Genesys Voice Platform 8.1

Legacy Genesys VoiceXML 2.1

Reference Manual

About Genesys

Genesys Telecommunications Laboratories, Inc., a subsidiary of Alcatel-Lucent, is 100% focused on software for contact centers. Genesys recognizes that better interactions drive better business and build company reputations. Customer service solutions from Genesys deliver on this promise for Global 2000 enterprises, government organizations, and telecommunications service providers across 80 countries, directing more than 100 million customer interactions every day. Sophisticated routing and reporting across voice, e-mail, and Web channels ensure that customers are quickly connected to the best available resource—the first time. Genesys offers solutions for customer service, help desks, order desks, collections, outbound telesales and service, and workforce management. Visit www.genesyslab.com for more information.

Each product has its own documentation for online viewing at the Genesys Technical Support website or on the Documentation Library DVD, which is available from Genesys upon request. For more information, contact your sales representative.

Notice

Although reasonable effort is made to ensure that the information in this document is complete and accurate at the time of release, Genesys Telecommunications Laboratories, Inc., cannot assume responsibility for any existing errors. Changes and/or corrections to the information contained in this document may be incorporated in future versions.

Your Responsibility for Your System's Security

You are responsible for the security of your system. Product administration to prevent unauthorized use is your responsibility. Your system administrator should read all documents provided with this product to fully understand the features available that reduce your risk of incurring charges for unlicensed use of Genesys products.

Trademarks

Genesys, the Genesys logo, and T-Server are registered trademarks of Genesys Telecommunications Laboratories, Inc. All other trademarks and trade names referred to in this document are the property of other companies. The Crystal monospace font is used by permission of Software Renovation Corporation, www.SoftwareRenovation.com.

Technical Support from VARs

If you have purchased support from a value-added reseller (VAR), please contact the VAR for technical support.

Technical Support from Genesys

If you have purchased support directly from Genesys, please contact Genesys Technical Support at the regional numbers provided on page 8. For complete contact information and procedures, refer to the ***Genesys Technical Support Guide***.

Ordering and Licensing Information

Complete information on ordering and licensing Genesys products can be found in the ***Genesys Licensing Guide.***

Released by

Genesys Telecommunications Laboratories, Inc. www.genesyslab.com

Document Version: 81gvp_ref_lgvxml_03-2010_v8.1.201.00

Table of Contents

Preface

Welcome to the *Genesys Voice Platform 8.1 Legacy Genesys VoiceXML 2.1 Reference Manual.* This manual describes the VoiceXML 2.1 language as implemented by the Genesys Voice Platform (GVP) in versions 7.6 and earlier, and which is now supported in the GVP 8.1 release. For information about the GVP Next Generation Interpreter (NGI), see the *Genesys Voice Platform 8.1 Genesys VoiceXML 2.1 Reference Help.*

This document is valid only for 8.1 release(s) of this product.

Note: For versions of this document created for other releases of this product, visit the Genesys Technical Support website, or request the Documentation Library DVD, which you can order by e-mail from Genesys Order Management at orderman@genesyslab.com.

This preface contains the following sections:

For information about related resources and about the conventions that are used in this document, see the supplementary material starting on page 117.

About GVP

GVP is a group of software components that constitute a robust, carrier-grade voice processing platform. GVP unifies voice and web technologies to provide a complete solution for customer self-service or assisted service.

In the Voice Platform Solution (VPS), GVP 8.1 is fully integrated with the Genesys Management Framework. GVP uses the Genesys Administrator, the standard Genesys configuration and management graphical user interface (GUI), to configure, tune, activate, and manage GVP processes and GVP voice and call control applications. GVP interacts with other Genesys components,

and it can be deployed in conjunction with other solutions, such as Enterprise Routing Solution (ERS), Network Routing Solution (NRS), and Network-based Contact Solution (NbCS).

Intended Audience

This document is primarily intended for voice application developers. It has been written with the assumption that you have a basic understanding of:

- Computer-telephony integration (CTI) concepts, processes, terminology, and applications
- Network design and operation
- Your own network configurations

You should also be familiar with HTML, XML, and VoiceXML concepts.

Making Comments on This Document

If you especially like or dislike anything about this document, feel free to e-mail your comments to Techpubs.webadmin@genesyslab.com.

You can comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this document. Please limit your comments to the scope of this document only and to the way in which the information is presented. Contact your Genesys Account Representative or Genesys Technical Support if you have suggestions about the product itself.

When you send us comments, you grant Genesys a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

Contacting Genesys Technical Support

If you have purchased support directly from Genesys, contact Genesys Technical Support at the following regional numbers:

Region	Telephone	E-Mail
North America and Latin America	+888-369-5555 (toll-free) +506-637-3900	support@genesyslab.com
Europe, Middle East, and Africa	+44-(0)-1276-45-7002	support@genesyslab.co.uk
Asia Pacific	+61-7-3368-6868	support@genesyslab.com.au
Malaysia	1-800-814-472 (toll-free) +61-7-3368-6868	support@genesyslab.com.au
India	1-800-407-436379 (toll-free) +91-(022)-3918-0537	support@genesyslab.com.au
Japan	+81-3-6361-8950	support@genesyslab.co.jp
Before contacting technical support, refer to the *Genesys Technical Support Guide* for complete contact information and procedures.		

Document Change History

This section lists topics that are new or that have changed significantly since the first release of this document.

GVP 8.1.2

- Information about Route/Port-based dialing has been added in Chapter 1.
- Additional information about Call Progress Analysis has been added in Chapter 2.
- New information about redirecting number, UUI data, and user data attached by SSG has been added in Chapter 2.
- The `afterconnecttimeout` attribute has been added in Chapter 3.
- Information about the AT&T out-of-band transfer has been added in Appendix A.

GVP 8.1.1

- The `telephonydata:put` and the `asr:freeResource` classids have been added to the object element extensions in Chapter 2.
- Information about the AT&T in-band transfer has been added in Appendix A, which is a new appendix.

Chapter

1 Overview

This chapter describes VoiceXML (Voice Extensible Markup Language) and provides an overview of the general platform architecture.

Note: This manual describes the VoiceXML 2.1 language as implemented by the Genesys Voice Platform (GVP) in versions 7.6 and earlier, and which is now supported in the GVP 8.1 release.

For information about the GVP Next Generation Interpreter (NGI), see the *Genesys Voice Platform 8.1 Genesys VoiceXML 2.1 Reference Help*.

This chapter covers the following topics:

Introducing VoiceXML

VoiceXML is a standard markup language used to develop voice applications. Voice applications use an HTTP browser and server model. The voice application host acts as the server and the telephony server acts as the browser that fetches and executes VoiceXML documents. VoiceXML provides a simple means for:

- Playing synthesized speech (text-to-speech) and audio files.
- Recognizing and recording spoken input.

- Recognizing Dual-Tone Multi-Frequency (DTMF) input.
- Controlling the flow of a call.

For more information about VoiceXML, refer to the *W3C Voice Extensible Markup Language (VoiceXML) 2.1, W3C Candidate Recommendation 13 June 2005*.

VoiceXML Platform Architecture

Figure 1 illustrates the architecture of a voice application. A customer calls a specified phone number; this call is answered at a VoiceXML gateway, and the request is passed to the Web Server.

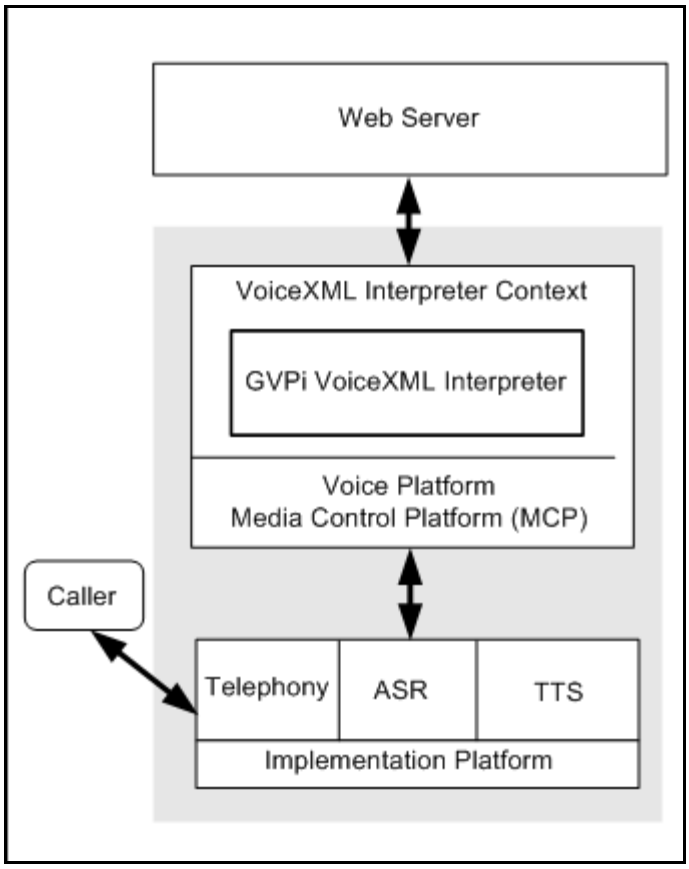

Figure 1: Platform for Voice Application

The client voice application, the VoiceXML Interpreter, sends requests to the Web Server through the VoiceXML Interpreter Context. The Web Server produces VoiceXML documents in reply. The VoiceXML Interpreter parses and executes the instructions in the VoiceXML document. For example, when the document indicates that user input is required, the Interpreter hands control over to a speech recognition engine that “hears” and interprets the spoken

response. The speech recognition component is entirely separate from the other components of the gateway.

The Interpreter Context works in conjunction with the Interpreter component. For example, Interpreter Context may listen for an escape phrase that will take the user to an agent or to another document in the voice application. The implementation platform is comprised of telephony, automatic speech recognition (ASR), and text-to-speech (TTS). These components generate events in response to caller actions (for example, spoken input received or caller disconnect) or system events (for example, timeout expiration), and the VoiceXML Interpreter or VoiceXML Interpreter Context acts on these events.

Supported Schemas

GVP supports the following schema files.

- VoiceXML 2.1 W3C R schema
- Genesys namespace (`http://www.genesyslab.com/vxml/2.0/ext/20020430`) schema
- SSML 1.0 R schema
- SRGS 1.0 R schema
- Telera namespace (`http://www.telera.com/vxml/2.0/ext/20020430`) schema

Platform Specifics

This section describes the enhancements in VoiceXML 2.1 and how GVPi supports them as well as how GVPi supports other features.

Refer to the *W3C Voice Extensible Markup Language (VoiceXML) 2.1, W3C Candidate Recommendation 13 June 2005,* for examples and sample scripts.

Table 1 lists the elements that have been introduced or enhanced in VoiceXML 2.1.

Table 1: New or Enhanced Elements in VoiceXML 2.1

Element	Purpose	New/Enhanced	GVP Support
<data>	Fetches arbitrary XML data from a document server.	New	Yes
<disconnect>	Disconnects a session.	Enhanced	Yes
<grammar>	References a speech recognition or DTMF grammar.	Enhanced	Yes

Table 1: New or Enhanced Elements in VoiceXML 2.1 (Continued)

Element	Purpose	New/Enhanced	GVP Support
<foreach>	Iterates through an ECMAScript array.	New	Yes
<mark>	Declares a bookmark in a sequence of prompts.	Enhanced	Yes
<property>	Controls platform settings.	Enhanced	Yes
<script>	References a document containing client-side ECMAScript.	Enhanced	Yes
<transfer>	Transfers the user to another destination.	Enhanced	Yes

Referencing Grammars Dynamically

A new attribute, `srcexpr`, is available in the `<grammar>` element of the VoiceXML application.

`srcexpr`—Equivalent to `src`, except that the URI is dynamically determined by evaluating the given ECMAScript expression in the current scope (for example, the current form item). The expression must be evaluated each time the grammar needs to be activated. If `srcexpr` cannot be evaluated, an `error.semantic` event is thrown.

Exactly one of `src`, `srcexpr`, or an inline grammar must be specified; otherwise, an `error.badfetch` event is thrown.

Referencing Scripts Dynamically

A new attribute, `srcexpr`, is available in the `<script>` element of the VoiceXML application.

`srcexpr`—Equivalent to `src`, except that the URI is dynamically determined by evaluating the given ECMAScript expression. The expression must be evaluated each time the script needs to be executed. If `srcexpr` cannot be evaluated, an `error.semantic` event is thrown.

Exactly one of `src`, `srcexpr`, or an inline script must be specified; otherwise, an `error.badfetch` event is thrown.

Concatenating Prompts Dynamically Using <foreach>

A new element, `<foreach>`, is available in VoiceXML 2.1.

`<foreach>`—Enables a VoiceXML application to iterate through an ECMAScript array and execute the content within the `<foreach>` element for each item in that array.

Table 2 shows the attributes.

Table 2: <foreach> Attributes

Attribute	Description
array	An ECMAScript expression that must evaluate to an array; otherwise, an `error.semantic` event is thrown.
item	The variable that stores each array item upon each iteration of the loop. A new variable will be declared if it is not already defined within the parent's scope.

Both `array` and `item` must be specified; otherwise, an `error.badfetch` event is thrown.

The `<foreach>` element can occur in executable content and as a child of `<prompt>`.

Example of <foreach> VoiceXML Element

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

  <script>
      function GetMovieList()
      {
        var movies = new Array(3);
        movies[0] = new Object();
        movies[0].audio = "godfather.wav"; movies[0].tts = "godfather";
        movies[1] = new Object();
        movies[1].audio = "high_fidelity.wav"; movies[1].tts = "high fidelity";
        movies[2] = new Object();
        movies[2].audio = "raiders.wav"; movies[2].tts = "raiders of the lost ark";

        return movies;
      }
  </script>

  <form id="pick_movie">

    <!--
    GetMovieList returns an array of objects
      with properties audio and tts.
      The size of the array is undetermined until runtime.
    -->
    <var name="prompts" expr="GetMovieList()"/>

    <field name="movie">
        <grammar xmlns="http://www.w3.org/2001/vxml"
            type="application/srgs+xml" xml:lang="en-US"
```

```
          version="1.0"  mode="voice" root="command">
        <rule id="command" scope="public">
          <one-of>
            <item> godfather </item>
            <item> high fidelty </item>
            <item> raiders of the lost ark </item>
          </one-of>
        </rule>
      </grammar>

      <prompt>
        <audio src="prelist.wav">Say the name of the movie from following list.</audio>
        <foreach item="thePrompt" array="prompts">
          <audio expr="thePrompt.audio"><value expr="thePrompt.tts"/></audio>
          <break time="300ms"/>
        </foreach>
      </prompt>

      <noinput>
        I'm sorry. I didn't hear you.
        <reprompt/>
      </noinput>

