qualifier.
CPC	Calling party category	Type of caller: subscriber, operator, payphone, etc.
OLI	Origination line information	Type of caller for North American networks: For example, subscriber, operator, payphone.
CID	Carrier ID	Identifies a transit carrier in North American networks.
TNS	Transit network selection	Identifies the transit carrier.
PCI	Parameter compatibility information	Contains ISUP variant specific parameter that is not defined in GTD.
FDC	Field compatibility information	Contains an ISUP variant specific parameter that is not defined in GTD.
TMR	Transmission medium required	Bearer capacity
BCI	Backward call indicators	Information sent from the called party to the calling party.
CHN	Charge number	Additional number used for charging.

Table 1-15 GTD Parameters and Fields Mapped to VoiceXML Variables

NI2C Message	GTD Message	GTD Parameter. Field	Cisco VoiceXML Variable
Setup	IAM	RGN.noa	com.cisco.rgn_noa
Setup	IAM	RGN.npi	com.cisco.rgn_npi
Setup	IAM	RGN.pi	com.cisco.rgn_pi
Setup	IAM	RGN.#	com.cisco.rgn_num
Setup	IAM	RNI.ri	com.cisco.rni_ri
Setup	IAM	RNI.orr	com.cisco.rni_orr
Setup	IAM	RNI.rc	com.cisco.rni_rc
Setup	IAM	RNI.rr	com.cisco.rni_rr
Setup	IAM	OCN.noa	com.cisco.ocn_noa
Setup	IAM	OCN.npi	com.cisco.ocn_npi
Setup	IAM	OCN.pi	com.cisco.ocn_pi
Setup	IAM	OCN.num	com.cisco.ocn_num
Setup	IAM	CHN.npi	com.cisco.chn_npi
Setup	IAM	CHN.#	com.cisco.chn_num
Setup	IAM	CHN.noa	com.cisco.chn_noa
Setup	IAM	GEA.type	com.cisco.gea_type
Setup	IAM	GEA.noa	com.cisco.gea_noa
Setup	IAM	GEA.npi	com.cisco.gea_npi
Setup	IAM	GEA.cni	com.cisco.gea_cni
Setup	IAM	GEA.pi	com.cisco.gea_pi
Setup	IAM	GEA.#	com.cisco.gea_num
Setup	IAM	GEA.si	com.cisco.gea_si
Setup	IAM	CPC.cpc	com.cisco.cpc
Setup	IAM	OLI.oli	com.cisco.oli
Setup	IAM	CID.ton	com.cisco.cid_ton
Setup	IAM	CID.cid	com.cisco.cid_cid
Setup	IAM	TNS.nip	com.cisco.tns_nip
Setup	IAM	TNS.cc	com.cisco.tns_cc
Setup	IAM	TNS.tns	com.cisco.tns_tns
Setup	IAM	TNS.ton	com.cisco.tns_ton
Setup, Alert, Progress, Connect	IAM, ACM, CPG, ANM	PCI.instr	com.cisco.pci_instr
Setup, Alert, Progress, Connect	IAM, ACM, CPG, ANM	PCI.tri	com.cisco.pci_tri
Setup, Alert, Progress, Connect	IAM, ACM, CPG, ANM	PCI.dat	com.cisco.pci_dat

Table 1-15 GTD Parameters and Fields Mapped to VoiceXML Variables (continued)

NI2C Message	GTD Message	GTD Parameter. Field	Cisco VoiceXML Variable
Setup, Alert, Progress, Connect	IAM, ACM, CPG, ANM	FDC.parm	com.cisco.fdc_parm
Setup, Alert, Progress, Connect	IAM, ACM, CPG, ANM	FDC.fname	com.cisco.fdc_fname
Setup, Alert, Progress, Connect	IAM, ACM, CPG, ANM	FDC.instr	com.cisco.fdc_instr
Setup, Alert, Progress, Connect	IAM, ACM, CPG, ANM	FDC.dat	com.cisco.fdc_dat

The section below describes the Cisco VoiceXML variables that are mapped to the GTD parameters in Table 1-15.

Note All field values are case sensitive and use the seven bit US ASCII character set.

The Cisco VoiceXML variables providing GTD parameters are:

- com.cisco.rgn_noa, page 1-76
- com.cisco.rgn_npi, page 1-77
- com.cisco.rgn_pi, page 1-78
- com.cisco.rgn_num, page 1-78
- com.cisco.rni_ri, page 1-78
- com.cisco.rni_orr, page 1-78
- com.cisco.rni_rc, page 1-78
- com.cisco.rni_rr, page 1-79
- com.cisco.ocn_noa, page 1-79
- com.cisco.ocn_npi, page 1-80
- com.cisco.ocn_pi, page 1-81
- com.cisco.ocn_num, page 1-81
- com.cisco.chn_noa, page 1-81
- com.cisco.chn_npi, page 1-81
- com.cisco.chn_num, page 1-82
- com.cisco.gea_type, page 1-82
- com.cisco.gea_noa, page 1-82
- com.cisco.gea_npi, page 1-83
- com.cisco.gea_cni, page 1-84
- com.cisco.gea_pi, page 1-84
- com.cisco.gea_si, page 1-84
- com.cisco.gea_num, page 1-84

- com.cisco.cpc, page 1-84
- com.cisco.oli, page 1-85
- com.cisco.cid_ton, page 1-86
- com.cisco.cid_cid, page 1-86
- com.cisco.tns_ton, page 1-87
- com.cisco.tns_nip, page 1-87
- com.cisco.tns_cc, page 1-87
- com.cisco.tns_tns, page 1-87
- com.cisco.pci_instr, page 1-87
- com.cisco.pci_tri, page 1-88
- com.cisco.pci_dat, page 1-88
- com.cisco.fdc_parm, page 1-88
- com.cisco.fdc_fname, page 1-88
- com.cisco.fdc_instr, page 1-89
- com.cisco.fdc_dat, page 1-89

com.cisco.rgn_noa

Syntax	**com.cisco.rgn_noa**
GTD parameter	Redirecting number (RGN)

Field	noa— Nature of address
Field values	00—Unknown, number present 01—Unknown, number absent, presentation restricted 02—Unique subscriber number 03—Nonunique subscriber number 04—Unique national (significant) number 05—Nonunique national number 06—Unique international number 07—Nonunique international number 08—Network specific number 09—Nonsubscriber number 10—Subscriber number, operator requested 11—National number, operator requested 12—International number, operator requested 13—No number present, operator requested 14—No number present, cut through call to carrier 15—950+ call from local exchange carrier public station, hotel/motel or nonexchange access end office 16—Test line test code 17—Unique 3 digit national number 18—Credit card 19—International inbound 20—National or international with carrier access code included 21—Cellular - global ID GSM 22—Cellular - global ID NWT 900 23—Cellular - global ID autonet 24—Mobile (other) 25—Ported number 26—VNET 27—International operator to operator outside WZ1 28—International operator to operator inside WZ1 29—Operator requested - treated 30—Network routing number in national (significant) format 31—Network routing number in network specific format 32—Network routing number concatenated with called directory number 33—Screened for number portability 34—Abbreviated number

com.cisco.rgn_npi

Syntax	**com.cisco.rgn_npi**
GTD parameter	Redirecting number (RGN)
Field	npi— Numbering plan indicator
Field values	1—ISDN numbering plan 2—Data numbering plan 3—Telex numbering plan 4—Private numbering plan 5—National 6—Maritime mobile 7—Land mobile 8—ISDN mobile 252—Unknown

com.cisco.rgn_pi

Syntax	**com.cisco.rgn_pi**
GTD parameter	Redirecting number (RGN)
Field	pi—Presentation indicator
Field values	0—Unknown 1—Presentation allowed 2—Presentation not allowed 3—Address not available

com.cisco.rgn_num

Syntax	**com.cisco.rgn_num**
GTD parameter	Redirecting number (RGN)
Field	# —Address. # is represented by num in the Cisco VoiceXML variable.
Field values	A string of one or more telephony digits.

com.cisco.rni_ri

Syntax	**com.cisco.rni_ri** [1]
GTD parameter	Redirection information (RNI)
Field	ri— Redirecting indicator
Field values	0—No redirection or unknown 1—Call rerouted 2—Call rerouted, all redirection info presentation restricted 3—Call diverted 4—Call diverted, all redirection information presentation restricted 5—Call rerouted, redirection number presentation restricted 6—Call diversion, redirection number presentation restricted

1. This session variable may be replaced in a future Cisco IOS release by an alternate method of accessing signaling information.

com.cisco.rni_orr

Syntax	**com.cisco.rni_orr**[1]
GTD parameter	Redirection information (RNI)
Field	orr— Original redirection reason
Field values	1—User busy 2—No reply 3—Unconditional 4—Deflection during alerting 5—Deflection immediate response 6—Mobile subscriber not reachable 252—Unknown

com.cisco.rni_rc

Syntax	**com.cisco.rni_rc** [1]
GTD parameter	Redirection information (RNI)
Field	rc— Redirection counter
Field values	Valid values are in the range of 1–15.

com.cisco.rni_rr

Syntax	**com.cisco.rni_rr**[1]
GTD parameter	Redirection information (RNI)
Field	rr— Redirection reason
Field values	V1—User busy 2—No reply 3—Unconditional 4—Deflection during alerting 5—Deflection immediate response 6—Mobile subscriber not reachable 252—Unknown

com.cisco.ocn_noa

Syntax	**com.cisco.ocn_noa**[1]
GTD parameter	Original called number (OCN)