      <nomatch>
        I'm sorry. I didn't get that.
        <reprompt/>
      </nomatch>
      <filled>
        You said <value expr="movie"/>
      </filled>
     </field>
  </form>
</vxml>
```

Using <mark> to Detect Bargein During Prompt Playback

The Media Control Platform (MCP) is dependent on the partner MRCP TTS server's ability to support <mark> as described in Section 3.3.2 of the Speech Synthesis Markup Language (SSML) specification. The <mark> element places a marker into the text/element sequence. The MRCP TTS server must inform the interpreter when <mark> is executed during audio output.

GVP provides to the VoiceXML application the last <mark> element (if any) that was executed before bargein or the end of the prompt and the amount of time that had elapsed since the <mark> element.

GVP fully supports <mark> contingent upon the MRCP TTS servers support of sending MRCP SPEECH-MARKER events that are synchronized with the sending of the corresponding RTP packets for the TTS stream.

GVP supports using Nuance SWMS and NSS with RealSpeak for <mark>.

Note: RealSpeak 4.0/4.5 only supports <mark> with names that are unsigned 32-bit integers. The <mark> elements that do not meet this requirement are ignored by RealSpeak.

Recording User Utterances While Attempting Recognition

Several elements defined in VoiceXML can instruct the Interpreter to accept user input during execution. GVP has extended the <field>, <initial>, <link>, and <menu> elements to allow utterance recordings.VoiceXML 2.1 extends these elements so that the Interpreter can conditionally enable recording while simultaneously gathering input from the user. Support for utterance recordings in the <transfer> and <record> element is optional with VoiceXML 2.1, and is not supported by GVP.

To enable recording during recognition, set the value of the recordutterance property to true. If the recordutterance property is set to true in the current scope, the three shadow variables shown in Table 3 are set on the application.lastresult$ object whenever the application.lastresult$ object is assigned (for example, when a <link> is matched).

Table 3: Recordutterance-Related Shadow Variables

Shadow Variable	Description
recording	The variable that stores a reference to the recording or is undefined if no audio is collected. Like the input item variable associated with a <record> element, as described in VoiceXML specification.
recordingsize	The size of the recording in bytes, or undefined if no audio is collected.
recordingduration	The duration of the recording in milliseconds, or undefined if no audio is collected.

When these properties are set on the application.lastresult$ object, if an input item (as defined in the VoiceXML specification) has also been filled and has its shadow variables assigned, the Interpreter also assigns recording, recordingsize, and recordingduration shadow variables for these input items; the values of these equal the corresponding properties of the application.lastresult$ object. For example, in the case of <link> and <menu>, since no input item has its shadow variables set, the Interpreter sets only the application.lastresult$ properties. Like recordings created using the <record> element, utterance recordings can be played back using the expr attribute on <audio>.

User utterances can be recorded when there is a recognition active. This implies that during <transfer>, if the recognition is turned on, user utterances can be recorded.

Like recordings created using the <record> element, utterance recordings can be submitted to a document server via HTTP POST, using the namelist attribute of the <submit> and <subdialog> elements. The enctype attribute must be set to multipart/form-data, and the method attribute must be set to post. To provide flexibility in the naming of the variable that is submitted to the document server, the Interpreter also enables the utterance recording to be assigned to and, posted via, any valid ECMAScript variable.

Note: During execution of the <submit> element, GVP uses multipart-formdata encoding type for POSTing recording/user utterance variables.

The user utterance recording is supported for MRCP ASR only.

Posting User Utterances

User utterances can be posted to the VoiceXML application during <submit> or <subdialog>. When the Interpreter encounters either of these two elements, utterances for all of the recognitions that have occurred so far will be posted. For example, if three recognitions have occurred since the last <submit>/<subdialog>, all three utterances will be posted to the application during current <submit>/<subdialog>.

Posting of utterances is a two step process:

1. The Interpreter fetches all of the captured utterances from the ASR server to MCP.
2. The Interpreter posts it to the URL specified by the <submit>/<subdialog> elements.

After each recognition, the Interpreter downloads the captured utterance from the ASR server to the MCP. During <submit> or <subdialog> execution, the Interpreter packages all of the utterances and any other fields (including the regular <record>, for example) into one multipart/form-data package, and posts it to the VoiceXML application.

Specifying the Media Format of Utterance Recordings

The Legacy GVP Interpreter (GVPi) does not support the `recordutterancetype` property. GVP defaults the `recordutterancetype` property to the ASR vendor's default recording type.

Note: The `recordutterancetype` property does not affect the `<record>` element.

Interacting with Existing Controls

Under IVR profile, `gvp.service-parameters, voicexml.gvpi.$asrwavfilelog$` instructs the MCP to turn on/off utterance capture (no VoiceXML application intervention is necessary).

This control applies to utterance captures on MRCP ASR.

Table 4 shows how `recordutterance` property and `$asrwavfilelog$` interact.

Table 4: Interaction of recordutterance and $asrwavfilelog$

recordutterance	$asrwavfilelog	Comments
true	true	Utterance recordings will be available.
true	false	Utterance recordings will not be available.
false	true	
false	false	

If the application turns on the utterance recording (via `recordutterance`), but the recording does not occur because of the IVR Profile controls, the value of the recording variable is undefined (see Table 3 on page 17).

Adding namelist to <disconnect>

A new attribute, `namelist`, is available in the `<disconnect>` element of the VoiceXML application:

`namelist`—Specifies the variable names to be returned to the Interpreter context. The default is to return no variables; this means that the Interpreter context receives an empty ECMAScript object. If an undeclared variable is referenced in the namelist, an `error.semantic` is thrown (refer to the VoiceXML 2.1 specification)

The namelist variables are available to the Interpreter context as part of the `session.genesys.disconnect` object, and each variable can be accessed as `session.genesys.disconnect.<varname>`, where `<varname>` can be any variable name specified on the namelist.

Adding type to <transfer>

VoiceXML 2.1 extends the `<transfer>` element to support the following additional attribute:

`type`—The type of transfer. The value can be `bridge`, `blind`, or `consultation`.

GVP supports all of the preceding transfer types.

Note: In VoiceXML 2.0, `<transfer>` has a bridge attribute. With VoiceXML 2.1, exactly one of bridge or type can be specified; otherwise an `error.badfetch` event is thrown.

It is common for switch vendors to describe a *consultative* transfer using different terms, such as, *whisper* transfer, *screen* transfer, or *consultative* transfer to mean essentially the same concept—the extension currently talking with an incoming caller can invoke a transfer request to the switching system, provide a new target extension, talk with the person who answers that extension number (an agent) and then hang-up. The caller is connected to the agent. GVP can support this using internal or external bridging methods. To accomplish a screen transfer as described, the developer must use Genesys TXML if they want the GVP platform to dial out to the agent and play some information before bridging the caller and agent together while GVP stops listening (internal bridging) or releasing the call entirely where the bridge is external.

The function consultative transfer, defined in VoiceXML standards, is much more restricted and takes on a narrower meaning.

VoiceXML standards define consultation transfer as similar to blind transfer except that the outcome of the transfer call setup is known and the caller is not dropped as a result of an unsuccessful transfer attempt. In this sense, VoiceXML consultative transfer is about a bad outcome, not a means of talking to the target phone first. When the browser responds to a VoiceXML consultative transfer, if the call fails from an unsuccessful transfer attempt, GVPi will return an error response, which is an exception toward the VoiceXML application. It is the responsibility of the application programmer to recover from the exception if received within the application, such as trying again.

GVPi supports VoiceXML consultative transfer requests over IP SIP using the consultative transfer tag for SIP `REFER with Replaces` configuration.

If the value of the `type` attribute is set to `bridge`, the Interpreter's behavior is identical to its behavior when the value of the `bridge` attribute is set to `true`. If the value of the `type` attribute is set to `blind`, the Interpreter's behavior is identical to its behavior when the `bridge` attribute is set to `false`. The behavior of the `bridge` attribute is detailed in the VoiceXML specification.

The `bridge` attribute is maintained for backward compatibility with VoiceXML 2.0. Since all of the functionality of the `bridge` attribute has been incorporated

into the type attribute, VoiceXML application developers are encouraged to use the type attribute.

The connecttimeout attribute of <transfer> applies if the type attribute is set to bridge or consultation. The maxtime attribute of <transfer> applies if the type attribute is set to bridge.

Using Route/Port Based Dialing

When used with the GVP PSTN Connector, GVPi enables route/port based dialing. GVPi will allow the value of the destination parameter to:

- ROUTE:X as a prefix to the actual dest number, where X is the route number as required by the PSTN Connector configuration.
- PORTNUM:X, where X is the port number as defined in the PSTN Connector configuration.
- PORTDLGC:X-Y, where X=Dialogic Card ID and Y=Dialogic Port ID as defined in the PSTN Connector configuration.
- AUTHCODE:X, where X=Authorization code to outpulse before the call is connected.

Example

```
<transfer name="mycall" dest="ROUTE:1:sip:000916504664649@10.10.30.5" type="bridge">
```

Accessing Additional Properties from ASR Results

If the VoiceXML application is required to access additional properties from ASR results, use the com.genesys.accessasrresultproperties property. By default, this property is set to false. Advanced application developers can enable it to make use of ASR vendor-specific properties passed back from third-party ASR servers.

Example

```
<result>
  <interpretation grammar="session:CNGRAMMAREXPRESSION1_grammar21" confidence="29">
    <input mode="speech">New York New York</input>
      <instance>
        <city confidence="29">NYC</city>
        <state confidence="70">New York</state>
      </instance>
  </interpretation>
</result>
```

In the above grammar example, if com.genesys.accessasrresultproperties is set to false, the application.lastresult$.interpretation would populate as:

application.lastresult$.interpretation is an object

`application.lastresult$.interpretation.city` = `NYC`

`application.lastresult$.interpretation.state` = `New York`

This is equivalent to the platform behavior prior to GVP 7.6 before the `com.genesys.accessasrresultproperties` property was introduced.

On the other hand, if `com.genesys.accessasrresultproperties` is set to `true`, additional result properties are exposed and the `application.lastresult$.interpretation` would populate as:

`application.lastresult$.interpretation` is an object

`application.lastresult$.interpretation.city` is an object

`application.lastresult$.interpretation.city.$` = `NYC`

`application.lastresult$.interpretation.city.confidence` = `29`

`application.lastresult$.interpretation.state` is an object

`application.lastresult$.interpretation.state.$` = `New York`

`application.lastresult$.interpretation.state.confidence` = `70`

The `$` property holds the text value of interpretation top-level properties like city and state when sub-properties such as confidence exist. Otherwise, without sub-properties like confidence, the text value would be assigned directly to the top-level property without the need for the `$` property. This behavior is ASR vendor-specific, and dependent on the results received for a given grammar. A VoiceXML application can be written to be ASR vendor-independent by verifying that the property is an object, and by accessing the `$` property accordingly.

Example

```
<var name="result" expr="application.lastresult$.interpretation.city" />
  <if cond="typeof(result)=='object'" >
    <assign name="result" expr="application.lastresult$.interpretation.city.$"/>
  </if>
```

VoiceXML Properties

This section provides a comprehensive list of properties defined by VoiceXML 2.1 that can be specified through the `<property>` element in GVP. For each property in VoiceXML, the tables specify whether GVP supports the property. If the property is supported, the default value is provided.

For detailed descriptions of the properties, refer to the appropriate W3C VoiceXML specification. The VoiceXML 2.1 specification only describes the features that are in addition to VoiceXML 2.0. Refer to the VoiceXML 2.0 specification for the base 2.0 features.

References

VoiceXML version 2.0, W3C Recommendation, 16 March 2004
VoiceXML version 2.1, W3C Candidate Recommendation 13 June 2005

VoiceXML 2.0

Table 5: Generic Speech Recognizer Properties

Property	Supported	Default
confidencelevel	yes	0.5
sensitivity	yes	0.5
speedvsaccuracy	yes	0.5
completetimeout	yes	1s
incompletetimeout	yes	1s
maxspeechtimeout	yes	20s

Table 6: Fetching Properties

Property	Supported	Default
fetchaudio	yes	n/a
fetchtimeout	yes	3s
fetchaudiodelay	yes	0s
fetchaudiominimum	yes	0s
audiofetchhint	yes	prefetch
audiomaxage	yes	-1s
audiomaxstale	yes	-1s
documentfetchhint	yes	safe
documentmaxage	yes	-1s
documentmaxstale	yes	-1s
grammarfetchhint	yes	safe
grammarmaxage	yes	-1s

Table 6: Fetching Properties (Continued)

Property	Supported	Default
grammarmaxstale	yes	-1s
objectfetchhint	yes	safe
objectmaxage	yes	-1s
objectmaxstale	yes	-1s
scriptfetchhint	yes	safe
scriptmaxage	yes	-1s
scriptmaxstale	yes	-1s

Table 7: Miscellaneous Properties

Property	Supported	Default
inputmodes	yes	"voice dtmf"
maxnbest	yes	1
universals	yes	"none"

Table 8: Prompt and Collect Properties

Property	Supported	Default
bargein	yes	true
bargeintype	yes	speech
timeout	yes	0s

Table 9: Generic DTMF Recognizer Properties

Property	Supported	Default
interdigittimeout	yes	1s
termtimeout	yes	0s
termchar	yes	#

VoiceXML 2.1

Table 10: <data> Fetching Properties

Property	Supported	Default
datafetchhint	yes	-1s
datamaxage	yes	-1s
datamaxstale	yes	-1s

Table 11: Recording User Utterances While Attempting Recognition

Property	Supported	Default
recordutterance	yes	false
recordutterancetype **Note:** The type of recording provided is dependent on the MRCP ASR server vendor. Consult with the vendor regarding which types are supported and whether this property is configurable on the vendor's MRCP server.	no	

Support Notes

Refer to the *Genesys Voice Platform 8.1 Application Migration Guide* for detailed information about migrating applications from earlier GVP releases.

<tag>

GVP does not support the use of `<tag>` inside a Speech Recognition Grammar Specification (SRGS) grammar when ASR is disabled in the VoiceXML application.

<data>

The `<data>` element enables a VoiceXML application to fetch arbitrary XML data from a document server without transitioning to a new VoiceXML document. The XML data fetched by the `<data>` element is bound to

ECMAScript through the named variable that exposes a read-only subset of the W3C Document Object Model (DOM).

GVP fully supports the <data> element. GVP does not support additional data formats by recognizing additional media types (this is optional as per VoiceXML 2.1).

GVP fully supports the <data> fetching properties.

GVP does not support the <?access-control?> processing instruction.

<transfer>

For the <transfer> element, a negative value for the connecttimeout attribute will be ignored and will not result in an error.

Grammars

- If an inline grammar contains <meta> or <metadata>, it is ignored. If an inline grammar (or an external grammar without ASR in use) contains only metadata and does not refer to any rules, an error.badfetch will be thrown. Support of <meta> or <metadata> in an external grammar is subject to support by the third-party ASR server. Consult with your ASR vendor.
- Inline grammars that are missing the version attribute will not throw "error.badfetch". Instead, GVPi will assume that the version is 1.0.
- Support of the weight attribute in the <grammar> element is dependent on support by the ASR server. Do not use the weight attribute in the VoiceXML grammars if the ASR servers that are being used do not support it.

ASR Grammars

- SRGS support is via a third-party MRCP ASR server. ABNF support is on the IBM WVS only.
- GVPi does not support recognition of voice grammars while <record> is active. This is optional as per the VoiceXML 2.0 specification.

DTMF Grammars

SRGS v1.0 support is via a third-party MRCP ASR server for ASR applications. For non-ASR applications, GVP provides native SRGS grammar support. Native GVP does not provide support for optional ABNF format or optional semantic interpretation.

Root Context

The root context is not initialized when a <submit> element causes the leaf to root transition. A leaf to root transition within the same application occurs when the current document is a leaf document and the target document's absolute URI is the same as the name of the current application.

Builtin Types

If the specified builtin type is not supported, GVP will throw `"error.badfetch.grammar.uri"` instead of `"error.unsupported.builtin"`.

Event Counts

Event counts within form items in the same form are not reset. They are carried over to the next form item until the form is re-entered.

Chapter

2 Platform Extensions

The Genesys Voice Platform (GVP) provides enhanced telephony features that you can utilize in a voice application. This chapter describes Genesys Voice Platform extensions to VoiceXML and is divided into the following sections.

Note: This manual describes the VoiceXML 2.1 language as implemented by GVP in versions 7.6 and earlier, and which is now supported in the GVP 8.1 release. For information about the GVP Next Generation Interpreter (NGI), see the *Genesys Voice Platform 8.1 Genesys VoiceXML 2.1 Reference Help*.

Platform Extensions to VoiceXML

VoiceXML platform extensions consist of:

- Element Extensions
- Property Extensions
- Error Extensions
- Platform-Specific Properties

Element Extensions

The element extensions provide the following functionality:

- Playing Dynamic Data
- Playing DTMF (Dual-Tone Multi-Frequency) Tones
- Call Progress Analysis (CPA)

- `Append` attribute in `<param>` element
- Telera Namespace Information
- Genesys Namespace Information
- Posting `<log>` element Information

Playing Dynamic Data

In a voice application, there are two methods for playing information provided by a backend database system (for example, the bank account of a person) to a caller. One method is to use speech synthesis using text-to-speech elements; the other method is to use snippets of recorded `.vox` files that are selected using a JavaScript script. Given the current limitations in speech synthesis technology, the caller experience may be improved when the voice application uses pre-recorded snippets rather than speech synthesis.

The VoiceXML standard does not support the use of prerecorded `.vox` files to play currency, dates, days, numbers, digits, letters, or ordinals. You can use the enhanced `<audio>` element to play an array of `.vox` files that have been selected by a JavaScript script. The Telera namespace `xmlns` is required when you use this extension. Here is an example:

```
<script src='Languages/en-US/PlayBuiltinType.js'/>
<audio expr='PlayBuiltinType("3775","number");'
xmlns='http://www.telera.com/vxml/2.0/ext/20020430'/>
<audio expr='PlayBuiltinType("41433p","time");'
xmlns='http://www.telera.com/vxml/2.0/ext/20020430'/>
<audio expr='PlayBuiltinType("20012711","date","f.SPEAK_YEAR |
f.SPEAK_DAY | f.SPEAK_MONTH");'
xmlns='http://www.telera.com/vxml/2.0/ext/20020430'/>
<audio expr='PlayBuiltinType("4532.99","currency");'
xmlns='http://www.telera.com/vxml/2.0/ext/20020430'/>
<audio expr='PlayBuiltinType("21","ordinal");'
xmlns='http://www.telera.com/vxml/2.0/ext/20020430'/>
<audio expr='PlayBuiltinType("IBM 200","alphanumeric");'
xmlns='http://www.telera.com/vxml/2.0/ext/20020430'/>
```

Playing DTMF Tones

The `<value>` element has been enhanced to play DTMF tones. The Telera namespace `xmlns` is required when you use this extension. Here is an example:

```
<value mode="dtmfplay" expr="'411#'"
xmlns='http://www.telera.com/vxml/2.0/ext/20020430'/>
```

Call Progress Analysis

Call Progress Analysis (CPA) determines the result of an outbound call.

When using the PSTN Connector as the gateway, this feature depends on the trunk card vendor—for example, Dialogic—to supply the appropriate CPA detection, as well as the MCP's ability to support Call Progress Detection (CPD).

For VoIP, this feature depends on the VoIP Media Gateway having the appropriate signaling message, which it obtains by performing the CPA detections. Far End Busy support is dependent on the far end providing the appropriate SIP messages. The CPA is done pre-connect, before the call is answered and is dependent on appropriate SIP protocol messages being provided for CPA.

Genesys Voice Platform supports the CPA features listed in Table 12.

The return result for CPA is through a dollar variable called `$_cparesult$`. Table 12 lists the various values:

Table 12: Values of CPA Features

CPA Feature	$cparesult$	VoIP	TDM
Normal Answer	CPA_NORMAL	✓	✓
Ring No Answer	CPA_NOANSWER	✓	✓
Far End Busy	CPA_BUSY	✓	✓
Answering Machine	CPA_ANSWERMACHINE	✓	✓
Fax/Modem	CPA_FAXMODEM	✓	✓
Operator/Network Intercept	CPA_OPERATORINTERCEPT		✓
No Ringback	CPA_NO_RINGBACK		✓
Not In Service	CPA_NOT_INSERVICE		✓
No Dialtone	CPA_NODIALTONE		✓
Unallocated Number	CPA_UNALLOCATED		✓
Vacant Circuit	CPA_VACANT_CIRCUIT		✓
SIT Unknown	CPA_SIT_UNKNOWN		✓

To support CPA, use the `AFTERCONNECTTIMEOUT` attribute (value is in seconds) in the Genesys extension to the VoiceXML `<transfer>` element (Example 1), and use the `AFTERCONNECTTIMEOUT` attribute for the TXML `<CREATE_LEG_AND_DIAL>` element (Example 2). See page 59 for information about this attribute.

Example 1

```
<genesys:transfer name="mycall" dest="sip:abc@10.10.30.5" connecttimeout="60s"
   type="bridge" afterconnecttimeout="15s">
```

Example 2

```
<CREATE_LEG_AND_DIAL AFTERCONNECTTIMEOUT="20" .../>
```

Post-Connect CPA

GVPi supports post-connect CPA, on both `<transfer type="bridge"/>` and TXML bridge transfer applications, with the Genesys Media Server and the GVP PSTN Connector. To achieve this using either VoiceXML or TXML, configure the following settings:

VoiceXML

1. Set the `<transfer>` element with `bridge=true` or `type=bridge`.
2. Configure the IVR Profile with the following options:
   - `transfer type`—`2SignalChannel` or unspecified
   - `transfer option`—unspecified

The CPA result will be captured in the `$_cparesult$` variable and made available to the VoiceXML application as a `session.genesys._cparesult` session variable.

The CPA result will also be available immediately after the `<transfer>` element is executed, and will appear as a value of the `<transfer> formitem` variable.

TXML

1. Set the `<CREATE_LEG_AND_DIAL>` element with `bridge=yes|no`.
2. Configure the IVR Profile with the following options:
   - `transfer type`—`2SignalChannel` or unspecified
   - `transfer option`—unspecified

When `bridge=yes`, the CPA result will be made available in the `$_cparesult$` variable immediately after executing the `<CREATE_LEG_AND_DIAL>` element.

When `bridge=no`, the CPA result will be made available in the `$_cparesult$` variable immediately after the dial out request and before executing the IVR URL of the second leg. This result is available only on the second leg.

Telera Namespace Information

To use the Telera namespace for extended VoiceXML elements, use the namespace value `http://www.telera.com/vxml/2.0/ext/20020430`.

For information about using the Telera namespace, refer to the extension of <submit>, <value>, or <audio> (<audio> applies to single-tenant only) elements.

Note: The namespace that GVPi defines is used solely as a unique identifier, which is consistent with the W3C specification.

Genesys Namespace Information

To use the Genesys namespace for extended VoiceXML elements, use the namespace http://www.genesyslab.com/vxml/2.0/ext/20020430.

All new platform extensions will be added to the Genesys namespace.

Note: The namespace that GVPi defines is used solely as a unique identifier, which is consistent with the W3C specification.

Posting <log> Element Information

A new attribute, posturl, has been added to the <log> element so that the text within the <log> element can be posted to the URL that is specified by the posturl attribute.

Example

```
<log posturl="http://.../capture.asp">This is a post text from log element</log>
```

Property Extensions

The Genesys Voice Platform $ variables are accessible inside VoiceXML applications as properties. All platform $ variables are read-only. Variables containing a hyphen (-) cannot be included in a VoiceXML page, because the ECMA Script interprets a hyphen as a minus sign. In order to get the value of a $ variable containing a hyphen from the list below, you can pass it as a query string parameter as shown in "Example 2" on page 36.

Table 13 lists the platform-specific variables.

Table 13: Platform-Specific Variables

Variable	Description
did	The called telephone number that the incoming call came in on.
ani	The caller's telephone number (may not be available in certain cases).
$sessionid$	A unique identifier for this call generated by the platform.

Table 13: Platform-Specific Variables (Continued)

Variable	Description
$ivr-root-dir$	The URL of the voice application's root directory (not including web scripts and the query string). For example, its value may be `http://www.mycompany.com/teleb/ivr`, assuming that all the web scripts and xml pages for the voice application are under the `teleb/ivr` on the `www.mycompany.com` website.
$ivr2-root-dir$	The URL of the backup voice application's root directory (not including web scripts and the query string).
$ivr-url$	The URL of the voice application's main web script (but not including any query strings).
$ivr2-url$	The URL of the backup voice application's main web script (but not including any query strings).
$start-ivr-url$	The URL of the voice application's main web script (including any query strings).
$last-error$	The last error encountered by the MCP. Possible values are: • `BAD_XMLPAGE` • `XMLPAGE_NOTFOUND` • `XMLPAGE_TIMEOUT` • `RESOURCE_NOTFOUND` • `RESOURCE_TIMEOUT` • `BAD_XMLTAGNAME` • `BAD_XMLTAGVALUE` • `OPSERVER_ERROR` The MCP clears this variable after the next error-free operation.
$last-error-url$	The URL of the voice application's XML page or resource (`.vox`) that caused the last error.
$last-error-string$	A free format string filled in by the MCP code to give diagnostic information regarding the last error.
$toll-free-num$	The phone number that was dialed by the caller to make this call.
$ccerror-telnum$	The outbound telephone number that the MCP code dials if GVPi and voice applications are not accessible (because of a problem with the data network).
$callerhup$	The MCP code sets this variable to true if the caller terminated the call. The voice application typically uses this variable to find out whether the user terminated the call after entering some information in the middle of a form.

Table 13: Platform-Specific Variables (Continued)

Variable	Description
$_cparesult$	The MCP code sets this variable after a `CREATE_LEG_AND_DIAL` request when the outbound leg has dialed the number. It contains the result of the call progress analysis performed by the platform. Possible values are: • `CPA_NORMAL` • `CPA_BUSY` • `CPA_NOANSWER` This indicates that an outbound call was initiated but the far end did not answer the call. • `CPA_ANSWERMACHINE` • `CPA_FAXMODEM` • `CPA_OPERATORINTERCEPT` • `CPA_NO_RINGBACK` • `CPA_NOT_INSERVICE` • `CPA_NODIALTONE` • `CPA_UNALLOCATED` • `CPA_VACANT_CURCUIT` • `CPA_SIT_UNKNOWN`
sid	This variable contains the value of the ScriptID as generated by the CTI Connector to handle call treatments.
$scriptdata$	This variable contains any data specific to the ScriptID.
$scripturl$	This is invoked by GVPi after it receives a request to Play Treatment.
$playfilesize$	This predefined variable for supporting unified messaging sets the `playfilesize` after each play. When the caller interrupts the play with the tone input, the voice application knows exactly when the play was interrupted by including the predefined variable in the URL.
$recordfilesize$	This predefined variable for supporting unified messaging sets the record filesize after each recording. The voice application can include the predefined variable in the URL as a querystring.

Table 13: Platform-Specific Variables (Continued)

Variable	Description
$badxmlpageposturl$	The voice application can set this variable to point to a URL (within the application) where the teleserver, upon encountering an XML page with syntax errors, posts the bad XML page.
$dialed-number$	GVPi sets this variable when it dials the outbound number. The voice application can access this variable to find the number being dialed. The trigger to dial out can be from the voice application using the TXML `CREATE_LEG_AND_DIAL` element, the VoiceXML `transfer` element, the TXML `QUEUE_CALL` element, or Outbound notification service. If there is an error, GVP transfers to `$ccerror-telnum$`.

Example 1

Traditional dollar variables can be used only in URLs:

```
<vxml ...>
   <form>
      <block>
         <goto next="$ivr-root-dir$/mystart.asp" />
      <clock>
   </form>
</vxml>
```

Example 2

If a dollar variable has to be accessed in the scripting engine, the variable has to passed through server-side scripting, such as ASP:

VoiceXML document #1

```
<vxml ...>
   <form>
      <block>
         <goto next="page2.asp?IvrRootDir=$ivr-root-dir$" />
      </block>
   </form>
<vxml>
```

VoiceXML document #2—page2.asp:

```
<vxml ...>
   <script>
      var a = <%=Request( "IvrRootDir" ) %>;
      ...
      if ( a == "..." )
      {
         ...
      }
   </script>
</vxml>
```

Error Extensions

The following ECMA script variables are available when an error event is thrown:

- `telera.error.name`

  Specifies the name of the error from a list of generic errors.

- `telera.error.description`

  A detailed description of the error that occurred.

- `telera.error.currenturl`

  Specifies the URL of the page on which the error occurred.

- `telera.error.element`

  Specifies the identity of the element in which the error occurred. If this is not available, the type of element is specified instead. For example, the `<prompt>` element has no `id` attribute; therefore, the `telera.error.element` would specify `<prompt>`.

The following events are thrown by the platform:

- `error.com.telera.createleg`

  Occurs for a failure condition in the `CREATE_LEG_AND_DIAL` call control element.

- `error.com.telera.dial`

  Occurs when there is a dial error during the `CREATE_LEG_AND_DIAL` call control element.

- `error.com.telera.bridge`

  Occurs for a failure to bridge after issuing the `<BRIDGE/>` call control element.

- `error.com.telera.unbridge`

  Occurs when there is a failure to unbridge after issuing the `<UNBRIDGE/>` call control element.

- `error.com.telera.queue`

  Occurs for all URS communication and for interaction failures due to any one of the three QUEUE control elements.

Platform-Specific Properties

The following platform-specific properties are available:

- `com.telera.speechenabled`

  Use this property to define whether an ASR engine is used in the voice application. The value is either `true` or `false`. The default is `true`.

- `com.telera.audioformat`

Use the property `com.telera.audioformat` of the `<property>` element to specify the audio format for the audio file. The default is `audio/basic`.

These values are available in VoiceXML on the MCP:

Mu Law .vox formats:
audio/x-mulaw-6khz
audio/basic (Raw headerless) 8kHz 8-bit mono Mu Law [PCM] (G.711)

Mu Law .wav formats:
audio/x-mulaw-6khz-wav
audio/wav 8kHz Mu Law
audio/x-wav (RIFF header) 8kHz 8-bit mono Mu Law [PCM] (G.711)
audio/x-mulaw-8khz-wav

A Law .vox formats:
audio/x-alaw-6khz
audio/x-alaw-8khz
audio/x-alaw-basic (Raw headerless) 8kHz 8-bit mono A Law [PCM](G.711)

A Law .wav formats:
audio/x-alaw-6khz-wav
audio/x-wav (RIFF header) 8kHz 8-bit mono A Law [PCM] (G.711)

Note: The A Law values in the preceding list are for use in Europe and in areas outside North America.

Call Control Elements

VoiceXML has been extended in the current implementation to allow the execution of call control elements from the TXML language. These elements provide support for the following:

- Providing caller-entered digits to the agent or to the agent's desktop application before transferring a caller
- Allowing the agent to transfer a call to another agent

Voice applications that require call control on different legs of the call are beyond the scope of VoiceXML. However, call control extensions provide this functionality in addition to dialog interaction using VoiceXML.

Call control extensions can be used for:

- **Controlling multiple calls**—Independently control multiple outbound calls and optionally provide voice dialog interaction on each of them.
- **Whispering transfer**—Provide a message to the agent who is going to receive the call before connecting the caller to that agent.
- **Three-Way Calling**—Enable more than two people to converse with each other at the same time.

- **Event Handling**—Handle asynchronous events that come from the telephony infrastructure and the VoiceXML Interpreter.
- **VoiceXML Interpreter session initiation and termination**—Initiate a dialog session that is executed in a VoiceXML Interpreter and provide the ability to start and stop a VoiceXML session at any time.
- **Conditional logic**—Add conditional logic to voice applications using elements such as `<if>`, `<else>`, and `<elseif>`.
- **Post data to a web server**—Interact with a web server using elements such as `<goto>` and `<submit>`.

TXML

To use the TXML call extensions, it is necessary to know the syntax and layout of a TXML document.

The `<XMLPage>` element is the root element for every TXML document. Each TXML element must be set in an XMLPage. Each XMLPage represents a unit of work to be performed by the voice application. The unit of work consists of a single action or a linear list of a few actions.

An XMLPage has the following syntax:

Example