Field	noa— Nature of address
Field values	0—Unknown, number present 1—Unknown, number absent, presentation restricted 2—Unique subscriber number 3—Nonunique subscriber number 4—Unique national (significant) number 5—Nonunique national number 6—Unique international number 7—Nonunique international number 8—Network specific number 9—Nonsubscriber number 10—Subscriber number, operator requested 11—National number, operator requested 12—International number, operator requested 13—No number present, operator requested 14—No number present, cut through call to carrier 15—950+ call from local exchange carrier public station, hotel/motel or nonexchange access end office 16—Test line test code 17—Unique 3 digit national number 18—Credit card 19—International inbound 20—National or international with carrier access code included 21—Cellular - global ID GSM 22 - Cellular - global ID NWT 900 23 - Cellular - global ID autonet 24 - Mobile (other) 25 - Ported number 26 - VNET 27 - International operator to operator outside WZ1 28 - International operator to operator inside WZ1 29 - Operator requested - treated 30 - Network routing number in national (significant) format 31 - Network routing number in network specific format 32 - Network routing number concatenated with called directory number 33 - Screened for number portability 34 - Abbreviated number

com.cisco.ocn_npi

Syntax	**com.cisco.ocn_pi**[1]
GTD parameter	Original called number (OCN)
Field	npi— Numbering plan indicator
Field values	1—ISDN numbering plan 2—Data numbering plan 3—Telex numbering plan 4—private numbering plan 5—national 6—maritime mobile 7—land mobile 8—ISDN mobile 252—unknown

com.cisco.ocn_pi

Syntax	**com.cisco.ocn_pi**[1]
GTD parameter	Original called number (OCN)
Field	pi— Presentation indicator
Field values	0—unknown 1—presentation allowed 2—presentation not allowed 3—address not available

com.cisco.ocn_num

Syntax	**com.cisco.ocn_num**[1]
GTD parameter	Original called number (OCN)
Field	#— Address. # is representated by num in the Cisco VoiceXML variable.
Field values	A string of one or more telephony digits.

com.cisco.chn_noa

Syntax	**com.cisco.chn_noa**[1]
GTD parameter	Charge number (CHN)
Field	noa— Nature of address
Field values	0—unknown 1—calling subscriber - not available 2—calling subscribers number 3—calling subscriber - national number 4—calling subscriber - international number 5—calling subscriber VPN 6—called subscriber no number present 7—called subscriber number 8—called subscriber national number 9—called subscriber international number 10—called subscriber VPN 11—VNET

com.cisco.chn_npi

Syntax	**com.cisco.chn_npi**[1]
GTD parameter	Charge number (CHN)
Field	npi— Numbering plan indicator
Field values	1—ISDN numbering plan 2—data numbering plan 3—telex numbering plan 4—private numbering plan 5—national 6—maritime mobile 7—land mobile 8—ISDN mobile 252—unknown

com.cisco.chn_num

Syntax	**com.cisco.chn_num**[1]
GTD parameter	Charge number (CHN)
Field	#— Address. # is represented by num in the Cisco VoiceXML variable.
Field values	A string of one or more telephony digits.

com.cisco.gea_type

Syntax	**com.cisco.gea_type** [1]
GTD parameter	Generic address (GEA)
Field	type— Type of address
Field values	0—dialed number 1—destination number/additional called number 2—supplemental user provided calling address; failed network screening 3—supplemental user provided calling address; not screened 4—completion number 5—ported number 6—transfer number 1 7—transfer number 2 8—transfer number 3 9—transfer number 4 10—transfer number 5 11—transfer number 6 12—caller emergency service ID 13—reserved 14—called number emergency service ID

com.cisco.gea_noa

Syntax	**com.cisco.gea_noa**[1]
GTD parameter	Generic address (GEA)

Field	noa— Numbering plan indicator
Field values	0—unknown, number present 1—unknown, number absent, presentation restricted 2—unique subscriber number 3—nonunique subscriber number 4—unique national (significant) number 5—nonunique national number 6—unique international number 7—nonunique international number 8—network specific number 9—nonsubscriber number 10—subscriber number, operator requested 11—national number, operator requested 12—international number, operator requested 13—no number present, operator requested 14—no number present, cut through call to carrier 15—950+ call from local exchange carrier public station, hotel/motel or nonexchange access end office 16—test line test code 17—unique 3 digit national number 18—credit card 19—international inbound 20—national or international with carrier access code included 21—cellular - global ID GSM 22—cellular - global ID NWT 900 23—cellular - global ID autonet 24—mobile (other) 25—ported number 26—VNET 27—international operator to operator outside WZ1 28—international operator to operator inside WZ1 29—operator requested - treated 30—network routing number in national (significant) format 31—network routing number in network specific format 32—network routing number concatenated with called directory number 33—screened for number portability 34—abbreviated number

com.cisco.gea_npi

Syntax	**com.cisco.gea_npi**[1]
GTD parameter	Generic address (GEA)
Field	npi— Numbering plan indicator
Field values	1—ISDN numbering plan 2—data numbering plan 3—telex numbering plan 4—private numbering plan 5—national 6—maritime mobile 7—land mobile 8—ISDN mobile 252—unknown

com.cisco.gea_cni

Syntax	**com.cisco.gea_cni**[1]
GTD parameter	Generic address (GEA)
Field	cni— Complete number indicator
Field values	0—unknown 1—number complete 2—number incomplete

com.cisco.gea_pi

Syntax	**com.cisco.gea_pi** [1]
GTD parameter	Generic address (GEA)
Field	pi— Address presentation indicator
Field values	0—unknown 1—presentation allowed 2—presentation restricted 3—address not available

com.cisco.gea_si

Syntax	**com.cisco.gea_si** [1]
GTD parameter	Generic address (GEA)
Field	si— Screening indicator
Field values	1—user provided not screened (verified) 2—user provided screening passed 3—user provided screening failed 4—network provided 252—unknown or not applicable

com.cisco.gea_num

Syntax	**com.cisco.gea_num**[1]
GTD parameter	Generic address (GEA)
Field	#—Address. # is represented by num in the Cisco VoiceXML variable.
Field values	A string of one or more telephony digits.

com.cisco.cpc

Syntax	**com.cisco.cpc**[1]
GTD parameter	Calling party category (CPC)

Field	cpc— Calling party category
Field values	0—unknown 1—operator, language French 2—operator, language English 3—operator, language German 4—operator, language Russian 5—operator, language Spanish 6—admin1 7—admin2 8—admin3 9—ordinary calling subscriber 10—ordinary calling subscriber with customer meter 11—calling subscriber with priority 12—data call 13—test call 14—customer pay phone 15—public pay phone 16—emergency service call 17—high priority emergency service call 18—national security and emergency preparedness (NS/EP call) 19—trunk offering 20—mobile customer 21—PBX subscriber 22—operator with forward facility 23—intercept operator 24—cross-border operator 25—long distance pay phone 26—international pay phone 27—international test equipment 28—check calling party number 29—national operator

com.cisco.oli

Syntax	**com.cisco.oli**[1]
GTD parameter	Originating line information (OLI)

Field	oli— Originating line information
Field values	0—pots 1—multiparty line 2—ANI failure 6—station level rating 7—special operator handling required 8—inter-LATA restricted 10—test call 20—AIOD-listed DN sent 23—coin or noncoin on calls using database access 24—800 service call 25—800 service call from a pay station 27—pay phone using coin control signaling 29—prison/inmate service 30—intercept (blank) 31—intercept (trouble) 32—intercept (regular) 34—telco operator handled call 36—CPE 52—OUTWATS 60—TRS call from unrestricted line 61—wireless/cellular PCS (type 1) 62—wireless/cellular PCS (type 2) 63—wireless/cellular PCS (roaming) 66—TRS call from hotel 67—TRS call from restricted line 68—inter-LATA restricted hotel 78—inter-LATA restricted coin-less 70—private pay-stations 93—private virtual network

com.cisco.cid_ton

Syntax	**com.cisco.cid_ton**[1]
GTD parameter	Carrier identification (CID)
Field	ton— Type of network
Field values	0—unknown 1—ITU/CCITT 2—national

com.cisco.cid_cid

Syntax	**com.cisco.cid_cid**[1]
GTD parameter	Carrier identification (CID)
Field	cid— Carrier identification
Field values	One or more characters from 0–9 and A–F to identify the carrier.

com.cisco.tns_ton

Syntax	**com.cisco.tns_ton**[1]
GTD parameter	Transit network selection (TNS)
Field	ton— Type of network
Field values	0—unknown 1—ITU/CCITT 2—national

com.cisco.tns_nip

Syntax	**com.cisco.tns_nip**[1]
GTD parameter	Transit network selection (TNS)
Field	nip— Network identification plan
Field values	1—public data network identification code 2—public land mobile network identification code 3—3-digit carrier identification with circuit code 4—4-digit carrier identification with circuit code 252—unknown

com.cisco.tns_cc

Syntax	**com.cisco.tns_cc**[1]
GTD parameter	Transit network selection (TNS)
Field	cc— Circuit code
Field values	1—international call, no operator requested 2—international call, operator requested 251—not applicable 252—unknown

com.cisco.tns_tns

Syntax	**com.cisco.tns_ns**[1]
GTD parameter	Transit network selection (TNS)
Field	tns— Network identification
Field values	IA5—Characters from 0–9 and A–F of length defined by the fields ton and nip.

com.cisco.pci_instr

Syntax	**com.cisco.pci_instr**[1]
GTD parameter	Parameter compatibility (PCI)