```
<?xml version="1.0"?>
<XMLPage TYPE= "IVR" CUSTID="AcmeTeleBroker-SF"
   PAGEID="0006" VERSION="2.5" SESSIONID="$sessionid$"
   HREF="http://telera.net/voiceXML.vxml/url">
   ...content....
</XMLPage>
```

The first element of a TXML document is the XML declaration. This element identifies the document as an XML document. The contents of a TXML page are enclosed within an opening element, `<XMLPage>`, and a closing element, `</XMLPage>`. The opening element contains the attributes listed in Table 14.

Table 14: <XMLPage> Attributes

Attribute	Description
TYPE	Voice applications must use the value `IVR`. Other platform components use different values to identify the source of the page.
CUSTID	Identifies the customer; for example, `ACME-SF`.
PAGEID	(Optional) A character string, which can consist of numerals, to identify each page. (This attribute does not have a functional purpose; it is for the convenience of the programmer only.)

Table 14: <XMLPage> Attributes (Continued)

Attribute	Description
VERSION	Identifies the version of the TXML specification used by the application.
SESSIONID	Identifies the particular session with the caller. This is part of the query string sent by the MCP in the HTTP request for each page.
HREF	URI of the next XMLPage or VoiceXML document to fetch when the voice application reaches the end of the page.

Object Element Extensions

Genesys Voice Platform provides the following object extensions:

- `CRData:get`—Gets data from Framework
- `CRData:put`—Sends attached data to Framework
- `CRData:genericAction`—Performs an action on Framework
- `telephonydata:put`—Sends SIP `INFO` messages
- `transactionalrecord:start`—Starts transactional recording
- `transactionalrecord:stop`—Stops transactional recording
- `asr:freeResource`—Frees an ASR resource

Classid CRData:get

Note: This object extension is only applicable for the IVR Server, URS interaction.

The application can retrieve the data from the server by having a VoiceXML object element with `classid="CRData:get"` in the form. The `param` element can specify the keys. All the `param` names pass to the server as a namelist (key1 key2...).

Note: You must specify `expr` as an empty string for the `param` element.

Example

```
<form id="user_data">
  <object name="getuserdata"
      classid="CRData:get">
<param name="key1"  expr="''" />
```

```
    </object>
</form>
```

When this `<object>` is executed, it returns the value of all the keys in an XMLPage that will be set to the value of the object form item variable.

The returned XMLPage will be:

```
<crGetData><key name='key1' value='xyz'/><key name='key2'
value='xyz'/><key name='key3' value='xyz'/></crGetData>
```

The schema of this XMLPage will be:

```
            <!--Here we are restricting @name to ID to be unique -->
<xsd:element name="telera:key">
            <xsd:complexType>
                  <xsd:attribute name="name" type="xsd:ID" />
                  <xsd:attribute name="value" type="xsd:string" />
            </xsd:complexType>
</xsd:element>
<!--Here we are insist that you cannot have empty tag -->
<!--If it needs to be empty then lastresult array will-->
<!--have 'undefined'set as per VoiceXML spec.........-->
<xsd:element name="telera:crGetData">
            <xsd:complexType>
                  <xsd:choice minOccurs="1" maxOccurs="unbounded">
                        <xsd:element ref="telera:key" />
                  </xsd:choice>
            </xsd:complexType>
</xsd:element>
```

Length/size limit: 2 KB

Classid CRData:put

Note: This object extension is only applicable for the IVR Server, URS interaction.

To send the user data from the application to the server, include the `classid="CRData:put"` in the `object` element.

Example

```
<form id="user_data">
  <object name="senduserdata"
```

```
      classid="CRData:put">
<param name="key1"  value="value1" />
    </object>
</form>
```

When this `<object>` is executed, the value of the object form item variable will be set to true.

Classid CRData:genericAction

You can include the `classid="CRData:genericAction"` in the `object` element for additional types of data.

Example

```
<object name='myData' classid='CRData:genericAction'>
   <param name='IServer_Action' value='CED'/>
   <param name='mydata' value='45'/>
   ......
   ......
</object>
```

The preceding element shows how data is sent to and from the web server. More than one `<object/>` element may be sent in a single form. The return result will be in this format:

```
<crGetData>
<key name='ResultCode' value='Success'/>
</crGetData>
```

The above item shows a successful transaction.

The @value can be:

- `Success`
- `NoSuchStat` (specific to `IServer_Action: PeekStatReq` and `GetStatReq`)
- `MiscError`
- `Failure`

A `ResultCode` of `Success` may include additional parameters that return information requested.

The following `IServer_Action` values are currently supported:

- `UDataDel`
- `UData`
- `CED`
- `ExtnsEx`
- `GetStatReq`
- `PeekStatReq`
- `AccessNumGet`

UDataDel Example

This example deletes a list of keys.

```
<form>
<object name='mydel' classid='CRData:genericAction'>
<param name='IServer_Action' value='UDataDel'/>
<param name='Action' value='DeleteKey'/>
<param name='mykey1' value=''/>
<param name='mykey2' value=''/>
</object>
</form>
```

The <param/> with @name="IServer_Action" and @value = "UDataDel" is mandatory. The user must then specify the @name='action'.

This example deletes all keys.

```
<form>
<object name='mydel' classid ='CRData: genericAction'>
<param name='IServer_Action' value=' UdataDel'/>
<param name='Action' value=' DeleteAll'/>
</object>
</form>
```

The <param/> with @name="IServer_Action" and @value = "DeleteKey" is mandatory and is the only child. The user does not send any keys.

UData, CED, ExtnsEx Example

```
<form>
<object name='myUdata' classid ='CRData:genericAction'>
<param name='IServer_Action' value=' UData'/>
<param name='GenesysRouteDN' value='5001'/>
<param name='mykey2' value='vys'/>
</object>
<object name='myCED' classid ='CRData:genericAction'>
<param name='IServer_Action' value=' CED'/>
<param name='ced' value='567'/>
</object>
<object name='myExtnData' classid ='CRData: genericAction'>
<param name='IServer_Action' value=' ExtnsEx'/>
<param name='mykey4' value='34'/>
<param name='mykey5' value='23232'/>
</object>
</form>
```

GetStatReq Example

```
<form>
<object name='mydel' classid ='CRData: genericAction'>
<param name='IServer_Action' value='GetStatReq'/>
<param name='RequestID' value='xyz'/>
<param name='ServerName' value='abc'/>
<param name='StatType' value='asf'/>
<param name='ObjectId' value='asf'/>
<param name='ObjecType' value='asf'/>
</object>
</form>
```

All the above parameters are mandatory.

PeekStatReq Example

```
<form>
<object name='mydel' classid ='CRData:genericAction'>
<param name='IServer_Action' value='PeekStatReq'/>
<param name='RequestID' value='xyz'/>
<param name='StatName' value=' CurrNumberWaitingCalls'/>
</object>
</form>
```

Note: The `StatName` can have one of two values: `CurrNumberWaitingCalls` or `ExpectedWaitTime`.

AccessNumGet Example

```
<form>
   <object name='mydel' classid ='CRData:genericAction'>
      <param name='IServer_Action' value='AccessNumGet'/>
      <param name='DestDN' value='xyz'/>
      <param name='XRouteType' value='Default'/>
   </object>
</form>
```

The `@name=DestDN` is mandatory. The `@name=XRouteType` is optional.

The return value is either the number or the failure information.

Classid telephonydata:put

User data using SIP INFO can be sent using the `telephonydata:put` object class. The following syntax defines how a voice application can send user data:

```
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml"
xmlns:genesys="http://www.genesyslab.com/vxml/2.0/ext/20020430"
xml:lang="en-US">
   <form id="user_form">
      <object name="user_data" classid="telephonydata:put">
       <genesys:param name="msgtype" value="INFO" />
```

```
      <genesys:param name="header" append="false"
        value="x-genesys-header1:value1" />
      <genesys:param name="header" append="false"
        value="x-genesys-header1:value2" />
      <genesys:param name="body" value="This is a test message" />
     </object>
   </form>
</vxml>
```

Table 15 details the contents of the `telephonydata:put` class.

Table 15: Telephonydata:put Class Contents

Parameter	Values	Description
protocol	SIP	Indicates telephony call control protocol. If not provided, then the platform defaults is `SIP`.
msgtype	INFO	Indicates the SIP request/response to send. If not provided, then the platform defaults to `INFO`. Example: `<param name="msgtype" value="INFO">`
header	SIP headers with their values	Headers to add to the protocol defaults. The `append` key name will be added as a key that you can set to `true` or `false`. If `true`, the header is appended to other headers. If `false`, the header overwrites any existing headers with the same name. The default is `false`. The headers are not validated. the platform appends/replaces the given headers. If the body is given, the `Content-type` must be given as one of the headers. If it is not, the default header will be `Content-type: application/text` Example: `<param name="header" append="false" value="x-genesys-header-1:value1">`
body	Entire body of the SIP message	Content of the entire body portion of the SIP message. Maximum size is 1024 bytes. If the body is unspecified, the default body is empty.

In order to retrieve the data sent through the SIP INFO header inside a VoiceXML application, use the `session.genesys.sip` property.

Receiving User Data

GVP allows call control data to be exchanged between the SIP protocol and the VoiceXML application. The VoiceXML application accesses all incoming messages using the `Session` object, and it uses the ECMAScript `array` object

to present the SIP `INFO` messages that are received at anytime during the call. These messages are appended to the array as they are received.

```
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml"
xmlns:genesys="http://www.genesyslab.com/vxml/2.0/ext/20020430"
xml:lang="en-US">
<form id="user_form">
   <field type="digits">
      <prompt>This example assumes that the caller shall generate a
SIP INFO message before this prompt ends and the user presses 1.
Please press 1. </prompt>
   </field>
   <filled>
      <block>
         <prompt> The body of the SIP INFO message is <value
expr="session.genesys.sip[0].body"/></prompt>
      </block>
   </filled>
</form>
<vxml>
```

Classid transactionalrecord:start

Transactional recording can be started by having an `object` element with the attribute `classid transactionalrecord:start`. This element cannot be used multiple times, which means that transactional recording can be started only once during the call. If the application attempts to start transactional recording multiple times, an error event is thrown. Also, if the old style controls are in place (for example, turned on in application profile by enabling Transaction Recording), and the application attempts to start the recording using the object element, the same error event will be thrown. Both `posturl` and the mode of posting should be specified at the start, using the `param` element.

If the `transactionalrecord:stop` is not specified, the posting will be done at the end of the call to the `posturl` specified at the start. Therefore, `posturl` is a mandatory parameter, and an error event is thrown if it missing. The recording is posted directly to the application and the application is guaranteed to get the recording after the `stop` is done.

When this `<object>` is executed successfully, the value of the object form item variable will be set to `true`.

Example:

```
<object classid="transactionalrecord:start">
<param name="posturl"  value="http://... " />
<param name="mode"  value="sync" />
<param name=”type” value=””/>
</object>
```

Classid transactionalrecord:stop

Transactional recording can be stopped by having an `object` element with the attribute `classid transactionalrecord:stop`. This element cannot appear without a corresponding `object` element with the attribute `transactionalrecord:start`. If it appears without this corresponding object element, an error event will be thrown.

The `posturl` and `mode` are optional parameters in `transactionalrecord:stop`. If specified, they override the corresponding parameters that were specified in `transactionalrecord:start`.

When this `<object>` is executed successfully, the value of the object form item variable will be set to `true`. The recording and the variables present in the namelist parameter will be posted to the `posturl` that is specified in the `stop` (or `start`, if not specified in `stop`), and using the appropriate mode, as specified in the `stop` (or `start`, if not specified in `stop`). The `namelist` parameter is optional.

Example:

```
<object name="TestRecStop" classid="transactionalrecord:stop">
            <param name="posturl"
value="http://dev-transrec/upload/capture1.asp" />
            <param name="mode"  value="sync" />
            <param name="namelist"  value="mainmenu_input" />
        </object>
```

Note: Sometimes the `<prompt>` fails to record, even though it is placed ahead of the `<transactionalrecord:stop>` element in the voice application. This behavior occurs because the prompt is queued for playing only when `<transactionalrecord:stop>` is executed. The prompt is played later (when the user needs to provide input or because of call termination; see Section 4.1 in the VoiceXML 2.0 specification).

Transactional Recording Error Events

The errors in Table 16 could be thrown by GVP to the application depending on the scenario:

Table 16: Transactional Recording Error Events

Scenario	Error Event	Error Event Message
Transactional recording is already started, by an `appid` control or by a previous `transactionalrecord:start`, and the application attempts to start recording again.	`error.semantic.transrec.oneallowed`	Duplicate transactionrecord:start encountered.
One transactional recording is done in the call, and the application attempts to start recording again.	`error.semantic.transrec.oneallowed`	Only one transactional recording per call is allowed.
The `posturl` is not specified in `transactionalrecord:start`.	`error.semantic.posturl`	The transactional recording does not specify where the recording should be posted. Recording will not be started.
Transactional recording is not started and `stop` is issued.	`error.semantic.transrec.notstarted`	Transaction record should be started using transactionrecord:start, before stop is issued.
Transactional recording is already stopped and another `stop` is issued.	`error.semantic.transrec.oneallowed`	Duplicate transactionalrecord:stop encountered.
The application attempts to start transactional recording in the MCP.	`error. unsupported.object.transrec`	Transactional recording is not supported in MCP.
Transactional recording is started on more than one leg—for example, a bridged call scenario.	`error.semantic.transrec.notallowed.multiplelegs`	Transactional recording can only be started on one leg per call.

Transactional Recording and Regular Recording

GVP supports transactional recording and regular recording (for example, `<record>`) at the same time.

Transactional Recording and Media Types

GVP supports transactional recording with media formats. If the specified media format is not supported by the Media Server, GVP will generate the following event: `error.unsupported.transrec.type`.

The following code snippet is an example of how to specify the media format:

```
<vxml version="2.1"  xmlns="http://www.w3.org/2001/vxml"
   xmlns:genesys="http://www.genesyslab.com/vxml/2.0/ext/20020430" xml:lang="">
<form>
<var name="GS_callerleg_callrecording_filename"/>
<var name="GS_agentleg_callrecording_filename"/>
<object name="StartTrxnRecording" classid="transactionalrecord:start">
<param name="posturl"value="AutoGenerateCapture.jsp?SESSIONID=$sessionid$"/>
<param name="mode" value="async"/>
<param name="type" value="audio/basic"/>
</object>
<block>
<if cond="session.genesys.agent_leg_flag=='true'">
<assign name="GS_agentleg_callrecording_filename" expr="'C.\\\\Program Files\\\\VOice
   Platform Studio\\\\TEMP\\\\TRANSACTION-RECORDS\\\\ivronly-recording.vox'"/>
<submit next="startrecord_PROMPT.jsp" method="post"
   namelist="GS_agentleg_callrecording_filename"/>
<else/>




<assign name="GS_callerleg_callrecording_finleman" expr="'C:\\\\Program Files\\\\Voice
   Platform Studio\\\\TEMP\\\\TRANSACTION-RECORDS\\\\ivronly-recording.vox'"/>
<submit next="startrecord_PROMPT.jsp" method="post"
   namelist="GS_callerleg_callrecording_filename"/>
</if>
</block>
</form>
</vxml>
```

Classid asr:freeResource

GVP enables Voice applications to free an ASR resource on demand. You can use the `asr:freeResource` `classid` in the VoiceXML `<object>` element to instruct the Media Control Platform to free the MRCP ASR session that is currently being used for the call. This can have the effect of freeing the associated ASR license on the Media Control Platform. Check with your MRCP ASR vendor for details on how licenses are released and whether this is compliant with the vendor license agreements.

When the `<object>` element with `classid` `asr:freeResource` is issued, GVP tears down the MRCP session for the call. If there is a subsequent recognition in the call after `classid` `asr:freeReource` is issued, GVP implicitly sets up a new MRCP ASR session. It is not necessary to issue a separate `classid` to allocate the ASR resource.

The following is an example VoiceXML application that uses the `<object>` element with `classid` `asr:freeReource`.

```
<form id="user_form">
   <object name="freeResource" classid="asr:freeResource">
```

```
   </object>
</form>
```

When this `<object>` is executed, the value of the object form item variable is set to `true`.

If `asr:freeResource` is used in an ASR disabled VoiceXML application, the execution of `<object>` generates `error.semantic.ASRFrrResource.notapplicable`.

If TXML is used to control multiple legs that are to be bridged and recognition has already taken place on both legs, specify `asr:freeResource` separately on both legs.

Additionally, the Media Control Platform implicitly releases the MRCP session when bridging a call if there are no active grammars. The MRCP session is released from both legs of the bridged call.

Session Variables

session.genesys Object

All of the TXML `$` variables can be accessed using the VoiceXML `session.genesys` object. For example, `$application-name$` maps to `session.genesys.application_name`. Note that all hyphens (-) are converted to underscores (_).

Universal Connection ID

GVP fetches the universal connection ID in the behind-the-switch and in-front-of-the-switch modes. In the Network mode, the universal connection ID is not available to GVP at call setup time. The universal connection ID is exposed to the voice application through a session variable—`session.genesys.connid`.

The universal connection ID will be passed to the MRCP server through the logging MRCP element. The logging element format looks like this: `<SessionID>_<ConnectionID>_<TenantID>`

If the universal connection ID is not available, GVP fetches the connection ID.

Redirecting Number

The PSTN Connector provides the redirecting number using SIP `History-Info` header. GVPi exposes it using the `session.connection.redirect` array session variable.

UUI Data

GVPi exposes UUI data from the PSTN Connector using the `session.connection.aai` session variable.

User Data Attached by SSG

GVPi exposes the value of the header `X-Genesys-OutboundData` received in the `INVITE` as the `session.genesys.outbounddata` session variable to the voice application.

Chapter

3 Call Control Elements

This chapter describes the call-control-extension elements supported by the Genesys Voice Platform (GVP). Elements and their attributes are described together with their parent/child relationship to other elements. The elements are described according to their implementation in Genesys Voice Platform only.

Note: This manual describes the VoiceXML 2.1 language as implemented by the Genesys Voice Platform (GVP) in versions 7.6 and earlier. For information about the GVP Next Generation Interpreter (NGI), see the *Genesys Voice Platform 8.1 Genesys VoiceXML 2.1 Reference Help.*

This chapter covers the following topics:

Call Control Elements

Table 17 lists the call control elements and their attributes. The subsequent sections of this chapter describe each element's syntax, attributes, and child/parent elements, and provide an example of how the voice application uses the element.

Table 17: Call Control Elements

Element	Attribute
<ALERT_LEG>	LEG_ID IVRURL
<BRIDGE_CALL>	LEG_ID
<CREATE_LEG_AND_DIAL>	TELNUM IVRURL BRIDGE ENDSESSIONONHUP URL_ONLEG2HUP CPATIMEOUT ANI AFTERCONNECTTIMEOUT
<END_SESSION>	This element has no attributes.
<HANGUP_AND_DESTROY_LEG>	REASON
<LEG_WAIT>	TIMEOUT HREF
<ON_LEGHUP>	ENDSESSION OTHER_LEG_URL
<QUEUE_CALL>	AGENTGRP USR_PARAMS AGENT_URL
<SCRIPT_RESULT>	USR_PARAMS
<SET>	VARNAME VALUE
<UNBRIDGE_CALL>	LEG_ID OTHER_LEG_URL

<ALERT_LEG>

Use the <ALERT_LEG> element after unbridging a call to alert the other leg and direct it to a specified URL. The <ALERT_LEG> element interrupts an interpreter in a <LEG_WAIT> state and asks it to send an HTTP GET request to the specified URL.

Syntax

```
<ALERT LEG
LEG_ID="name"
IVRURL="URL"
/>
```

Attributes

LEG_ID	Identifies other leg(s) to alert. These other legs normally sit in a LEG_WAIT state. If the value is ALL, all other legs of this session are alerted.
IVRURL	URL from which to execute the next document for the other legs after they have been alerted.

Child/Parent Elements

Child Elements (can contain)	none
Parent Elements (used in)	<XMLPage>

Example

```
<?xml version="1.0"?>
<XMLPage TYPE="IVR" PAGEID="Alert1" SESSIONID=""
HREF="$ivr-root-dir$/BRIDGE1.ASP?PAGEID=Alert1">
    <ALERT_LEG LEG_ID="ALL" IVRURL="$ivr-root-dir$/
    PROMPT1.ASP?PAGEID=Alert1" />
</XMLPage>
```

<BRIDGE_CALL>

This element bridges the current call with another call on the same PopGateway. When the voice application executes the <BRIDGE_CALL> element, it interrupts the leg of the bridged call and aborts any messages being played on the leg.

Syntax

```
<BRIDGE_CALL
LEG_ID="NAME"
/>
```

Attribute

LEG_ID	The ID of the leg whose call is to be bridged with the call on this leg. The `ALL` value bridges the call on this leg with all other calls associated with the various legs of the session.

Child/Parent Elements

Child Elements (can contain)	none
Parent Elements (used in)	`<XMLPage>`

A `<LEG_WAIT>` element in the XMLPage must follow the `<BRIDGE_CALL>` element; otherwise, the interpreter looks for an XML element after the `<BRIDGE>` element, and if there are no more elements, it will try to get the next document from the `HREF` specified at the top of the XML page.

If there is an error in bridging the calls, the voice application raises an error. See the "Error Extensions" on page 37 for details. It is important to note that the appropriate error extensions must be placed in the VoiceXML 2.1 root document.

Examples

Caller Document

```
<?xml version="1.0"?>
<vxml version="2.1" application="app-root.vxml">

    <form id="customer_service">
       <field name="agent_transfer">

          <prompt>
             Please wait while we are transferring you to an agent
          </prompt>
          <goto next="http://cld.xml"/>
       </field>
    </form>
</vxml>
```

CLD Document

```
<?xml version="1.0"?>
<XMLPage TYPE= "IVR" CUSTID="bridge_call" VERSION="2.5"
    SESSIONID="$sessionid$" HREF="http://Telera.net/
    agent.xml">
```

```
    <CREATE_LEG_AND_DIAL/>
       <LEG_WAIT/>
       <!-- BRIDGE_CALL must be followed by LEG_WAIT -->
</XMLPage>
```

Agent Document

```
<?xml version= "1.0"?>
<vxml version="2.1 application= "app-root2.vxml">

    <form id="agent">
       <field name="call_transfer">

          <prompt>
             You are getting a call from
             <var name="customer" expr="name"/>
          </prompt>
       </field>

       <goto next="bridge_call.xml"/>
    </form>
</vxml>
```

Bridge_Call Document

```
<?xml version="1.0"?>
< XMLPage TYPE= "IVR" CUSTID="caller" VERSION="2.5"
    SESSIONID="$sessionid$" HREF="http://Telera.net/
    caller.vxml">

    <BRIDGE_CALL LEGID="ALL"/>
    <LEG_WAIT/>
</XMLPage>
```

<CREATE_LEG_AND_DIAL>

The <CREATE_LEG_AND_DIAL> element enables a voice application to make outbound calls. Using this element, instead of the VoiceXML <transfer> element, enhances call control of the outbound call.

Syntax

```
<CREATE_LEG_AND_DIAL
TELNUM="telephone number to be dialed"
IVRURL="URL"
ANI="Callers phone number"
BRIDGE="YES | NO"
ENDSESSIONHUP="YES | NO"
URL_ONLEG2HUP="URL"
CPATIMEOUT="timeinsecs"
AFTERCONNECTTIMEOUT="timeinsecs"
/>
```

Attributes

TELNUM	The telephone number that the Telephony server needs to call. **Note:** The `TELNUM` attribute specifies the phone number. Additional characters sent as in-band DTMF (dual-tone multi-frequency) tones optionally follow the phone number. The optional additional characters can contain the 12 keypad characters as well as pause characters (commas, each of which is treated as a 1-second pause). The phone number itself cannot contain any pause characters. The first pause character indicates the end of the phone number. For example, `TELNUM=12159090,,,**,90061234#` instructs the telephony server to dial 12159090 to establish the call, pause 3 seconds, send ** tones, pause 1 second, and send 90061234# tones. The processing of the additional characters begins immediately after the phone number is dialed; it does not wait for an answer. The number of pauses may have to be tuned before obtaining the correct pause for the ACD/CTI application.
IVRURL	The URL of the document to execute after the outbound call is made and the call is answered. The voice application uses this attribute to play an audio message or carry out other VoiceXML commands. This attribute can also carry a value of `LEG_WAIT`, in which case the new VoiceXML interpreter goes into an interruptible Wait state. The caller hears silence until an interrupt causes the interpreter to bridge the call. The `IVRURL` must be equal to `LEG_WAIT,` in case the `BRIDGE` attribute is set to `YES`.
BRIDGE	(Optional) `YES` or `NO`. If YES, the new call bridges with the call on the current leg as soon as the new call is dialed and answered.

Attributes

ENDSESSIONONHUP	(Optional) `YES` or `NO`. Default value is `YES`. If YES or absent, a hangup on the new leg ends the session and destroys all the legs in the session. If the value is NO, a hangup on the outbound leg only destroys that leg. The `URL_ONLEG2HUP` attribute determines the behavior of the inbound leg.
URL_ONLEG2HUP	(Optional) This attributes specifies the URL where the inbound leg should fetch its next XML page if the outbound leg hangs up and `ENDSESSIONONHUP=NO`. If `ENDSESSIONONHUP=NO` and this attribute is missing, the inbound leg continues in the current state (`LEG_WAIT` if it was previously bridged) after the hangup occurs on the outbound leg.
CPATIMEOUT	(Optional) The maximum time, in seconds, before the MCP returns the Call Progress Analysis result. The default value = `0` (do not perform call progress analysis). The CPA result is set in `$_cparesult$`. The voice application should include `$_cparesult$` in the `IVRURL` of `<CREATE_LEG_AND_DIAL>`. The `$_cparesult$` should also be included in the exception event URL of `DIAL_ERROR`.
ANI	This is the Caller ID sent out for the outbound call.
AFTERCONNECTTIMEOUT	(Optional) This is in seconds. This timeout is set when the application needs to detect an answering machine or fax. If this timeout is set, after the outbound call is answered, it will wait for specified `afterconnecttimeout` seconds to detect a fax or answering machine.

Child/Parent Elements

Child Elements (can contain)	none
Parent Elements (used in)	`<XMLPage>`

Normally, a `<LEG_WAIT>` element follows the `<CREATE_LEG_AND_DIAL>` element. This method is used to wait for the other leg to synchronize with this leg. In the case of errors, the voice application throws an error. See the "Error Extensions" on page 37 for details. It is important to note that the appropriate error extensions must be placed in the VoiceXML 2.1 root document.

How It Works

When the voice application first presents the request to the Interpreter Context, it immediately receives a `<RESPONSE>` element with `RESULT=SUCCESS`. Depending upon the `HREF` at the top of the XML page containing the `<CREATE_LEG_AND_DIAL>` element or the elements following that element, the

Interpreter Context instructs the voice application to fetch its next page from the HREF (it could be "<LEG_WAIT>" on page 63).

For example, if the Interpreter Context discovers later that the new leg was created but the dial out failed, the voice application raises the error.com.telera.dial exception if this error exception is placed in the VoiceXML 2.1 root document. The URL that handles this exception destroys the second leg and can send an interrupt to the first leg. This forces the voice application to raise the exception on the first leg. The voice application then consults the voice application exception map to find the URL to consult for the exception.

In the following example, when the caller requests the customer service option, the <CREATE_LEG_AND_DIAL> element connects the caller with an agent. The TELNUM attribute specifies the number to dial. The BRIDGE attribute is set to YES so that the caller is immediately connected with the agent. The IVRURL specifies the document to execute for the second leg once the call has been connected. ANI=ani specifies that the original caller's ANI will be sent out as the caller ID on the outbound call.

This document can "whisper" information about the caller to the agent before using the <BRIDGE> element to bridge the agent with that caller.

Example

```
<XMLPage TYPE="IVR" CUSTID="VVAA"  PAGEID="2" VERSION="0.72" SESSIONID="$sessionid$"
HREF="<%=m_onholdURL%>">
<ON_LEGHUP ENDSESSION="NO" OTHER_LEG_URL="<%=m_agentafterURL%>" />
<CREATE_LEG_AND_DIAL TELNUM="<number>"
       BRIDGE="NO"
       CPATIMEOUT="20"
       AFTERCONNECTTIMEOUT="10"
       IVRURL="<%=m_AgentStartURL%>"
       URL_ONLEG2HUP="<%=m_callerafterURL%>"
       ENDSESSIONONHUP="NO" />
       ANI=$did$
</XMLPage>
```

Document 1

```
<?xml version= "1.0"?>
<vxml version="2.1" application="app-root.vxml">
   <menu>

       <prompt>Welcome to Banking 24. Say one of:
       <enumerate/>
       </prompt>

       <choice next="http://www.personal_options.vxml">
          Personal Options
       </choice>
```

```
        <choice next="http://www.pay_bill.vxml">
            Bill Payment
        </choice>

        <choice next="customer_service">
            Register with Banking 24
        </choice>
    </menu>

    <form id="customer_service">
        <field name="agent_transfer">
            <prompt>Please wait while we are transferring you
                to an agent
            </prompt>
            <goto next="http://customer_service.xml"/>
        </field>
    </form>
</vxml>
```

Document 2 (customer_service.xml)

```
<?xml version="1.0"?>
<XMLPage TYPE= "IVR" CUSTID="customer_service" VERSION="2.5"
    SESSIONID="$sessionid$"
    HREF="http://Telera.net/tele/menu.vxml">

    <CREATE_LEG_AND_DIAL TELNUM="4082159090" BRIDGE="NO" ANI="4081234567"
        IVRURL="http://acme.Telera.net/tele/agent.asp"/>

        <LEG_WAIT/>
            <!-- waiting for other leg at agent.asp to bridge
                with this leg -->
</XMLPage>
```

<HANGUP_AND_DESTROY_LEG>

The <HANGUP_AND_DESTROY_LEG> element instructs the application to terminate the call on this leg and to destroy the interpreter leg receiving this element.

Syntax

```
<HANGUP_AND_DESTROY_LEG
REASON="NORMAL | ERROR | CALLERHUP"
/>
```

Attribute

REASON	Reserved for future use.

Child/Parent Elements

Child Elements (can contain)	none
Parent Elements (used in)	<XMLPage>

Example

```
<?xml version="1.0"?>
<XMLPage TYPE="IVR" PAGEID="KillLeg1" SESSIONID="" HREF="http://mypage.foo.com/exit.asp">
    <HANGUP_AND_DESTROY_LEG REASON="CALLERHUP" />
</XMLPage>
```

<END_SESSION>

The <END_SESSION> element destroys all legs of the session, and hangs up all of the associated calls.

Syntax

```
<END_SESSION/>
```

Attributes

The <END_SESSION> element has no attributes.

Child/Parent Elements

Child Elements (can contain)	none
Parent Elements (used in)	<XMLPage>

The following example illustrates the use of the <END_SESSION> element to terminate the session.

Example

```
<?xml version="1.0"?>
<vxml version="2.1" application="app-root.vxml">

    <form>

        <field name="add_services" type="Boolean">

            <prompt>
               would you like any additional services
            </prompt>
```

```
            <if cond="add_services=='yes'">
               <goto next="main_menu.vxml"/>
            <elseif cond="add_services=='no'"/>
               <goto next="goodbye.vxml"/>
            </if>
         </field>

         <field name="goodbye">
            <audio src="http//bank_services/goodbye.vox">
               Good bye and have a nice day
            </audio>

            <goto next="http://bank_services/
               end_session.xml"/>
         </field>
      </form>
</vxml>
```

Document 2

```
<?xml version="1.0"?>
<XMLPage TYPE= "IVR" CUSTID="end_session" VERSION="2.5"
    SESSIONID="$sessionid$" HREF="http://acme.Telera.net/tele/
    exit.xml">
    <END_SESSION/>
</XMLPage>
```

<LEG_WAIT>

The <LEG_WAIT> element places the Interpreter Context receiving this element into a Wait state. The conversation controller then waits for one of the following:

- An event from the voice application (for example, caller hangup)
- An interrupt from the interpreter
- An <ALERT_LEG> element from the voice application

Syntax

```
<LEG_WAIT
TIMEOUT="time in seconds"
HREF="URL"
/>
```

Attributes

TIMEOUT	Specifies the time, in seconds, after which control comes back to the voice application, unless interrupted by another leg before the timeout expires.
HREF	Specifies the URL of the next document to execute after the timeout expires.

Child/Parent Elements

Child Elements (can contain)	none
Parent Elements (used in)	`<XMLPage>`

In case an unexpected event occurs while waiting and no event handler is defined for that event, the `HREF` in the XMLPage root element at the top of the page is an error-handling URL.

The following example illustrates the use of the `<LEG_WAIT>` element. In this example, the `<LEG_WAIT>` element puts the voice application into a Wait state while the call is being bridged. The `TIMEOUT` attribute specifies the Wait state of `60` seconds. After this period, the voice application executes the services document.

Example

```
<?xml version="1.0"?>
<XMLPage TYPE= "IVR" CUSTID="transfer" VERSION="2.5"
SESSIONID="$sessionid$" HREF="http://.../next.xml">
<CREATE_LEG_AND_DIAL TELNUM="<number>" BRIDGE="NO" IVRURL="<http://...bridge.xml ...>"
   />
<LEG_WAIT TIMEOUT="20" HREF="http://... /
error.xml"/>
</XMLPage>
```

When an outbound call is successful, the outbound leg will go to the IVR URL that is specified in the `CREATE_LEG_AND_DIAL` element.