Field	instr— Instruction
Field values	0—release call regardless of the ability to forward the parameter 1—discard message regardless of the ability to forward the parameter, no notification required, but continue call 2—discard message regardless of the ability to forward the parameter, send notification (in confusion), but continue call 3—discard parameter regardless of the ability to forward the parameter, no notification required, but continue call 4—discard parameter regardless of the ability to forward the parameter, send notification (in Confusion) but continue call 5—attempt to forward the parameter, if unable to forward the parameter release the call 6—attempt to forward the parameter, if unable to forward the parameter discard message without notification but continue the call 7—attempt to forward the parameter, if unable to forward the parameter, discard message, send notification but continue the call. 8—attempt to forward the parameter, if unable to forward the parameter, discard the parameter, without notification but continue the call. 9—attempt to forward the parameter, if unable to forward the parameter discard the parameter, send notification but continue the call. 252—unknown

com.cisco.pci_tri

Syntax	**com.cisco.pci_tri**[1]
GTD parameter	Parameter compatibility (PCI)
Field	tri— Transit at intermediate exchange indicator
Field values	0—no transit 1—yes transit

com.cisco.pci_dat

Syntax	**com.cisco.pci_dat**[1]
GTD parameter	Parameter compatibility (PCI)
Field	dat— Representation of the parameter contents.
Field values	One or more characters from 0–9 and A–F representing the hexadecimal value of the parameter.

com.cisco.fdc_parm

Syntax	**com.cisco.fdc_parm**[1]
GTD parameter	Known field compatibility information (FDC)
Field	parm— Parameter name
Field values	A string of three ASCII characters.

com.cisco.fdc_fname

Syntax	**com.cisco.fdc_field**[1]
GTD parameter	Known field compatibility information (FDC)

Field	fname— Field name. Refers to the field name declared against the parameter.
Field values	A string of five ASCII characters with a lower case alphabetic field name.

com.cisco.fdc_instr

Syntax	**com.cisco.fdc_instr**[1]
GTD parameter	Known field compatibility information (FDC)
Field	instr— Instruction
Field values	1—release call if not understood, regardless of the ability to forward the call. 2—use the default value if not understood regardless of the ability to forward; no notification required, but continue the call. 3—use the default value if not understood regardless of the ability to forward; send notification (in Confusion), but continue the call. 4—attempt to forward value; if unable to forward the value, release the call. 5—attempt to forward value; if unable to forward the value, use default value without notification, but continue the call. 6—attempt to forward value; if unable to forward the value, use default value and send notification, but continue the call. 252—unknown

com.cisco.fdc_dat

Syntax	**com.cisco.fdc_dat**[1]
GTD parameter	Known field compatibility information (FDC)
Field	dat— Hexadecimal representation of the contents of the parameter .
Field values	One or more ASCII characters from 0–9 and A–F . The entire parameter uses ASCII characters to represent hexadecimal values.

GTD Parameter Reference

This section describes the GTD parameters used for implementing call control features in Cisco VoiceXML. For detailed information on GTD parameters, see *MIME Media Types for Generic Transparency Descriptor (GTD) Objects*.

GTD parameter	Redirecting number (RGN)
Field	noa— Nature of address
Field values	00—unknown, number present 01—unknown, number absent, presentation restricted 02—unique subscriber number 03—nonunique subscriber number 04—unique national (significant) number 05—nonunique national number 06—unique international number 07—nonunique international number 08—network specific number 09—nonsubscriber number 10—subscriber number, operator requested 11—national number, operator requested 12—international number, operator requested 13—no number present, operator requested 14—no number present, cut through call to carrier 15—950+ call from local exchange carrier public station, hotel/motel or nonexchange access end office 16—test line test code 17—unique 3 digit national number 18—credit card 19—international inbound 20—national or international with carrier access code included 21—cellular - global ID GSM 22—cellular - global ID NWT 900 23—cellular - global ID autonet 24—mobile (other) 25—ported number 26—VNET 27—International operator to operator outside WZ1 28—International operator to operator inside WZ1 29—operator requested - treated 30—network routing number in national (significant) format 31—network routing number in network specific format 32—network routing number concatenated with called directory number 33—screened for number portability 34—abbreviated number

GTD parameter	Redirecting number (RGN)
Field	npi— Numbering plan indicator
Field values	1—ISDN numbering plan 2—data numbering plan 3—telex numbering plan 4—private numbering plan 5—national 6—maritime mobile 7—land mobile 8—ISDN mobile 252—unknown

GTD parameter	Redirecting number (RGN)
Field	pi—Presentation indicator
Field values	0—unknown 1—presentation allowed 2—presentation not allowed 3—address not available

GTD parameter	Redirecting number (RGN)
Field	# —Address. # is represented by num in the Cisco VoiceXML variable.
Field values	A string of one or more telephony digits.

GTD parameter	Redirection information (RNI)
Field	ri— Redirecting indicator
Field values	0—no redirection or unknown 1—call rerouted 2—call rerouted, all redirection info presentation restricted 3—call diverted 4—call diverted, all redirection information presentation restricted 5—call rerouted, redirection number presentation restricted 6—call diversion, redirection number presentation restricted

GTD parameter	Redirection information (RNI)
Field	orr— Original redirection reason
Field values	1—user busy 2—no reply 3—unconditional 4—deflection during alerting 5—deflection immediate response 6—mobile subscriber not reachable 252—unknown

GTD parameter	Redirection information (RNI)
Field	rc— Redirection counter
Field values	Valid values are in the range of 1–15.

GTD parameter	Redirection information (RNI)
Field	rr— Redirection reason
Field values	V1—user busy 2—no reply 3—unconditional 4—deflection during alerting 5—deflection immediate response 6—mobile subscriber not reachable 252—unknown

GTD parameter	Original called number (OCN)

Field	noa— Nature of address
Field values	0—unknown, number present 1—unknown, number absent, presentation restricted 2—unique subscriber number 3—nonunique subscriber number 4—unique national (significant) number 5—nonunique national number 6—unique international number 7—nonunique international number 8—network specific number 9—nonsubscriber number 10—subscriber number, operator requested 11—national number, operator requested 12—international number, operator requested 13—no number present, operator requested 14—no number present, cut through call to carrier 15—950+ call from local exchange carrier public station, hotel/motel or nonexchange access end office 16—test line test code 17—unique 3 digit national number 18—credit card 19—international inbound 20—national or international with carrier access code included 21—cellular - global ID GSM 22 - cellular - global ID NWT 900 23 - cellular - global ID autonet 24 - mobile (other) 25 - ported number 26 - VNET 27 - international operator to operator outside WZ1 28 - international operator to operator inside WZ1 29 - operator requested - treated 30 - network routing number in national (significant) format 31 - network routing number in network specific format 32 - network routing number concatenated with called directory number 33 - screened for number portability 34 - abbreviated number

GTD parameter	Original called number (OCN)
Field	npi— Numbering plan indicator
Field values	1—ISDN numbering plan 2—data numbering plan 3—telex numbering plan 4—private numbering plan 5—national 6—maritime mobile 7—land mobile 8—ISDN mobile 252—unknown

GTD parameter	Original called number (OCN)
Field	pi— Presentation indicator
Field values	0—unknown 1—presentation allowed 2—presentation not allowed 3—address not available

GTD parameter	Original called number (OCN)
Field	#— Address. # is representated by num in the Cisco VoiceXML variable.
Field values	A string of one or more telephony digits.

GTD parameter	Charge number (CHN)
Field	noa— Nature of address
Field values	0—unknown 1—calling subscriber - not available 2—calling subscribers number 3—calling subscriber - national number 4—calling subscriber - international number 5—calling subscriber VPN 6—called subscriber no number present 7—called subscriber number 8—called subscriber national number 9—called subscriber international number 10—called subscriber VPN 11—VNET

GTD parameter	Charge number (CHN)
Field	npi— Numbering plan indicator
Field values	1—ISDN numbering plan 2—data numbering plan 3—Telex numbering plan 4—Private numbering plan 5—National 6—Maritime mobile 7—Land mobile 8—ISDN mobile 252—Unknown

GTD parameter	Charge number (CHN)
Field	#— Address. # is represented by num in the Cisco VoiceXML variable.
Field values	A string of one or more telephony digits.

GTD parameter	Generic address (GEA)
Field	type— Type of address
Field values	0—Dialed number 1—Destination number/additional called number 2—Supplemental user provided calling address; failed network screening 3—Supplemental user provided calling address; not screened 4—Completion number 5—Ported number 6—Transfer number 1 7—Transfer number 2 8—Transfer number 3 9—Transfer number 4 10—Transfer number 5 11—Transfer number 6 12—Caller emergency service ID 13—Reserved 14—Called number emergency service ID

GTD parameter	Generic address (GEA)
Field	noa— Numbering plan indicator
Field values	0—Unknown, number present 1—Unknown, number absent, presentation restricted 2—Unique subscriber number 3—Nonunique subscriber number 4—Unique national (significant) number 5—Nonunique national number 6—Unique international number 7—Nonunique international number 8—Network specific number 9—Nonsubscriber number 10—Subscriber number, operator requested 11—National number, operator requested 12—International number, operator requested 13—No number present, operator requested 14—No number present, cut through call to carrier 15—950+ call from local exchange carrier public station, hotel/motel or nonexchange access end office 16—Test line test code 17—Unique 3 digit national number 18—Credit card 19—International inbound 20—National or international with carrier access code included 21—Cellular - global ID GSM 22—Cellular - global ID NWT 900 23—Cellular - global ID autonet 24—Mobile (other) 25—Ported number 26—VNET 27—International operator to operator outside WZ1 28—International operator to operator inside WZ1 29—Operator requested - treated 30—Network routing number in national (significant) format 31—Network routing number in network specific format 32—Network routing number concatenated with called directory number 33—Screened for number portability 34—Abbreviated number

GTD parameter	Generic address (GEA)
Field	npi— Numbering plan indicator
Field values	1—ISDN numbering plan 2—Data numbering plan 3—Telex numbering plan 4—Private numbering plan 5—National 6—Maritime mobile 7—Land mobile 8—ISDN mobile 252—Unknown

GTD parameter	Generic address (GEA)
Field	cni— Complete number indicator
Field values	0—Unknown 1—Number complete 2—Number incomplete

GTD parameter	Generic address (GEA)
Field	pi— Address presentation indicator
Field values	0—Unknown 1—Presentation allowed 2—Presentation restricted 3—Address not available

GTD parameter	Generic address (GEA)
Field	si— Screening indicator
Field values	1—User provided not screened (verified) 2—User provided screening passed 3—User provided screening failed 4—Network provided 252—Unknown or not applicable

GTD parameter	Generic address (GEA)
Field	#—Address. # is represented by num in the Cisco VoiceXML variable.
Field values	A string of one or more telephony digits.