```
Sample of bridge.xml
<XMLPage TYPE= "IVR" CUSTID="bridge_call" VERSION="2.5"
SESSIONID="$sessionid$" >
<BRIDGE_CALL LEG_ID="ALL" />
<ALERT_LEG LEG_ID="ALL" IVRURL="<http://...infinitelegwait.xml" />
<LEG_WAIT/>
</XMLPage>
```

Example of `infinitelegwait.xml`:

```
<XMLPage TYPE= "IVR" CUSTID="bridge_call" VERSION="2.5"
SESSIONID="$sessionid$" >

<LEG_WAIT/>
</XMLPage>
```

Use Cases for `LEG_WAIT` Timeout

After issuing a `CREATE_LEG_AND_DIAL` or `QUEUE_CALL`, if the application is waiting for the outbound call to be made, it can use `LEG_WAIT` with the `TIMEOUT`.

If the `TIMEOUT` attribute is not used, and there is no response from the CTI or there is an outbound failure, the inbound caller would wait indefinitely.

If the `TIMEOUT` attribute is used, the application should calculate the approximate time for the outbound call to be made, and it should take care of the successful call scenario.

This means that even if the outbound call is successful and bridged, with a timeout set, the inbound leg will always timeout after the specified timeout, and the application cannot distinguish between the failure and success scenarios.To avoid this issue, when the call is bridged on the outbound call, the VoiceXML application should alert the inbound leg and the inbound leg should go to an infinite leg wait state (which means a LEG_WAIT with no TIMEOUT attribute).

<ON_LEGHUP>

The `ON_LEGHUP` element can be used by the application to specify whether the session should be ended when the leg hangs up. The default behavior is to end the session when any of the legs hang up the call. However, if you set the `ENDSESSION` attribute, the application can control whether the entire session should be ended, or whether only that call leg goes down. If the `endsession` attribute is set to `false`, the `OTHER_LEG_URL` can be specified so that the inbound leg is informed when the current leg, which set `endsession` to false, is going down.

This element can be used in any of the legs, which means that it can be in the inbound leg or outbound leg. However, the design of the application should take into account that the session will not be ended if `ENDSESSION` is set to `false`, and it should handle different scenarios in which the other leg is active, without being informed if `OTHER_LEG_URL` is not specified.

Attributes

ENDSESSION	Default is true. When it is set to false, only that leg goes down and the session is still active.
OTHER_LEG_URL	This is an optional attribute. This URL indicates whether the other leg should be informed when the current leg goes down. Then, the other leg will be interrupted, and it will go to the URL specified.

Example:

```
<XMLPage TYPE="IVR" CUSTID="VVAA" PAGEID="2" VERSION="0.72" SESSIONID="$sessionid$"
   HREF="<%=m_onholdURL%>">
<ON_LEGHUP ENDSESSION="NO" OTHER_LEG_URL="<%=m_agentafterURL%>" />
<CREATE_LEG_AND_DIAL TELNUM="<number>" BRIDGE="NO" CPATIMEOUT="20" AFTERCONNECTTIMEOUT="10"
   IVRURL="<%=m_AgentStartURL%>" URL_ONLEG2HUP="<%=m_callerafterURL%>" ENDSESSIONHUP="NO"
   />
</XMLPage>
```

<QUEUE_CALL>

When the GVPi executes this element, it sends a `RouteRequest` request to the CTI Connector > IVR Server > URS to queue the call.

Syntax

```
<QUEUE_CALL USR_PARAMS="comma separated key-value pairs"
AGENT_URL="URL"
/>
```

Attributes

USR_PARAMS	(Optional) Information that the caller enters into the Genesys Voice Platform system before requesting transfer to an agent. This field contains subparameters and values separated by a colon (:), with the different parameter-value pairs separated by commas. For example, `PARAM1:25,PARAM2:busy,PARAM3:411`. The CTI Connector forwards the values of the subparameters to the IVR Server. The `GenesysRouteDN` parameter is mandatory when used with IVR Server.
AGENT_URL	(Optional) This is the URL from which the leg of the Telephony server with the outbound call to the ACD fetches its XML page after the agent is connected. This can be used for "whispering" some caller-related information into the agent's telephone before bridging the agent with that caller.
TELNUM	Reserved for future use.

USR_PARAMS

If you are using the CTI Connector, USR_PARAMS must consist of one parameter called GenesysRouteDN. Therefore, USR_PARAMS is mandatory.

```
<QUEUE_CALL USR_PARAMS="GenesysRouteDN:9001" AGENT_URL="URL" />
```

The CTI Connector receives USR_PARAMS in the following format:

```
GenesysRouteDN=111&name1=value1&name2=value2
```

Child/Parent Elements

Child Elements (can contain)	none
Parent Elements (used in)	<XMLPage>

The following example illustrates the use of the <QUEUE_CALL> element to put a call on hold while transferring the call to another agent. In this example, when the voice application attempts to transfer a call and detects that the line is busy, it executes the <QUEUE_CALL> element. The AGENT_URL attribute specifies the document to execute once the call is connected.

It is important to note that the error exception error.com.telera.queue, described in “Error Extensions” on page 37, must be placed in the VoiceXML 2.1 root document.

Example

```
<?xml version="1.0"?>
<vxml version="2.1">

    <form id="welcome">
       <block>
          <prompt>
             Welcome to Telera
          </prompt>
       </block>

       <transfer name="newcall" dest="phone://4168592"
          connecttimeout="10s" bridge="true">

       <filled>
          <if cond="newcall='busy'">
             <prompt>
                 All our customer care agents are
                 currently busy; please hold while we
                 try to connect your call
             </prompt>

             <goto next="http://queue_call.xml"/>
          </if>
       </filled>
       </transfer>
```

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

QUEUE_CALL.xml

```
<?xml version="1.0"?>
<XMLPage TYPE= "IVR" CUSTID="queue_call" VERSION="2.5"
    SESSIONID="$sessionid$" HREF="http://Telera.net/tele/
    agent_sales.xml">
    <QUEUE_CALL AGENT_URL="agent_sales.vxml"/>
</XMLPage>
```

Example of TXML page:

```
<?xml version="1.0" encoding="utf-8"?>
<XMLPage TYPE="IVR" PAGEID="" SESSIONID=""
   HREF="http://172.24.129.161/studio/LegWait1.asp" >
<SET VARNAME="$scripturl$"
   VALUE="http://172.24.129.161/studio/START.asp?ScriptID=$sid$&amp;ScriptData=$scriptd
   ata$"/>
<QUEUE_CALL USR_PARAMS="GenesysRouteDN:1111"
   AGENT_URL="http://172.24.129.161/studio/RoutePoint1.asp?ACTION=SET_AGENT_LEG_FLAG"/>
</XMLPage>
```

<SCRIPT_RESULT>

The GVPi executes this element in response to the IVR Server's TreatCall request to the voice application. When the GVPi executes this element, it sends a `SCRIPT_RESULT` request to the Interpreter Context. The Interpreter Context presents the data to the IVR Server as a response to the previous request.

Syntax

```
<SCRIPT_RESULT
USR_PARAMS="CDATA"
/>
```

Attribute

USR_PARAMS	Information that the caller enters is provided to the voice application before requesting transfer to an agent. This field contains subparameters and values separated by a colon (:), with the different parameter-value pairs separated by commas. For example, `PARAM1:25,PARAM2:busy,PARAM3:411`. The CTI Connector translates the names and values of the subparameters for interpretation by the Call Router.

Note: Before sending this element, specify a dollar variable called `$scripturl$` in the voice application. This is the top `HREF` for the page that Interpreter Context generates in response to a `RUN_SCRIPT_REQ` from the CTI Connector.

USR_PARAMS

The CTI Connector receives `USR_PARAMS` passed in the following format:

```
name1=value1&name2=value2
```

Child/Parent Elements

Child Elements (can contain)	none
Parent Elements (used in)	`<XMLPage>`

Example

```
<?xml version="1.0"?>
<XMLPage TYPE="IVR" PAGEID="Script_Result1" SESSIONID="" HREF="" >
<SET VARNAME="$scripturl$" VALUE="$start-ivr-url$"/>
<SCRIPT_RESULT
USR_PARAMS="ICMVarMappingMode:MAP_SPECIFICVARS
CallVar1:APPID:,CallVar2:LOG_MODE:,CallVar3:APPLICATIONVERSION:,CallVar4:VOXFILEDIR:,CallVa
r5:VRCLVER:" >
</SCRIPT_RESULT>
<LEG_WAIT/>
</XMLPage>
```

Example:

```
<?xml version="1.0" encoding="utf-8" ?>
<XMLPage TYPE="IVR" PAGEID="" SESSIONID="{2B26B794-6E06-4743-B82E-E073CF498851}"
   HREF="" >
<SET VARNAME='$scripturl$'
   VALUE='http://172.24.129.161/studio/START.asp?ScriptID=$sid$&amp;ScriptData=$scriptd
   ata$'/>
<SCRIPT_RESULT/>
<LEG_WAIT TIMEOUT='15'
   HREF='http://172.24.129.161/studio/Treatment1.asp?ACTION=GenerateLegWaitEvent'/>
</XMLPage>
```

<SET>

The <SET> element provides stateless voice applications with the mechanism to maintain state and use session variables for any web server environment. Voice applications can use the <SET> element to define session variables (variables that last the lifetime of the current session/call). The <SET> element is also used to set certain standard variables for the TXML interpreter.

Syntax

```
<SET
VARNAME="name"
VALUE="value"
/>
```

Attributes

VARNAME	The name of the variable. The name must start and end with a dollar sign ($). Within the enclosed dollar signs, it can have up to 30 letters, digits, underscores, or hyphens in any combination.
VALUE	The value to assign to the variable. The value can have any printable ASCII characters. The maximum length allowed for a value is 255 characters.

Child/Parent Elements

Child Elements (can contain)	none
Parent Elements (used in)	<XMLPage>

Voice applications use the <SET> element to set the value of a user definable variable for later substitution by the MCP. The <SET> element tells the MCP to associate a particular value with a variable name. When the voice application uses the specified variable in a document, the MCP substitutes the variable name with the value. To retrieve the value of the session variable, put the variable in the query string of the voice application URI that must use the value of the variable.

Example

```
<?xml version="1.0"?>
<XMLPage TYPE="IVR" PAGEID="Start1" SESSIONID="" HREF="http://mypage.foo.com/start1.asp">
<!--Set the $badxmlpageposturl$ value by using the set tag -->
<SET VARNAME="$badxmlpageposturl$"
VALUE="http://mypage.foo.com/handleErrot.asp?NextAction=BadPage&amp;
LASTERROR=$last-error$&amp;LASTERRSTR=$last-error-string$&amp;LASTERRURL=$last-error-url$/>
    <HANGUP_AND_DESTROY_LEG REASON="CALLERHUP" />
  </XMLPage>
```

<UNBRIDGE_CALL>

The <UNBRIDGE_CALL> element breaks the bridge between designated legs of a call. After the calls are unbridged, the voice application executes the element following the <UNBRIDGE_CALL> element. If there is no element following the <UNBRIDGE_CALL> element, the voice application executes the next document from the HREF specified in the XMLPage root element at the top of the page.

Syntax

```
<UNBRIDGE_CALL
LEG_ID="name"
OTHER_LEG_URL="URL"
/>
```

Attributes

LEG_ID	(Required) Specifies the leg whose call is to be unbridged. A value of ALL breaks the bridge between the current leg's call and all other calls bridged with this call.
OTHER_LEG_URL	(Optional) The URL from where the other leg(s) should fetch their next XML page after being unbridged. If this attribute is omitted, the other leg(s) remain in LEG_WAIT state until the voice application executes an <ALERT_LEG> element.

It is assumed that one of the legs is implicitly the one receiving this <UNBRIDGE_CALL> element. If there is an error in unbridging the calls, the voice application raises an error. See the "Error Extensions" on page 37 for details. It is important to note that the appropriate error extensions must be placed in the VoiceXML 2.1 root document.

Child/Parent Elements

Child Elements (can contain)	none
Parent Elements (used in)	<XMLPage>

The following example illustrates the use of the <UNBRIDGED> element. In this example, a hang-up event executes the <UNBRIDGE_CALL> element. The LEG_ID attribute specifies which leg of the call to unbridge, and control of the voice application returns to the caller.vxml document.

Example Caller Document

```
<?xml version="1.0"?>
<vxml version="2.1" application="app-root.vxml">

   <form id="customer_service">
      <field name="agent_transfer">
```

```
          <prompt>
             Please wait while we are transferring you to an agent
          </prompt>

          <goto next="http://bridge_call.xml"/>
       </field>

       <catch event="hangup">
          <goto next="http://unbridge_call.xml"/>
       </catch>
    </form>
</vxml>
```

Example Bridge_Call.xml

```
<?xml version="1.0"?>
<XMLPage TYPE= "IVR" CUSTID="bridge_call" VERSION="2.5"
    SESSIONID="$sessionid$" HREF="http://Telera.net/
    agent.xml">

    <BRIDGE_CALL  LEG_ID="ALL" />
       <LEG_WAIT/>
       <!-- BRIDGE_CALL must be followed by LEG_WAIT -->
</XMLPage>
```

Example Agent Document

```
<?xml version="1.0"?>
<vxml version="2.1" application="app-root2.vxml">

    <form id="agent">
       <field name="call_transfer">
          <prompt>
             You are getting a call from
             <var name="customer" expr="name"/>
          </prompt>
       </field>

       <throw event="hangup">
          <goto next="http://unbridge_call.xml"/>
       </throw>
    </form>
</vxml>
```

Example Unbridge_Call.xml

```
<?xml version="1.0"?>
<XMLPage TYPE= "IVR" CUSTID="unbridge_call" VERSION="2.5"
    SESSIONID="$sessionid$" HREF="http://Telera.net/
    customer_service.vxml">

    <UNBRIDGE_CALLLEG_ID="agent.xml"/>
</XMLPage>
```

Treatments

All of the parameters for the treatments are transparent to the Genesys Voice Platform (GVP). The interpretation of these parameters is entirely within the context of the voice application and the URS strategy.

Note: The `APP_ID` parameter in the URS Strategy is passed as the `$sid$` variable to GVP.

PlayAnnouncement

This treatment is used to play an announcement block to the calling party. The entire announcement block can consist of a series of Announcement Elements pieced together. Each Announcement Element can be described as interruptible or noninterruptible.

Parameters

PROMPT

Contains 10 possible sublists, numbered 1–10. Each sublist contains entries describing an `announcement` element, which are as follows:

- Interruptible (boolean)—Indicates whether the caller can interrupt the announcement.

Specify one of the following options:

- ID (integer)—ID of a message to play.
- Digits—Number to pronounce. The first digit defines how the number should be pronounced:
  - 0—One at a time (For example, 411 will be pronounced as four-one-one)
  - 1—Date (For example, the eleventh of April)
  - 2—Time (For example, four eleven AM)
  - 3—Phone number (For example, four-one-one)
  - 4—Money (For example, four dollars and eleven cents)
  - 5—Number (For example, four hundred and eleven)
- User_Ann_ID (integer)—User Announcement ID as returned after a successful `RecordUserAnnouncement` request.
- Text (ASCII text)—Pronounce using text-to-speech technology (if supported by the IP equipment).

Parameters

LANGUAGE	(Optional) Language indicator. Contains a string specifying the language in which the announcement should be made. The valid languages include, but are not limited to, English (US), Spanish, Mandarin, Cantonese, Vietnamese, French, French (Canada), German, Italian, Japanese, Korean, and Russian.

PlayAnnouncementandCollectDigits

This treatment is used to play an announcement block and collect digits from the caller. Typically, the announcement includes instructions requesting information from the caller.

Parameters

All of the parameters of PlayAnnouncement are recognized.

PlayApplication

This treatment is used to execute a voice application or a script on the voice application. It is possible to pass parameters to the voice application and get return values.

Parameters

APP_ID	Application ID (integer). Specifies the voice application to be run.
LANGUAGE	Language specifier (string).

Music

This treatment is used to connect the interaction to a music source.

Parameters

MUSIC_DN	Directory number of the music source.
DURATION	(Optional) Music duration in seconds.

Retransfer

Retransfer is the mechanism by which a user is transferred from one agent to another agent. Currently, reroute is the only retransfer option available.

The voice application should be coded to support treatments because after the reroute has been initiated by the first agent, the user leg can be issued treatments before retransferring to a new agent.

For this type of transfer, you must set the following parameters in the IVR Profile:

- `voicexml.gvpi.$transfer-type$—fixed, 2SignalChannel`
- `voicexml.gvpi.$cti_endcall_on_agentleg_hup$—fixed, 0`

Note: The Genesys IVR Server supports reroute in `Network` mode only.

You can control how long the platform will wait before ending the call if the agent leg drops without initiating a reroute by setting the `voicexml.gvpi.$rexfertimeout$` in the IVR Profile. For more information on this parameter, see the *Genesys Voice Platform 8.1 User's Guide*.

Appendix

A AT&T Transfer Connect

This appendix describes the AT&T Transfer Connect feature, and how to configure GVP to use this feature.

This appendix contains the following section:

AT&T In-Band Transfer Connect

The in-band transfer connect feature allows GVP to transfer calls to agents using DTMF tones. The network also provides call states and call progress updates with DTMF tones.

There are three types of in-band transfers:

- Courtesy Transfer (ATTCourtesy)—GVPi is disconnected when the agent DN is sent to the network (same as a blind transfer).

  Configure the IVR Profile with the following options:

  - `transfer type`—`1SignalChannel`
  - `transfer option`—`ATTCourtesy`

- Consult and Transfer—GVPi remains on the call until a successful connection is established between the caller and the agent.

  Configure the IVR Profile with the following options:

  - `transfer type`—`1SignalChannel`
  - `transfer option`—`ATTConsultative`

- Conference and Transfer—GVPi and the agent can have a private conversation while the caller is on hold.

  Configure the IVR Profile with the following options:

  - `transfer type`—`1SignalChannel`
  - `transfer option`—`ATTConference`

For information about which parameters are configured in the IVR Profile, see the *Genesys Voice Platform 8.1 User's Guide*.

Courtesy Transfer Example Script

The following lists the steps required for the TXML Script to implement Courtesy Transfer:

1. Define the network error tones:

   1—Hangup; HANGUP_DTMF=1

   5—Network error, limits exceeded; ERROR_DTMF_1=5

   6—Transfer; HANGUP_DTMF=6

   7—Dialed Number not recognized;ERROR_DTMF_2=7

   8—Invalid command; ERROR_DTMF=8
2. Trigger the Transfer Connect:

   10_ACTION = DialDTMF

   10_PARAM = *8
3. Wait, then outpulse Transfer numbers that the application provides:

   15_ACTION = Pause

   15_PARAM = 1

   20_ACTION = DialDTMF

   20_PARAM = <TNT_DialNum>
4. Get the end of the transfer number:

   25_ACTION = DialDTMF

   25_PARAM = #
5. Get the confirmation that the call was transferred:

   30_ACTION = GetDTMF

   30_PARAM = **6
6. Hangup the call:

   40_ACTION = Hangup

   50_ACTION = END

Converted XML Script for XferConnect

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml"
  xmlns:telera="http://www.telera.com/vxml/2.0/ext/20020430"
  xmlns:genesys="http://www.genesyslab.com/vxml/2.0/ext/20020430"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
   http://www.w3.org/TR/voicexml20/vxml.xsd">
```

```
<property name="com.telera.speechenabled" value="false" />
<property name="timeout" value="5s" />

<catch>
    <goto next="http://localhost/XferConnect/error.xml" />
</catch>

<form id="att_courtesy">


    <block>
        <prompt bargein="false" >
            <telera:value mode="dtmfplay" expr="'*8'"/>
            <audio src="silence1000ms.wav"/>
        </prompt>
    </block>



    <field name="returncode" >
            <prompt bargein="false" >
                <telera:value mode="dtmfplay"
   expr="session.genesys.transferscript_number"/>
                <telera:value mode="dtmfplay" expr="'#'"/>
            </prompt>
    <grammar version="1.0" xml:lang="en-US" xmlns="http://www.w3.org/2001/vxml"
   type="application/srgs+xml" root="rootdtmf"
        mode="dtmf">
            <rule id="rootdtmf" scope="public">
                <one-of>
                  <item> **6 </item>
                  <item> **5 </item>
                  <item> **7 </item>
                  <item> **8 </item>
                  <item> **1 </item>
                </one-of>
             </rule>
    </grammar>

    <filled>
                <if cond ="returncode == '**6'">
                    <goto next="http://localhost/XferConnect/hangup.xml" />
                <elseif cond ="returncode == '**5'" />
                    <goto next="http://localhost/XferConnect/error.xml" />
                <elseif cond ="returncode == '**7'" />
                    <goto next="http://localhost/XferConnect/error.xml" />
                <elseif cond ="returncode == '**8'" />
                    <goto next="http://localhost/XferConnect/error.xml" />
                <elseif cond ="returncode == '**1'" />
                    <goto next="http://localhost/XferConnect/hangup.xml" />
```

```
                </if>
    </filled>
    <noinput>
                <goto next="http://localhost/XferConnect/error.xml" />
    </noinput>

    </field>

</form>
</vxml>
```

Converted XML Script for Error Handling (error.xml)

```
<?xml version="1.0" ?>
<XMLPage TYPE= "IVR" CUSTID="ATT" PAGEID="ATT" VERSION="1.6"
HREF="hangup.xml" >
   <PLAY INTERRUPTIBLE="NO">
   <DTMF VALUE="$tntreclaimcode$" />
   </PLAY>
<!-- If reclaim code is present, then play that, otherwise the
application throws an error.
Enable the below snippet to throw an error. Since there is no input, it
leads to a timeout exception, which results as an error event to the
application.
   <MENU>
   <PLAY>
   <PAUSE TIME="1" />
   </PLAY>
   <EXCEPTIONMAP>
   </EXCEPTIONMAP>
   </MENU>-->
</XMLPage>
```

Converted XML Script for Hangup After Successful TransferConnect (hangup.xml)

```
<?xml version="1.0" ?>
<XMLPage TYPE= "IVR" CUSTID="ATT" PAGEID="ATT" VERSION="1.6" HREF=""
>
<END_SESSION />
</XMLPage>
```

AT&T Out-of-Band Transfer Connect

When performing a transfer connect that involves out-of-band (OOB) transfers, GVPi interacts with the network by notifying the GVP PSTN Connector about specific types of transfers.

GVPi supports three types of out-of-band transfers:

- Courtesy Transfer—GVPi is disconnected when the agent DN is sent to the network (same as a blind transfer).
- Consult and Transfer—GVPi remains on the call until a successful connection is established between the caller and the agent.
- Conference and Transfer—GVPi and the agent can have a private conversation while the caller is on hold.

For additional information about parameters that are configured in the IVR Profile, see the *Genesys Voice Platform 8.1 User's Guide*.

AT&T OOB Courtesy Transfer

GVPi allows AT&T OOB Courtesy Transfer with the PSTN Connector in front of the switch. To achieve this transfer using either VoiceXML or TXML, configure the following settings:

VoiceXML

1. Set the `<transfer>` element with `bridge=false` or `type=blind`.
2. Configure the IVR Profile with the following options:
   - `transfer type`—`1SignalChannel`
   - `transfer option`—`ATTOOBCOURTESY`

GVPi will execute `ATTOOBCourtesy.xml` before throwing a `connection.disconnect.transfer` event to the VoiceXML application. By default, `ATTOOBCourtesy.xml` is empty.

TXML

1. Set the `<CREATE_LEG_AND_DIAL>` element with `bridge=yes|no`.
2. Configure the IVR Profile with the following options:
   - `transfer type`—`1SignalChannel`
   - `transfer option`—`ATTOOBCOURTESY`

GVPi will execute `ATTOOBCourtesy.xml` before hangup. By default, `ATTOOBCourtesy.xml` is empty.

AT&T OOB Consult Transfer

GVPi allows AT&T OOB Consult Transfer with the PSTN Connector in front of the switch. To achieve this transfer using either VoiceXML or TXML, configure the following settings:

VoiceXML

1. Set the `<transfer>` element with `bridge=false` or `type=blind`.
2. Configure the IVR Profile with the following options:
   - `transfer type`—`1SignalChannel`
   - `transfer option`—`ATTOOBCONSULT`

GVPi will execute `ATTOOBConsult.xml` before throwing a `connection.disconnect.transfer` event to the VoiceXML application. By default, `ATTOOBConsult.xml` is empty.

TXML

1. Set the `<CREATE_LEG_AND_DIAL>` element with `bridge=yes|no`.
2. Configure the IVR Profile with the following options:
   - `transfer type`—`1SignalChannel`
   - `transfer option`—`ATTOOBCONSULT`

GVPi will execute `ATTOOBConsult.xml` before hangup. By default, `ATTOOBConsult.xml` is empty.

AT&T OOB Conference Transfer

GVPi allows AT&T OOB Consult Transfer with the PSTN Connector in front of the switch. To achieve this transfer using either VoiceXML or TXML, configure the following settings:

VoiceXML

1. Set the `<transfer>` element with `bridge=false` or `type=blind`.
2. Configure the IVR Profile with the following options:
   - `transfer type`—`1SignalChannel`
   - `transfer option`—`ATTOOBCONFERENCE`

GVPi will execute `ATTOOBConference.xml` before throwing a `connection.disconnect.transfer` event to the VoiceXML application. By default, `ATTOOBConference.xml` is empty.

TXML

1. Set the `<CREATE_LEG_AND_DIAL>` element with `bridge=yes|no`.
2. Configure the IVR Profile with the following options:

- `transfer type`—`1SignalChannel`
- `transfer option`—`ATTOOBCONFERENCE`

GVPi will execute `ATTOOBConference.xml` before hangup. By default, `ATTOOBConference.xml` is empty.

Appendix

B UTF-8 Support for Attached Data

This appendix describes UTF-8 support for attached data.

Note: This manual describes the VoiceXML 2.1 language as implemented by the Genesys Voice Platform (GVP) in versions 7.6 and earlier. For information about the GVP Next Generation Interpreter (NGI), see the *Genesys Voice Platform 8.1 Genesys VoiceXML 2.1 Reference Help.*

This appendix contains the following sections:

Overview

Attached data can be passed from the VoiceXML application to the IVR Server and vice versa. GVP supports attached data in UTF-8 encoding in both directions.

IVR Server to Application

To support UTF-8, the following occurs:

- When setting the `$script-data$`, GVP specifies the `enctype` as `utf8` in the XML page.
- GVP accepts the `$script-data$` UTF-8 data and makes the script data available in a VoiceXML session variable: `session.genesys.script_data`.

If the VoiceXML application does not need to process UTF-8 data, or if it is not expected to receive UTF-8 data from the IVR Server, no change is required in the application.

If the VoiceXML application is required to handle UTF-8 data, the VoiceXML application should access the VoiceXML session variable.

Example

```
<?xml version="1.0" ?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml">
<property name="termchar" value="D" />
<property name="com.telera.speechenabled" value="false" />
<form>
     <var name="ScriptData" expr="Session.scriptData"/>
     <block>
     <submit next="Branching1%2Easp method="post" namelist="ScriptData"/>
     </block>
</form>
```

When setting the `$scripturl$`, the VoiceXML application should remove `$script-data$` from the query string.

Example

```
<?xml version="1.0" ?>
<XMLPage   TYPE="IVR" PAGEID="" SESSIONID=""
   HREF="http://localhost/VXMLStudioSimulation/LegWait1.asp">
<SET  VARNAME="$scripturl$"
VALUE="http://localhost/VXMLStudioSimulation/START.asp?ScriptID=$sid" />
<QUEUE_CALL USR_PARAMS="GenesysRouteDN:1111" />
</XMLPage>
```

Double-Byte Character

To support the UTF-8 double-byte character:

1. In Framework, set the application type T-Server option `[XmlSap]:target-encoding` to `Chinese`.
2. Restart IVR Server.

Appendix

C Passing MRCP Vendor Specific Parameters

This appendix describes how to pass MRCP vendor–specific parameters.

Note: This manual describes the VoiceXML 2.1 language as implemented by the Genesys Voice Platform (GVP) in versions 7.6 and earlier. For information about the GVP Next Generation Interpreter (NGI), see the *Genesys Voice Platform 8.1 Genesys VoiceXML 2.1 Reference Help.*

This appendix contains the following sections:

Overview

GVP supports the sending of application-provided vendor-specific parameters to MRCP ASR and TTS servers.

Note: GVP does not support sending vendor-specific parameters from the MRCP ASR or TTS server back to the VoiceXML application.

Hotword Support

You must edit the `Baseline.xml` file in order for hotword detection with ASR grammar to work. The file is located in the following directory:

```
\SpeechWorks\OpenSpeech Recognizer\config\Baseline.xml
```

In the `baseline.xml` file, change the following value from `0` to `1`.

```
<param name="swiep_mrcp20_compatible_hotword">
   <value>0</value>
</param>
```

You can then enable hotword support by using the `vrm.client.DisableHotWord` parameter in Genesys Administrator. Set the value of this parameter to `false` for MRCP servers (for example, Nuance SWMS/NSS) that support hotword by the vendor-specific parameter `Recognition-Mode` on MRCPv1. For all other servers, leave the default value of the `vrm.client.DisableHotWord` parameter to `true`.

Using Nuance SWMS/NSS, you can use Nuance MRCPv1 extensions to perform `hotword` recognition. To enable `hotword`, use standard VoiceXML and set the `bargeintype` property to `hotword` in the VoiceXML application. As per the VoiceXML specification, `hotword` is enabled by default during transfer.

If `hotword` is not configured on the MRCP ASR server using the `vrm.client.DisableHotWord` parameter, and `bargeintype` is set to `hotword`, the `error.noresource` error is returned to the VoiceXML application. For VoiceXML transfer, `hotword` will not be activated if it is not configured on the MRCP ASR server.

Refer to the Nuance SWMS/NSS documentation for details on the parameters that you can optionally set on SWMS/NSS to fine-tune hotword recognitions.

Note: The SWMS/NSS vendor-specific parameter `Recognition-Mode` must not be set by the VoiceXML application; it is controlled by GVP through the VoiceXML `bargeintype`.