GTD parameter	Carrier identification (CID)
Field	ton— Type of network
Field values	0—Unknown 1—ITU/CCITT 2—National

GTD parameter	Calling party category (CPC)

Field	cpc— Calling party category
Field values	0—Unknown 1—Operator, language French 2—Operator, language English 3—Operator, language German 4—Operator, language Russian 5—Operator, language Spanish 6—Admin1 7—Admin2 8—Admin3 9—Ordinary calling subscriber 10—Ordinary calling subscriber with customer meter 11—Calling subscriber with priority 12—Data call 13—Test call 14—Customer pay phone 15—Public pay phone 16—Emergency service call 17—High priority emergency service call 18—National security and emergency preparedness (NS/EP call) 19—Trunk offering 20—Mobile customer 21—PBX subscriber 22—Operator with forward facility 23—Intercept operator 24—Cross-border operator 25—Long distance pay phone 26—International pay phone 27—International test equipment 28—Check calling party number 29—National operator

GTD parameter	Originating line information (OLI)
Field	oli— Originating line information
Field values	0—Pots 1—Multiparty line 2—ANI failure 6—Station level rating 7—Special operator handling required 8—Inter-LATA restricted 10—Test call 20—AIOD-listed DN sent 23—Coin or noncoin on calls using database access 24—800 service call 25—800 service call from a pay station 27—Pay phone using coin control signaling 29—Prison/inmate service 30—Intercept (blank) 31—Intercept (trouble) 32—Intercept (regular) 34—Telco operator handled call 36—CPE 52—OUTWATS 60—TRS call from unrestricted line 61—Wireless/cellular PCS (type 1) 62—Wireless/cellular PCS (type 2) 63—Wireless/cellular PCS (roaming) 66—TRS call from hotel 67—TRS call from restricted line 68—Inter-LATA restricted hotel 78—Inter-LATA restricted coin-less 70—Private pay-stations 93—Private virtual network

GTD parameter	Carrier identification (CID)
Field	cid— Carrier identification
Field values	One or more characters from 0–9 and A–F to identify the carrier.

GTD parameter	Transit network selection (TNS)
Field	ton— Type of network
Field values	0—Unknown 1—ITU/CCITT 2—National

GTD parameter	Transit network selection (TNS)
Field	nip— Network identification plan
Field values	1—Public data network identification code 2—Public land mobile network identification code 3—3-digit carrier identification with circuit code 4—4-digit carrier identification with circuit code 252—Unknown

GTD parameter	Transit network selection (TNS)
Field	cc— Circuit code
Field values	1—International call, no operator requested 2—International call, operator requested 251—Not applicable 252—Unknown

GTD parameter	Transit network selection (TNS)
Field	tns— Network identification
Field values	IA5—Characters from 0–9 and A–F of length defined by the fields ton and nip.

GTD parameter	Parameter compatibility (PCI)
Field	instr— Instruction
Field values	0—Release call regardless of the ability to forward the parameter 1—Discard message regardless of the ability to forward the parameter, no notification required, but continue call 2—Discard message regardless of the ability to forward the parameter, send notification (in confusion), but continue call 3—Discard parameter regardless of the ability to forward the parameter, no notification required, but continue call 4—Discard parameter regardless of the ability to forward the parameter, send notification (in Confusion) but continue call 5—Attempt to forward the parameter, if unable to forward the parameter release the call 6—Attempt to forward the parameter, if unable to forward the parameter discard message without notification but continue the call 7—Attempt to forward the parameter, if unable to forward the parameter, discard message, send notification but continue the call. 8—Attempt to forward the parameter, if unable to forward the parameter, discard the parameter, without notification but continue the call. 9—Attempt to forward the parameter, if unable to forward the parameter discard the parameter, send notification but continue the call. 252—Unknown

GTD parameter	Parameter compatibility (PCI)
Field	tri— Transit at intermediate exchange indicator
Field values	0—No transit 1—Yes transit

GTD parameter	Parameter compatibility (PCI)
Field	dat— Representation of the parameter contents.
Field values	One or more characters from 0–9 and A–F representing the hexadecimal value of the parameter.

GTD parameter	Known field compatibility information (FDC)
Field	parm— Parameter name
Field values	A string of three ASCII characters.

GTD parameter	Known field compatibility information (FDC)
Field	fname— Field name. Refers to the field name declared against the parameter.
Field values	A string of five ASCII characters with a lower case alphabetic field name.

GTD parameter	Known field compatibility information (FDC)
Field	instr— Instruction
Field values	1—Release call if not understood, regardless of the ability to forward 2—Use the default value if not understood regardless of the ability to forward; no notification required, but continue call 3—Use the default value if not understood regardless of the ability to forward; send notification (in Confusion), but continue call 4—Attempt to forward value; if unable to forward the value, release the call 5—Attempt to forward value; if unable to forward the value, use default value without notification, but continue the call 6—Attempt to forward value; if unable to forward the value, use default value and send notification, but continue the call. 252—Unknown

GTD parameter	Known field compatibility information (FDC)
Field	dat— Hexadecimal representation of the contents of the parameter.
Field values	One or more ASCII characters from 0–9 and A–F . The entire parameter uses ASCII characters to represent hexadecimal values.

GTD Manipulation, Cisco IOS Release 12.3

Cisco VoiceXML enables a VoiceXML script to access GTD parameters before and after a call transfer, to append and override GTD parameters, and to create new GTD messages. For example, if an incoming call setup event contains a GTD IAM message, the <transfer> element uses the IAM message and overrides or appends the parameters specified by the <transfer> GTD attributes. If the incoming setup event does not contain the IAM message, and if the VoiceXML script wants to send a GTD message on the outgoing leg, it creates a new IAM GTD message and adds or appends the required GTD parameters to it. The VoiceXML script sends this new GTD message to the outgoing leg using the <transfer> element. The GTD messages can only contain parameters specified in the tables in this document.

GTD messages and their individual parameters are represented as JavaScript objects. The GTD message object is at the top level, and the individual parameters are represented at the subobject level. The individual subfields of GTD parameters can be accessed and modified as properties of the parameter subobject. GTD messages received on an incoming call leg during the setup event are represented by the Cisco VoiceXML session variable *com.cisco.signal.gtdlist* which is an array of GTD objects indexed by the call signal event name (setup_indication). This session variable allows the VoiceXML script to only read the GTD parameters received in the setup message. Being a session variable, the VoiceXML script has *read-only* acccess to the GTD parameters.

The Cisco VoiceXML shadow variable of the form *<transfer_name>$.com.cisco.signal.gtdlist* is used to read GTD parameters on the outgoing leg after the call transfer terminates. This shadow variable is a read-only variable representing an array of GTD objects indexed by the call signal event name. The VoiceXML script uses the Cisco shadow variable to only read GTD messages passed in the alert, disconnect, and connect events. Reading a valid parameter that is not included in a GTD message results in the return value *undefined*. Setting a parameter that is not supported causes an exception error event to be thrown by the system.

The object *com.cisco.objclass.gtd* is a Cisco object class that is used to modify an existing GTD message or create a new GTD message that allows the VoiceXML script to have read-write access to the GTD messages.

The VoiceXML script can use the Cisco attribute *cisco-gtd* with the <*transfer*> element to send a modified GTD message to the outbound call leg. The attribute represents the GTD object that is sent on the outbound leg. *cisco-gtd* is also used with the <*disconnect*> element to send a GTD message during a call release.

Usage Guidelines

Table 1-16 describes GTD parameters. Table 1-17 describes the GTD messages mapped to the GTD parameters and their fields.

Note A GTD parameter can be sent or received in any GTD message.

Table 1-16 GTD Parameters

GTD Parameter	Name	Description
RGN	Redirecting Number	The number of the endpoint from which the call is re-directed.
RNI	Redirection Information	Sent in the forward direction. Includes redirecting indicator, redirect reason, and redirect count.
OCN	Original Called Number	In the case of multiple redirections it contains the number of the endpoint at which the first redirection occured.
RNN	Redirection Number	The number of the endpoint to which the call is redirected.
RNR	Redirection Number Restriction	Indicates whether the redirection number may be presented by the caller.
CDI	Call Diversion Information	Sent in the backward direction. Indicates the reason for the diversion.
GNO	Generic Notification Indicator	Indicates call status for supplementary services.
CNN	Connected number	Final connected number
GEA	Generic address	Contains additional addresses indentified by qualifier.
CPC	Calling party category	Type of caller: subscriber, operator, payphone, etc.
OLI	Origination line information	Type of caller for North American networks: For example, subscriber, operator, payphone.
CID	Carrier ID	Identifies a transit carrier in North American networks.
TNS	Transit network selection	Identifies the transit carrier.

Table 1-16 GTD Parameters (continued)

GTD Parameter	Name	Description
PCI	Parameter compatibility information	Contains ISUP variant specific parameter that is not defined in GTD.
FDC	Field compatibility information	Contains an ISUP variant specific parameter that is not defined in GTD.
TMR	Transmission medium required	Bearer capacity
BCI	Backward call indicators	Information sent from the called party to the calling party.
CHN	Charge number	Additional number used for charging.
CAI	Cause indicator	CCITT and ANSI cause codes.
CPN	Called party number	Same as DNIS. It provides additional parameters such as nature of address and numbering plan indicator.
RDC	Redirect capability	Indicates the point at which redirection is possible in a call setup.
UUS	User-to-user information	—

Table 1-17 GTD Parameters and Fields

GTD Parameter. Field
RGN.noa
RGN.npi
RGN.pi
RGN.#
RNI.ri
RNI.orr
RNI.rc
RNI.rr
OCN.noa
OCN.npi
OCN.pi
OCN.#
CHN.npi
CHN.#
CHN.noa
GEA.type

Table 1-17 GTD Parameters and Fields (continued)

GTD Parameter. Field
GEA.noa
GEA.npi
GEA.cni
GEA.pi
GEA.#
GEA.si
CPC.cpc
OLI.oli
CID.ton
CID.cid
TNS.nip
TNS.cc
TNS.tns
PCI.instr
PCI.tri
PCI.dat
FDC.parm
FDC.instr
FDC.dat
RNN.noa
RNN.inn
RNN.npi
RNN.#
RNR.rnr
CDI.nso
CDI.rr
GNO.ni
CNN.noa
CNN.npi
CNN.pi
CNN.si
CNN.#
PRN.prot
PRN.c
PRN.o

Table 1-17 GTD Parameters and Fields (continued)

GTD Parameter. Field
PRN.prv
RDC.rc
CGN.noa
CGN.npi
CGN.pi
CGN.si
CPN.noa
CPN.npi
CAI.cau [1]
UUS.dat

1. CAI.cau is set by passing it as an attribute *cisco-disc_cause* through the <disconnect> element.

To create, modify, and read GTDs, the following Cisco objects are used:

- **com.cisco.objclass.gtd**

  This is a JavaScript Cisco-specific object class that is used to create new GTD messages. To create an object of a specific class (instantiate) for creating a new GTD message, use:

  X= new com.cisco.objclass.gtd()

  Later, the GTD message can be represented in a format similar to:

  X.message_type = "IAM"

Note IAM is used only as an example.

- **com.cisco.signal.gtdlist**

  This is a Cisco-specific session variable that represents an array of GTD objects (each object representing a GTD message) for signaling events arriving on the incoming leg. The elements of this array can be accessed in read-only mode. The VoiceXML script accesses the GTD parameters and their fields from this session variable which is indexed by the signaling event as shown in the following format:

  com.cisco.signal.gtdlist[*"setup_indication"*] where

  The elements of this array are read-only.

  The VoiceXML/JSE script can access the GTD parameters and their fields from *com.cisco.signal.gtdlist* indexed by the signaling event as in:

  X = com.cisco.signal.gtdlist["setup_indication"]

  Initially, at the start of a session, an array is created with only one element (for Setup Ind event). Later, as other events are received, the GTD objects for the associated GTD messages are added to this array. For example,

X = com.cisco.signal.gtdlist[“setup_indication”] where X is read-only.

dat1 = X.PCI[0].dat for the *dat* field in the first instance of the PCI parameter.

Another Object can be created from *com.cisco.signal.gtdlist* using the *new* operation in com.cisco.objclass.gtd as in:

Y = new com.cisco.objclass.gtd(com.cisco.signal.gtdlist[“setup_indication”])

In this case, the new object Y contains a copy of the GTD message that arrived with the Setup Ind event and is a read-write object. *Y* can now be used for modifying GTD parameters.

- <*transfer-name*>**$.com.cisco.signal.gtdlist**

  This is a Cisco-specific shadow variable that is used by the VoiceXML script to access GTD parameters on the outgoing leg after the <transfer> is complete. The shadow variable consists of the transfer name and “$” prepended to *com.cisco.signal.gtdlist*.

  In <*transfer name= “gtd_xfer”..../>*, *gtd_xfer$.com.signal.gtdlist* is an array of GTD objects for the outgoing leg indexed by signaling event names mentioned in *gtd_xfer$.com.cisco.signal.gtdlist*[*“<event-name>”*] where

  <*event-name*> represents the following signaling events arriving on the incoming leg:

  – alert_indication
  – connect_indication
  – disconnect_indication

  Example:

```
<assign name="data" expr=
"gtd_xfer$.com.cisco.signal.gtdlist['alert_indication'].PCI[0].dat"/>
```

GTD Object and Parameter Syntax

GTD Parameter Syntax

<gtd-object-name>.<gtd-parameter-name> [<instance-number>]

For example, the first instance of the PCI parameter is referenced as *my_gtd.PCI[0]*, where *my_gtd* is created as an instance of the GTD message or *com.cisco.signal.gtdlist*.

GTD Parameter Field Syntax

<gtd-object-name>.<gtd-parameter-name>[<instance-number>].<field-name>

For example, the *data* field of the first instance of the PCI parameter is referenced as *my_gtd.PCI[0].dat*, where *my_gtd* is created as an instance of the GTD message or *com.cisco.signal.gtdlist*.

Multiple Instances

A parameter in a GTD message can have multiple instances which need not be contiguous. In VoiceXML and JavaScript, indexing of instance numbers of GTD parameters starts with zero. For example, the first instance of a GTD parameter is referenced with the index number zero, the second instance is referenced with the index number one, the third instance is referenced with the index number two.

The syntax <*gtd-object-name*>.<*gtd-parameter-object-name*>*[*<*instance-number*>*]* refers to a specific instance. The property *instance_count* is exposed to every GTD parameter object. The VoiceXML script must first read this property before accessing a specific instance. The instance number must be less than or equal to the total number of instances read by the script.

For example:

```
<assign name="my_gtd" expr= "new
com.cisco.objclass.gtd(com.cisco.signal.gtdlist['setup_indication'])"/>
<assign name="total_pci_inst" expr="my_gtd.PCI.instance_count"/>
<assign name="pci2" expr="my_gtd.PCI[2]" >/
<assign name="tri2" expr="my_gtd.PCI[2].tri">/
```

GTD parameter instance numbers start with 0 for the first instance. For example,

```
<assign name= "my_gtd" expr= "com.cisco.signal.gtdlist['setup_indication']"/>
<assign name= "pci1" expr= "my_gtd.PCI[0]"/> ==> for first PCI instance
<assign name= "pci2" expr= "my_gtd.PCI[1]"/> ==> for second PCI instance
```

Note If VoiceXML and JavaScript do not accept some special characters such as #, -, or + for field names, those field names must be provided within square brackets ([]). For example, to access the number (#) field of the first instance of the RGN parameter using JavaScript, it is specified as *var rgn_num= my_gtd.RGN[0]['#']*.

Creating a New GTD Message

The Cisco object class *com.cisco.objclass.gtd* is used to create GTD objects for read-write purposes. This object class is used to create a completely new GTD message or to create a copy of a GTD message received in a signaling event.

To create a new GTD message, use the format *new com.cisco.objclass.gtd()*.

Example:

<var name= "my_new_gtd" expr= "new com.cisco.objclass.gtd()"> creates a new empty GTD message. The message type for this GTD can be set later. For example,

<var name= "my_new_gtd.message_type" expr="IAM"/>. This GTD message is sent out using the <transfer> element.

Only GTD objects can be created; parameter objects cannot be created. For example,

<assign name="my_pci" expr="new PCI"/> is incorrect.

<assign name="my_pci" expr="my_gtd.PCI[0]"/> is correct (for the first instance).

Reading and Modifying GTD Parameters

To modify a GTD message, an instance is created using the *new* operator in *com.cisco.objclass.gtd*.

Examples

1. *<var name= "my_gtd" expr= "new com.cisco.objclass.gtd(com.cisco.signal.gtdlist['setup_indication'])"/>*

   This example creates a copy of the GTD message that came with the Setup Ind event on the incoming leg.

2. *<var name= "out_gtd" expr= "new com.cisco.objclass.gtd(gtd_xfer$.com.cisco.signal.gtdlist['alert_indication'])"/>*

   This example creates a copy of the GTD message from the <transfer> shadow variable *gtd_xfer$.com.cisco.signal.gtdlist* (assuming that gtd_xfer was the name of the transfer.) *out_gtd* contains a copy of the GTD parameters that arrived in the alert message on the outgoing leg. This is a read-write variable and is used to modify GTD parameters.

3. *<var name= "my_gtd" expr= "new com.cisco.objclass.gtd(com.cisco.signal.gtdlist['setup_indication'])"/>*

   This example creates a copy of the GTD message received in the setup message on the incoming leg which will be used to modify a GTD message.

   <assign name= "my_gtd.PCI[0].tri" expr="99"/> updates the *tri* field of the first instance of the PCI parameter of the GTD message represented by *my_gtd*.

```
<var name= "my_pci" expr= "my_gtd.PCI[0]"/> // first instance of PCI parameter
<var name= "pci_data" expr= "my_pci.dat"/>
<assign name= "my_pci.dat" expr= "pci_data+5"/>
```

   This example shows a VoiceXML script modifying a GTD message that arrived with a Setup Ind event.

   The same example is shown here using an ECMAScript:

```
<script>
var my_gtd= new com.cisco.objclass.gtd(com.cisco.signal.gtdlist["setup_indication"])
var my_pci= my_gtd.PCI[0]; // first instance of PCI parameter
var pci_data= my_pci.dat
my_pci.dat= pci_data+5;
<script/>
```

The following modifications of a GTD parameter or field are supported:

- Replace
- Append

Delete is not supported.

If the VoiceXML script adds an instance of a GTD parameter with the instance number being greater than the current instance count by one, the parameter is appended. If the instance number is less than or equal to the current instance count, the specified parameter is replaced.

Examples

If the number of instances currently existing in a GTD message is two, and:

- The VoiceXML script specifies *<assign name= "my_gtd[2].dat" expr= "'ABCDEF0123'"/>*, the GTD parameter is appended.
- The VoiceXML script specifies *<assign name= "my_gtd[1].dat" expr= "'ABCDEF0123'"/>*, the GTD parameter is replaced.
- The VoiceXML script specifies *<assign name= "my_gtd[4].dat" expr= "'ABCDEF0123'"/>*, an exception error *error.com.cisco.invalid.gtd.instance* is thrown because in this case only the instance numbers of 0 to 2 are valid, 0 and 1 for *replace* and 2 for *append*.

GTD Manipulation Sample Scripts

Here are some sample scripts you can use to get familiar with GTD Manipulation.

Accessing GTD parameters received during call setup.

```
<vxml version="2.0">
<form>
   <var name="X" expr="com.cisco.signal.gtdlist['setup_indication']"/>
    <var name="rgn1" expr="X.RGN[0].noa"/>
    <var name="rgn2" expr="X.RGN[0].npi"/>
    <var name="rgn3" expr="X.RGN[0].pi"/>
    <var name="x" expr="X.RGN[0]['#']"/>
```

```
    <var name="cgn1" expr="X.CGN[0].noa"/>
    <var name="cgn2" expr="X.CGN[0].npi"/>
    <var name="cgn3" expr="X.CGN[0].pi"/>
    <var name="cgn4" expr="X.CGN[0].si"/>
    <var name="y" expr="X.CGN[0]['#']"/>

    <var name="cpn1" expr="X.CPN[0].noa"/>
    <var name="cpn2" expr="X.CPN[0].npi"/>
    <var name="z" expr="X.CPN[0]['#']"/>

    <block>
        <log> x is:<value expr="x"/>:</log>
        <log> y is:<value expr="y"/>:</log>
        <log> z is:<value expr="z"/>:</log>
        <var name="rc" expr="'passed'"/>
        <if cond="x != '9876543210' &amp;&amp; y != '9876543210' &amp;&amp; z !=
'0123456789'">
            <assign name="rc" expr="'failed'"/>
        </if>
        <log> WB_FEAT_GTD_1001:END: <value expr="rc"/> </log>
        <disconnect/>
    </block>

    </form>
</vxml>
```

Accessing a Valid but Unavailable GTD Parameter

```
<form>
    <var name="X" expr="com.cisco.signal.gtdlist['setup_indication']"/>
    <var name="pcidat" expr="X.PCI[0].dat"/>
    <block>
    <if cond="pcidat== 'Undefined' || pcidat== 'undefined'">
          <log> Got expected value for GTD parameter PCI.dat as <value
expr="pcidat"/>:</log>
        <else/>
          <log> Got UNEXPECTED value for GTD parameter PCI.dat as <value
expr="pcidat"/>:</log>
          <assign name="test" expr="'failed'"/>
        </if>
        <goto next="#printresult"/>
    </block>
 </form>

  <form id="printresult">
     <block>
      <log> <value expr="testcase"/>:END: <value expr="test" /> </log>
     </block>
  </form>

</vxml>
```

Creating a GTD Message

```
<form>
    <var name="my_new_gtd0" expr="new com.cisco.objclass.gtd()"/>
    <var name="my_new_gtd1" expr="new com.cisco.objclass.gtd()"/>
    <var name="my_new_gtd2" expr="new com.cisco.objclass.gtd()"/>
    <var name="my_new_gtd3" expr="new com.cisco.objclass.gtd()"/>
    <var name = "my_new_gtd0.message_type" expr = "'IAM'"/>
    <var name = "my_new_gtd1.message_type" expr = "'ACM'"/>
    <var name = "my_new_gtd2.message_type" expr = "'CPG'"/>
```

```
    <var name = "my_new_gtd3.message_type" expr = "'REL'"/>
    <block>
           <log> message type for my_new_gtd0 is <value
expr="my_new_gtd0.message_type"/></log>
           <log> message type for my_new_gtd1 is <value
expr="my_new_gtd1.message_type"/></log>
           <log> message type for my_new_gtd2 is <value
expr="my_new_gtd2.message_type"/></log>
           <log> message type for my_new_gtd3 is <value
expr="my_new_gtd2.message_type"/></log>
    <goto next="#printresult"/>
    </block>
 </form>
<catch event="error.semantic">
    <assign name="res" expr="'failed'"/>
    <goto next="#printresult"/>
</catch>

  <form id="printresult">
     <block>
      <log> <value expr="testcase"/>:END:<value expr="res"/> </log>
     </block>
  </form>

</vxml>
```

Modifying Attributes in a GTD Message

```
<form>
    <var name="my_gtd0" expr="new com.cisco.objclass.gtd()"/>
    <var name = "my_gtd0.message_type" expr = "'IAM'" />
    <var name = "my_gtd0.RGN[0].npi" expr = "4" />
    <var name = "my_gtd0.RGN[0].pi" expr = "'y'" />
    <var name = "my_gtd0.RGN[0]['#']" expr = "'408-527-4800'" />

    <!-- Second gtd message use if needed -->

    <var name="my_gtd1" expr="new com.cisco.objclass.gtd()"/>
    <var name = "my_gtd1.message_type" expr = "'CPG'" />

    <block>
      <log> message type for my_gtd0 is <value expr="my_gtd0.message_type"/></log>
      <log> rgn.npi for my_gtd0 is <value expr="my_gtd0.RGN[0].npi"/></log>
      <log> rgn.pi for my_gtd0 is <value expr="my_gtd0.RGN[0].pi"/></log>
      <log> rgn.# for my_gtd0 is <value expr="my_gtd0.RGN[0]['#']"/></log>

     <if cond="my_gtd0.RGN[0].npi== '4' || my_gtd0.RGN[0].pi== 'y' || my_gtd0.RGN[0]['#']
== '408-527-4800'">
    <assign name="res" expr="'passed'"/>
        <else/>
        <log> Failed to modify new GTD instance.</log>
      </if>

      <goto next="#printresult"/>

    </block>

 </form>

  <form id="printresult">
     <block>
      <log> <value expr="testcase"/>:END:<value expr="res"/> </log>
     </block>
```

```
  </form>

</vxml>
```

Using <transfer> and <disconnect> for GTD Manipulation

The Cisco-specific attribute *cisco-gtd* is used with the <transfer> element to send GTDs on the outgoing call leg.

```
<transfer name="gtd_xfer" dest="tel: +1-555-555-0167" bridge="true"
!
!
cisco-gtd="my_gtd"
!
!
/>
```

It is also used as an attribute of the <disconnect> element to send GTDs during a call release.

```
<disconnect> cisco-disc_cause="cause_code"
                      cisco-gtd="my_gtd"
!
/>
```

User-to-User Information Manipulation

User-to-user information allows an external entity represented by a user-to-user information element (UUIE), to pass additional information for interaction with the Cisco IVR infrastructure. For example, a UUIE can pass "callerName=Joe; callerAccount=1234" which is processed by the ECMAScript and sent to the application server. A UUIE is sent in the UUS parameter of a GTD message through the <transfer> or <disconnect> elements. UUIE manipulation operations supported include *read*, *append*, and *replace*.

A VoiceXML script can access user-to-user information for ISDN and ISUP by reading the *dat* field of the UUS GTD parameter. The information can be set or modified by writing to the same *dat* field. This information is sent through the <transfer> and <disconnect> elements by specifying the GTD object that contains the information in the *cisco-gtd* attribute. If the GTD object specified in the attribute is invalid or null, an error event *error.com.cisco.invalid.gtd.object* is generated.

GTD Manipulation Error Events

Accessing a GTD parameter or its field returns an *undefined* value, and modifying a parameter or its field throws an error.semantic exception for the following errors in VoiceXML or JavaScript:

- Invalid GTD parameter or field name
- Invalid value of a GTD field
- Invalid instance that results from modifying a GTD parameter or field
- Invalid GTD message type

If the cisco-gtd attribute in <transfer> and <disconnect> does not translate to a valid GTD object, it is ignored with no error events are thrown, and the GTD message received from the incoming leg (in setup Indication) is sent to the outgoing leg.

If the disconnect cause code in <disconnect> is not translated to a positive integer, an error.semantic error event is thrown and VoiceXML document is terminated. No further validation (for example, checking the range) is performed on the disconnect cause code. It is the responsibility of the VoiceXML script to provide correct disconnect cause codes.

Redirecting Calls

Cisco IOS Release 12.3 allows a call to be redirected either through the CLI, or through a VoiceXML script using Cisco VoiceXML. These calls are redirected through the following two mechanisms:

- Release to Pivot: Redirecting Calls for ISUP
- Two B Channel Transfer: Redirecting Calls for ISDN

Release to Pivot: Redirecting Calls for ISUP

Call redirection using the Release-to-Pivot (RTPvt) mechanism can be invoked through the Cisco IOS CLI or through a VoiceXML script in Cisco VoiceXML. In Cisco VoiceXML, RTPvt is invoked through the <transfer> element with bridge= 'FALSE'. The call transfer modes are controlled through the Cisco root document property *com.cisco.transfer.mode*.

For more information on invoking RtPvt through the CLI, see "Configuring Telephony Call-Redirect Features" chapter of the *Cisco IOS Tcl IVR and VoiceXML Application Guide*.

Invoking Release-to-Pivot

In Cisco VoiceXML, the Release-to-Pivot (RTPvt) mechanism is invoked through the <transfer> element with bridge= 'FALSE'. The Cisco root document property *com.cisco.transfer.mode* with a specific value is used in the VoiceXML script to invoke RTPvt.

com.cisco.transfer.mode

The following transfer modes apply to RTPvt:

- Rotary— Gateway places a rotary call for outgoing call leg and hairpins the two calls together.
- Redirect_at_Connect—Call legs are connected by PSTN switch after the outgoing call leg is in *Connect* state.
- Redirect_Rotary—PSTN switch tries to directly connect the two call legs (redirect), otherwise, if that fails, call is hairpinned on the gateway (rotary).
- Redirect—PSTN switch directly connects the two call legs.

Sample Script

This sample script shows RTPvt invoked in VoiceXML version 2.0. The incoming GTD message contains RDC=3.

```
<

 <var name="res" expr="'failed'"/>
 <var name="testcase" expr="'ABC'"/>
 <var name="phone_num" expr="session.telephone.dnis" />
 <var name="mydur" />

<catch event="error.semantic" >
   <log> got the unexpected  error event </log>
   <goto next="#printresult"/>
</catch>
```

```
<property name="com.cisco.transfer.mode" value="redirect-rotary" />


    <form id = "transfer_me">
        <block>
           <audio src="en_welcome.au " />
        </block>

       <transfer name="mycall" destexpr="'tel: '+ phone_num"
                        connecttimeout="50s" bridge="false" maxtime="50s"
                        cisco-longpound="true">
         <filled>
              <assign name="mydur" expr="mycall$.duration"/>
             <log>The value in mycall is <value expr="mycall"/> </log>
             <log>Duration of call is <value expr="mydur"/></log>
         </filled>

         <catch event="telephone.disconnect.transfer" >
         <log> Call Transfered as expected </log>
         <var name="res" expr="'passed'"/>
         <goto next="#printresult"/>
        </catch>

     </transfer>
  </form>

   <form id="printresult">
    <block>
        <log> <value expr="testcase"/>:END:<value expr="res"/> </log>
        <disconnect/>
    </block>
  </form>

</vxml>
```

Two B Channel Transfer: Redirecting Calls for ISDN

Two B-Channel Transfer (TBCT) is an ISDN based call transfer feature that enables a Cisco voice gateway to perform call redirection through a VoiceXML script using Cisco VoiceXML. Call redirection using the Two B-Channel Transfer (TBCT) mechanism is invoked as a transfer initiator only for call setup. TBCT is invoked using VoiceXML blind transfer under the following conditions:

- Cisco VoiceXML root document property *com.cisco.transfer.mode* is enabled for the following transfer modes:
  - Redirect_Rotary
  - Redirect_at_XX , where XX represents *connect* or *alert*.
- The selected DS0s are within the same PRI line that has enabled TBCT, or the two PRI lines are configured for TBCT under the same trunk group.

If TBCT is invoked successfully, the call legs are eventually disconnected. If TBCT cannot be invoked because of an ISDN switch rejecting the TBCT request, the transfer falls back to a hairpinned call.

Invoking Two B-Channel Transfer

Two B-Channel Transfer (TBCT) for multiple trunk groups is invoked through a VoiceXML script using blind transfer under the following transfer modes:

redirect	Gateway redirects the call leg to the redirected destination number.
redirect-at-alert	Gateway places a new call to the redirected destination number and initiates a call transfer when the outgoing call leg is in the alerting state. If the call transfer is successful, the two call legs are disconnected on the gateway. If the transfer fails, the gateway bridges the two call legs. Supports TBCT.
redirect-at-connect	Gateway places a new call to the redirected destination number and initiates a call transfer when the outgoing call leg is in the connect state. If the call transfer is successful, the two call legs are disconnected on the gateway. If the transfer fails, the gateway bridges the two call legs. Supports TBCT.
redirect-rotary	Gateway redirects the call leg to the redirected destination number. If redirection fails, the gateway places a rotary call to the redirected destination number and hairpins the two call legs. For TBCT, this mode is the same as redirect-at-connect.
rotary	Gateway places a rotary call for the outgoing call leg and hairpins the two call legs. Call redirection is not invoked. This is the default.

To ensure successful TBCT for multiple trunk or trunkgroups that can reach the same destination, the voice gateway uses carrier sensitive routing to place an outgoing call from the router to the interfaces that are connected to the same trunk group as the incoming call.

To enable TBCT for multiple trunk or trunk groups:

- Configure the source carrier ID
- Configure the outbound dial peer. The target carrier ID is the same as the source carrier ID.
- Set the carrier ID in the VoiceXML script.

For information on configuring the carrier source ID and the outbound dial peers, see the *Cisco IOS Tcl and VoiceXML Application Guide*.

The following specific Cisco session variables and <transfer> attributes are used in a VoiceXML script to identify the carrier ID:

Session Variables

- com.cisco.carrierid.source —Defines the source ID for an incoming call.
- com.cisco.carrierid.target—Defines the target ID for an incoming call.

<transfer> attributes

- cisco-carrierid-source—Sets the source carrier ID for an outgoing call.
- cisco-carrierid-target—Sets the target carrier ID for an outgoing call.

Sample Script

This sample script shows a TBCT invoked for multiple trunk groups through a VoiceXML version 2.0 script using blind transfer.

```
<var name="res" expr="'failed'"/>
<var name="testcase" expr="'T_1_2'"/>
<var name="phone_num" expr="session.telephone.dnis"/>
<var name="source" expr="session.com.cisco.carrierid.source"
 <var name="mydur" />

<catch event="error.semantic" >
   <log> got the unexpected  error event </log>
   <goto next="#printresult"/>
</catch>

<property name="com.cisco.transfer.mode" value="redirect-at-alert"/>

<form id = "transfer_me">
<block>
    <audio src="flash:en_welcome.au" />
</block>

<transfer name="mycall" destexpr="'tel: '+ phone_num"
    connecttimeout="20s" bridge="false" maxtime="20s"
    cisco-longpound="true"
    cisco-carrierid-target="source">
         <filled>
              <assign name="mydur" expr="mycall$.duration"/>
             <log>The value in mycall is <value expr="mycall"/> </log>
             <log>Duration of call is <value expr="mydur"/></log>
        </filled>

       <catch event="telephone.disconnect.transfer" >
        <log> Call Transfered as expected </log>
        <var name="res" expr="'passed'"/>
         <goto next="#printresult"/>
        </catch>

     </transfer>
  </form>

   <form id="printresult">
    <block>
        <log> <value expr="testcase"/>:END:<value expr="res"/> </log>
       <disconnect/>
     </block>
  </form>

</vxml>
```

Blind Transfer Using SIP

Cisco VoiceXML can initiate a blind transfer using the Refer method in SIP. For more information on using the Refer method in SIP, see *SIP Call Transfer Enhancements Using the Refer Method*.

Disconnect Cause Code

Cisco VoiceXML allows a Cisco-specific disconnect cause code *cisco-disc_cause* to be specified as an attribute of the <disconnect> element. The cause code is specified as a value in the script. For cause code values, see the *Tcl IVR Version 2.0 Programmer's Guide*.

Example

```
<disconnect> cisco-disc_cause="cause_code"
cisco-gtd="gtd_object"
/>
!cisco-disc_cause and cisco-gtd are of type %expression.
```

Note

- If the GTD object specified in *cisco-gtd* is invalid or null, it will be ignored and no exceptions are thrown.
- If the cause code results in an integer, or if it is null or invalid (for example, an unknown variable), an *error.semantic* error event is thrown.
- The cause value can be specified in the cause code and the GTD object but they must both be consistent.

Hybrid Applications

Cisco IOS allows developers to use Tcl and VoiceXML scripts to develop hybrid applications. Tcl IVR 2.0 extensions allow Tcl applications to leverage support for ASR and TTS by invoking and managing VoiceXML dialogs within Tcl IVR scripts. This enables the implementation of hybrid applications using Tcl IVR for call control and VoiceXML for dialog management. VoiceXML is used for designing and conducting dialogs with the user over the telephone. For example, VoiceXML dialog is mainly used for IVR activities such as collecting user input, or playing prompts. However, it has limited capabilites for call control. Call control is the ability to receive, create, and manage new connections (call legs) with one or more users. Call control also allows advanced calling features to be implemented through the interconnection of these call legs. These advanced calling features create and manage multiple sessions with multiple users and simultaneously conduct dialogs with them. To allow implementation of these advanced calling features, the Cisco voice application infrastructure uses hybrid applications. Hybrid applications enable simple implementations of advanced call services that require ASR, TTS, or web integration.

Cisco hybrid applications use both Tcl IVR 2.0 and VoiceXML APIs that allow a developer to write only one application that runs and behaves like a single application instance. VoiceXML dialogs can be invoked through the Tcl IVR script. These VoiceXML dialogs are directed at any of the connected call legs or user sessions that are managed by the Tcl IVR application. The VoiceXML dialog code and the Tcl IVR script execute simultaneously, sharing control over the call leg and working together as one application.

The Tcl IVR script controls the call and all its call legs. It receives ev_setup_indication events for incoming call legs and issues leg alert or call leg acceptance commands through the leg connect Tcl IVR verb. The Tcl script can also create outgoing call legs and bridge multiple call legs. The Tcl script has two choices in conducting a dialog a with the user. It can use the existing leg collectdigits and media play Tcl IVR verbs to play individual audio prompts and collect digits, or it can use the leg vxmldialog Tcl IVR verb to initiate the VoiceXML dialog on the call leg. This verb starts a VoiceXML interpreter

session on the call leg under the direct control of the Tcl IVR script. The initial VoiceXML document can be embedded in a Tcl IVR script or it can be referenced by the Tcl script in form of a URI pointing to a VoiceXML document on a web server.

The VoiceXML session started on a call leg does not support the <transfer> element. In a hybrid application, calls are transferred in Tcl using the leg setup Tcl IVR command. The leg setup command requests the system to place a call to the specified destination number. To transfer a call, the VoiceXML dialog completes its execution, and passes control to the Tcl application to perform a leg setup for the call.

When the Tcl IVR script initiates a VoiceXML dialog on a call leg, it passes an array of parameters to the leg vxmldialog verb. The parameters are accessible in the VoiceXML session through the com.cisco.params.xxxx variable. In the VoiceXML session, the com.cisco.params object gets populated with information from the Tcl array where xxxx is the index of the array. After the VoiceXML dialog is complete, some information is returned to the Tcl IVR script throw the namelist attribute of the <exit> element. Upon completion of the VoiceXML dialog, the Tcl script receives a ev_vxmldialog_done event which contains the information returned in the <exit> element including a status code which can be accessed through the evt_status information tag.

The Tcl IVR script can also send intermediate messages to a VoiceXML dialog in progress through the leg vxmlsend Tcl IVR verb. The event specified in this verb is thrown inside the VoiceXML interpreter and caught by a <catch> handler looking for that event. The leg vxmlsend verb can also have a Tcl parameter array which is accessible inside the VoiceXML catch handler through the scoped variable *_message.params.xxxx* which is similar in function to the *com.cisco.params.xxxx* variable.

The VoiceXML interpreter environment or the VoiceXML document can also send events to the Tcl IVR script during various stages of execution. These events arrive at the Tcl script as *ev_vxmldialog_event* events. A VoiceXML dialog that is executing can use an <object> extension with classid= “builtin://com.cisco.ivrscript.sendevent” to send a specific message with associated parameter information to the parent Tcl script. If the VoiceXML dialog executes certain elements such as <transfer> or <disconnect> in the hybrid mode, the Tcl script receives an *ev_vxmldialog_event* event implicitly. An *ev_vxmldialog_done* event or an *ev_vxmldialog_event* event can arrive with the following information:

- The reason for receiving an *ev_vxmldialog_done* event or an *ev_vxmldialog_event* event is generated in a VoiceXML specific event name *vxml.** in the form of a string. These events are accessed through the *evt_vxmlevent* information tag in Tcl IVR 2.0. The string *vxml.** indicates an event name which can come from the VoiceXML interpreter environment (*vxml.session.**) or from the dialog executing in the interpreter (*vxml.dialog.**). Some of the messages coming from the VoiceXML interpreter environment are *vxml.session.complete* indicating normal completion of a dialog, or *vxml.session.transfer* indicating that the document tried to execute the <transfer> element which is not supported in hybrid mode.

  The *evt_vxmlevent* information tag contains a *vxml.dialog.** string, if:

  – The VoiceXML document throws an *error.badfetch* which is not caught, allowing the dialog to complete its execution, or

  – If the document uses the <object> element to send an explicit message to the Tcl script.

- A parameter array of information which is accessible through the *evt_vxmlevent_params* information tag.

Execution of the <disconnect> element in hybrid mode does not disconnect a call leg. A *vxml.session.disconnect* event is sent to the Tcl IVR script where a <disconnect> is emulated, the disconnect event is thrown, and the script continues execution. From this point onward, the VoiceXML dialog cannot play prompts or collect input. When the user hangs up, a <disconnect> is again emulated

and the leg stays connected. The Tcl script receives an *ev_disconnected* event and then must decide to either disconnect the leg immediately, or wait for the VoiceXML dialog to terminate and then disconnect the leg.

Execution of the <transfer> element in hybrid mode results in two events:

- A *vxml.session.transfer* event is sent by the VoiceXML environment to the Tcl script.
- The VoiceXML environment throws an *error.unsupported.transfer* event at the VoiceXML session which can be caught. If the event is not caught, the default handler causes the VoiceXML dialog to terminate, resulting in an *ev_vxmldialog_done* event thrown to the Tcl script.

Events and Errors

- error.unsupported.transfer

  The VoiceXML interpreter receives this event when the VoiceXML dialog executes a <transfer> element. The <transfer> element is not supported in hybrid applications.

sendevent Object

Recorded objects are represented as audio object variables in a VoiceXML script. In a Tcl script which is text based, they are represented in the form "*ram://xxxx"URI*. When information is sent from the Tcl to the VoiceXML script, Tcl array elements which have a value of "ram://xxxx" are available as audio variables or objects in the VoiceXML script. Similarly, when information is passed from a VoiceXML to a Tcl script, the audio variables or objects in the VoiceXML script will be available as Tcl array elements through the sendevent object in the form *"ram://xxxx"URI* in the Tcl script. The Tcl IVR script can play the audio object through the **media play** Tcl IVR verb. The sendevent object allows the VoiceXML session to send asynchronous events to the Tcl IVR script. The events are sent as parameters, in the form of strings, under the parameter *eventname* in the script shown below. The Tcl IVR script receives an *ev_vxmldialog_event* when the VoiceXML application invokes the sendevent object. The event name is available to the Tcl IVR script through the *evt_vxmldialog_event* infotag and is of the form *vxml.dialog.eventname*. The event carries the variables specified as *com.cisco.datatype.list* in the parameter name.

The sendevent object is accessed as follows:

```
<object>
   name="sendevent"
   classid="builtin://com.cisco.ivrscript.sendevent"
   <param name="eventname" expr="section6.stage5"/>
<param name="name1" expr="value1" valuetype= "com.cisco.datatype.list"/>
   <param name="name2" expr="value2" valuetype= "com.cisco.datatype.list"/>
....
   <param name="nameN" expr="valueN" valuetype= "com.cisco.datatype.list"/>
</object>
```

Table 1-18 Attributes and Parameters for the <sendevent> Object

Attribute or Parameter	Description
name (attribute)	The field item variable name that is filled with the results of the sendevent operation. It is an ECMAScript object containing the subobject ResultVariableName.status. ResultVariableName.status returns the status of the sendevent operation as a PASS or FAIL. **Note** For Cisco IOS Release 12.(2)11T, the status of the sendevent operation is always a PASS.
classid (attribute)	It invokes the "builtin://com.cisco.ivrscript.sendevent" command object resulting in an event being sent to the parent Tcl IVR script if the VoiceXML module is invoked as a subhandler by the Tcl IVR script.
eventname (parameter)	A mandatory parameter that specifies the name of the subevent sent to the Tcl IVR application. The subevent name is retrieved from the script using the evt_event_name infotag.
key 1 to key N (parameters)	• A set of optional parameters that allow the VoiceXML application to send parameters and the event to the Tcl IVR script. • The value of the attribute name in these parameters is available to the Tcl script through the evt_vxmlevent_params infotag. • The values of the attribute names become the indices of the Tcl array that contains the evt_vxmlevent_params infotag. • The group of attributes is recognized by the type attribute "array: com.cisco.aaa.vsa." • If an attribute name occurs multiple times, the indices available to the Tcl IVR script are of the form (Name, 0), (Name, 1), (Name,2), until the last applicable index.

Limitations and Restrictions

For limitations and restrictions on using Cisco VoiceXML, see the *Cisco IOS Tcl IVR and VoiceXML Application Guide*.

VoiceXML Document Loops

Cisco IOS VoiceXML provides safeguards against denial of service attacks that use infinite looping VoiceXML documents.

A maximum of ten loops are permitted per VoiceXML session to help prevent disruption of the system by a malicious looping program. Loops are counted when a VoiceXML document transits to another dialog within a document or goes to another document using <submit> or <goto> without any user interaction. If a document goes to another dialog or another document ten times without any prompts or digit collection, the session is aborted.

The loop count includes both before <disconnect> and after <disconnect> events. After <disconnect>, the VoiceXML document is mostly unrestricted if there is no user interaction.

Additional References

The following sections provide references related to Cisco VoiceXML.

Related Documents

Related Topic	Document Title
Tcl and IVR	*Cisco IOS Tcl IVR and VoiceXML Application Guide*
	Tcl IVR API Version 2.0 Programmer's Guide
	Enhanced Multi-Language Support for Cisco IOS Interactive Voice Response
Voice	Cisco IOS Voice Configuration Library
	Cisco IOS Voice Command Reference, Release 12.4T
Basic Configuration	*Cisco IOS Configuration Fundamentals Configuration Guide*
	Cisco IOS Configuration Fundamentals Command Reference, Release 12.4T
Security	*Configuring RADIUS*
	Configuring RADIUS with a Livingston Server
	RADIUS Vendor Specific Attributes Implementation Guide
	Authentication, Authorization, and Accounting: Cisco IOS Security Configuration Guide, Release 12.2
VoiceXML	*Cisco VoiceXML Solution Infrastructure*

Standards

Standard	Title
ECMA-262	*ECMAScript Specification 3.0* (*Standard ECMA-262, ECMAScript Language Specification*, 3rd edition, August 1998

RFCs

RFC	Title
RFC 2298	*An Extensible Message Format for Message Dispostion Notification*
RFC 1894	*An Extensible Message Format for Delivery Status Notifications*

Technical Assistance

Description	Link
The Cisco Technical Support & Documentation website contains thousands of pages of searchable technical content, including links to products, technologies, solutions, technical tips, tools, and technical documentation. Registered Cisco.com users can log in from this page to access even more content.	http://www.cisco.com/techsupport