Passing Parameters to ASR Servers

To pass vendor-specific parameters in the application, use the VoiceXML `<property>` element.

Example

```
<property name="swi.rec.channelName" value="FooChannel"> </property>
<property name="swi.rec.applicationName" value="FooApp"> </property>
```

GVP, in turn, sends the vendor-specific parameter to the MRCP server in an MRCP `SET-PARAMS` message.

Example

```
ANNOUNCE rtsp://dev-fry:4900/media/speechrecognizer RTSP/1.0
CSeq: 3
```

```
Session: ECKPFCGLAAAHFFAJAAAAAAAA
Content-Type: application/mrcp
Content-Length: 481
SET-PARAMS 3 MRCP/1.0
dtmf-term-timeout:0
speed-vs-accuracy:50
sensitivity-level:50
recognition-timeout:20000
n-best-list-length:1
speech-incomplete-timeout:3000
confidence-threshold:50
speech-complete-timeout:1000
dtmf-interdigit-timeout:2000
dtmf-term-char:#
no-input-timeout:10000
logging-tag:GenesysLab_Wesley_Wesley_MyApplication_5B93DC84-015C-460F-A944-9AA877AD61C1
vendor-specific-
   parameters:swi.rec.channelName="FooChannel";swi.rec.applicationName="FooApp"
```

Passing Parameters to TTS Servers

To pass vendor-specific parameters in the application, use the VoiceXML `<property>` element and prepend the vendor specific parameter with `com.genesys.tts`.

Example

```
<property name="com.genesys.tts.FooParameter1" value="FooValue1"> </property>
<property name="com.genesys.tts.FooParameter2" value="FooValue2"> </property>
```

GVP, in turn, sends the vendor-specific parameter in an MRCP `SET-PARAMS` message to the MRCP TTS server.

Example

```
ANNOUNCE rtsp://dev-fry:4900/media/speechsynthesizer RTSP/1.0
CSeq: 3
Session: ECKPFCGLAAAHFFAJAAAAAAAA
Content-Type: application/mrcp
Content-Length: 481

SET-PARAMS 3 MRCP/1.0
vendor-specific-parameters: FooParameter1="FooValue1"; FooParameter2=" FooValue2"
```

Appendix

D Key Ahead

This appendix describes Key Ahead with Automatic Speech Recognition (ASR).

Note: This manual describes the VoiceXML 2.1 language as implemented by the Genesys Voice Platform (GVP) in versions 7.6 and earlier. For information about the GVP Next Generation Interpreter (NGI), see the *Genesys Voice Platform 8.1 Genesys VoiceXML 2.1 Reference Help.*

This appendix contains the following sections:

Overview

The Key Ahead feature enables the user to terminate the multiple recognitions by sending a DTMF sequence.

Note: The Key Ahead feature is supported on MRCP only.

DTMF tones will be sent to the ASR server until a recognition result is received. Any remaining digits will be passed on to later recognitions, until no DTMF tones are left.

Key Ahead digits will not be passed on when the VoiceXML application transitions from a page with speech recognition enabled to a page with speech recognition disabled.

The Key Ahead buffer is flushed if any of the following occur:

- A grammar is not matched.
- A spoken grammar is matched.

- A prompt that does not allow the user to barge-in is played.

The following example describes the caller experience with Key Ahead.

Example

```
grammar: 1
grammar: 2
grammar: 1 3
<press 1 for weather, press 2 for sports>
<input>

grammar: 1
grammar: 2
<press 1 for san jose, press 2 for campbell>
<input>
```

Previously, the caller would have pressed `1,` paused, and then pressed `2` for Campbell weather. With Key Ahead enabled, the caller would press `[1 2]` and expect to jump to Campbell weather. That scenario will not work, because the `[1 2]` input will not match the `[1 3]` grammar on the first page, and as a result, a no-match will be thrown. Therefore, the application developer must make sure that he or she designs fixed-length DTMF sequence grammars.

Clearing Key Ahead Buffer

GVP enables VoiceXML applications to control when they clear the Key Ahead buffer.

Note: Clearing Key Ahead buffer is applicable to both ASR and non-ASR.

The VoiceXML interpreter supports Genesys extensions to all of the VoiceXML input elements. An attribute—`clearbuffer`—can be included as part of an `input` element, to suggest that the VoiceXML application should clear the buffer before executing that input element.

The `clearbuffer` attribute is supported in the following VoiceXML input elements:

- `<field>`
- `<record>`
- `<transfer>`
- `<menu>`

The following examples demonstrate how to use the `clearbuffer` attribute in each of the preceding `input` elements.

Example 1

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
CallFlow:

C (Computer):   This is field 1. Please press 1 to go to field 2.

H (Human):      (Press 1 2 3)

C:              This is field 3. Please press 3. (Note that prompt in "field2"
                is skipped because of the buffering of digit 2, while the prompt in
   "field3" is not skipped because the attribute "clearbuffer" is set to "true")

H (Human):      (Press 3)

C (Computer):   You have pressed 3.
-->

<vxml xmlns="http://www.w3.org/2001/vxml"
   xmlns:telera="http://www.telera.com/vxml/2.0/ext/20020430"
   xmlns:genesys="http://www.genesyslab.com/vxml/2.0/ext/20020430"
   xmlns:conf="http://www.w3.org/2001/vxml-conformance" version="2.1">
    <form id="form1">
        <field name="field1" type="digits?length=1">
            <prompt>
                This is field 1. Please press 1 to go to field 2.
            </prompt>
        </field>
        <filled>
            <if cond="field1 == '1'">
                <prompt>
                    You have pressed 1
                </prompt>
                <goto next="#form2"/>
            </if>
        </filled>
    </form>

    <form id="form2">
        <field name="field2" type="digits?length=1">
            <prompt>
                This is field 2. Please press 2 to go to field 3.
            </prompt>
        </field>
        <filled>
            <if cond="field2 == '2'">
                <prompt>
                    You have pressed 2
                </prompt>
```

```
                <goto next="#form3"/>
            </if>
        </filled>
    </form>

    <form id="form3">
        <genesys:field name="field3" clearbuffer="true" type="digits?length=1">
            <prompt>
                This is field 3. Please press 3.
            </prompt>
        </genesys:field>
        <filled>
            <if cond="field3 == '3'">
                <prompt>
                    You have pressed 3
                </prompt>
            </if>
        </filled>
    </form>
</vxml>
```

Example 2

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml"
  xmlns:genesys="http://www.genesyslab.com/vxml/2.0/ext/20020430"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
   http://www.w3.org/TR/voicexml20/vxml.xsd">

<!--
CallFlow:

C (Computer):   For sports press 1, For weather press 2, For movies press 3.

H (Human):      (Press 1 2 3)

C:              You have selected movies. Finally, for sports press 1, For
                weather press 2, For movies press 3. ( Note that prompt in
                "menu1.vxml" and "menu2.vxml" is skipped because of the
                buffering of digit 2, while the prompt in "menu3.vxml" is not skipped
                because the attribute "clearbuffer" is set to "true")

H (Human):      (Press 3)

C (Computer):   You have selected movies.

-->

    <menu>
```

```
        <property name="inputmodes" value="dtmf"/>
        <prompt>
            For sports press 1, For weather press 2, For movies press 3.
        </prompt>
        <choice dtmf="1" next="menu1.vxml"/>
        <choice dtmf="2" next="menu2.vxml"/>
        <choice dtmf="3" next="menu3.vxml"/>
    </menu>
</vxml>
```

menu1.vxml

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
   http://www.w3.org/TR/voicexml20/vxml.xsd">
    <menu>
        <property name="inputmodes" value="dtmf"/>
        <prompt>
            You have selected sports. For sports press 1, For weather press 2, For
   movies press 3.
        </prompt>
        <choice dtmf="1" next="menu1.vxml"/>
        <choice dtmf="2" next="menu2.vxml"/>
        <choice dtmf="3" next="menu3.vxml"/>
    </menu>
</vxml>
```

menu2.vxml

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
   http://www.w3.org/TR/voicexml20/vxml.xsd">
    <menu>
        <property name="inputmodes" value="dtmf"/>
        <prompt>
            You have selected weather. For sports press 1, For weather press 2, For
   movies press 3.
        </prompt>
        <choice dtmf="1" next="menu1.vxml"/>
        <choice dtmf="2" next="menu2.vxml"/>
        <choice dtmf="3" next="menu3.vxml"/>
    </menu>
</vxml>
```

menu3.vxml

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml"
  xmlns:genesys="http://www.genesyslab.com/vxml/2.0/ext/20020430"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
   http://www.w3.org/TR/voicexml20/vxml.xsd">
    <catch event="event.sports">
        <prompt>
            You have selected sports
        </prompt>
        <exit/>
    </catch>
    <catch event="event.weather">
        <prompt>
            You have selected weather
        </prompt>
        <exit/>
    </catch>
    <catch event="event.movies">
        <prompt>
            You have selected movies
        </prompt>
        <exit/>
    </catch>
    <genesys:menu clearbuffer="true">
        <property name="inputmodes" value="dtmf"/>
        <prompt>
           You have selected movies. Finally, for sports press 1, For weather press 2,
   For movies press 3.
        </prompt>
        <choice dtmf="1" event="event.sports"/>
        <choice dtmf="2" event="event.weather"/>
        <choice dtmf="3" event="event.movies"/>
    </genesys:menu>
</vxml>
```

Example 3

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
CallFlow:

C (Computer):   This is field 1. Please press 1 to go to field 2.

H (Human):      (Press 1 2 3)

C:              You have pressed 2. Please record message 3 and then press 3.
```

```
                (Note that prompt in "field2" is skipped because of the buffering
                of digit 2. While the prompt in "record3" is not skipped because the
                attribute "clearbuffer" is set to "true")

H (Human):      (Press 3)

C (Computer):   You have pressed 3.
-->

<vxml xmlns="http://www.w3.org/2001/vxml"
   xmlns:telera="http://www.telera.com/vxml/2.0/ext/20020430"
   xmlns:genesys="http://www.genesyslab.com/vxml/2.0/ext/20020430"
   xmlns:conf="http://www.w3.org/2001/vxml-conformance" version="2.1">
    <form id="form1">
        <field name="field1" type="digits?length=1">
            <prompt>
                This is field 1. Please press 1 to go to field 2.
            </prompt>
        </field>
        <filled>
            <if cond="field1 == '1'">
                <prompt>
                    You have pressed 1
                </prompt>
                <goto next="#form2"/>
            </if>
        </filled>
    </form>

    <form id="form2">
        <field name="field2" type="digits?length=1">
            <prompt>
                This is field 2. Please press 2 to go to field 3.
            </prompt>
        </field>
        <filled>
            <if cond="field2 == '2'">
                <prompt>
                    You have pressed 2
                </prompt>
                <goto next="#form3"/>
            </if>
        </filled>
    </form>

    <form id="form3">
        <genesys:record name="record3" clearbuffer="false">
            <prompt>
                Please record message 3 and then press 3.
            </prompt>
        </genesys:record>
```

```
        <filled>
            <if cond="lastresult$.interpretation == '3'">
                <prompt>
                    You have pressed 3
                </prompt>
            </if>
        </filled>
    </form>
</vxml>
```

Example 4

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
CallFlow:

C (Computer):   This is field 1. Please press 1 to go to field 2.

H (Human):      (Press 1 2 3)

C:              You have pressed 2. This is transfer 3. Please press 3.
                (Note that prompt in "field2" is skipped because of the buffering
                of digit 2. While the prompt in "transfer3" is not skipped because
                the attribute "clearbuffer" is set to "true")

H (Human):      (Press 3)

C (Computer):   You have pressed 3.
-->

<vxml xmlns="http://www.w3.org/2001/vxml"
   xmlns:telera="http://www.telera.com/vxml/2.0/ext/20020430"
   xmlns:genesys="http://www.genesyslab.com/vxml/2.0/ext/20020430"
   xmlns:conf="http://www.w3.org/2001/vxml-conformance" version="2.1">
    <form id="form1">
        <field name="field1" type="digits?length=1">
            <prompt>
                This is field 1. Please press 1 to go to field 2.
            </prompt>
        </field>
        <filled>
            <if cond="field1 == '1'">
                <prompt>
                    You have pressed 1
                </prompt>
                <goto next="#form2"/>
            </if>
        </filled>
    </form>
```

```
    <form id="form2">
        <field name="field2" type="digits?length=1">
            <prompt>
                This is field 2. Please press 2 to go to field 3.
            </prompt>
        </field>
        <filled>
            <if cond="field2 == '2'">
                <prompt>
                    You have pressed 2
                </prompt>
                <goto next="#form3"/>
            </if>
        </filled>
    </form>

    <form id="form3">
        <genesys:transfer name="xfer3" type="bridge" dest="123" clearbuffer="true">
            <grammar type="application/srgs+xml" root="r2" version="1.0" mode="dtmf">
                <rule id="r2" scope="public">
                    <one-of>
                        <item>1</item>
                        <item>2</item>
                        <item>3</item>
                    </one-of>
                </rule>
            </grammar>
            <prompt>
                This is transfer 3. Please press 3.
            </prompt>
        </genesys:transfer>
        <filled>
            <if cond="lastresult$.interpretation == '3'">
                <prompt>
                    You have pressed 3
                </prompt>
            </if>
        </filled>
    </form>
</vxml>
```

Note: GVPi can only clear the buffer for digits already received by the caller. Depending on timing, the interpreter may have already transitioned to another wait state and cleared the buffer while the caller is still entering additional digits.

Consider "Example 4" on page 98. If GVPi and MCP process the caller's input of `2` and proceed to clear the buffer in `<transfer>` in `form3` before the caller has pressed `3`, the user input of `3` would not be cleared, the `You have pressed 2. This is transfer 3. Please press 3.` prompt might not be heard or only partially heard as the DTMF 3 barges in on the prompt, and the call would be transferred without needing to press `3` again.

Timing and performance may vary based on factors such as (but not limited to):

- Speed of caller's input.
- Speed of caller's phone.
- Speed of MCP hardware.
- Speed of response from recognizer (native DTMF processing is usually faster than offboard recognition).
- Network latency.

Appendix

E SIP Headers

This appendix describes how to propagate SIP Header values to the VoiceXML application on Media Control Platform (MCP).

Note: This manual describes the VoiceXML 2.1 language as implemented by the Genesys Voice Platform (GVP) in versions 7.6 and earlier. For information about the GVP Next Generation Interpreter (NGI), see the *Genesys Voice Platform 8.1 Genesys VoiceXML 2.1 Reference Help.*

This appendix contains the following section:

Propagation of Headers

MCP passes certain SIP headers to the VoiceXML application when required.

P-Asserted-Identity

The P-Asserted-Identity header is used when a SIP server (proxy server) asserts the private identity of a user. When receiving an `INVITE` that contains the P-Asserted-Identity header, MCP passes the value to the VoiceXML application.

Note: GVPi only checks for this header in an incoming `INVITE` message.

Call-ID

Every INVITE message must have a Call-ID header. When MCP receives an INVITE with this header, GVPi passes it to the VoiceXML application.

Accessing Header Values

GVPi collects all the values for the SIP headers and presents it as a comma separated list to the VoiceXML application. The VoiceXML application will access the headers using the `session.connection` variables.

To access the P-Asserted-Identity header, use `session.connection.protocol.sip.headers["p-asserted-identity"]`.

To access the Call-ID header, use `session.connection.protocol.sip.headers["call-id"]`.

If these headers are not available in the INVITE message, or the MCP is not configured to present the headers to the VoiceXML application, the session.connection variables are set to undefined.

Example VoiceXML

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
xmlns:conf="http://ww.w3.org/2002/vxml-conformance" version="2.1">

<form>
   <block>
      <prompt>
         The P-Asserted-Identity header value is
         <value expr='session.connection.protocol.sip.headers["p-asserted-identity"]'/>.
         The Call-ID header value is
         <value expr='session.connection.protocol.sip.headers["call-id"]'/>.
      </prompt>
   </block>
</form>

</vxml>
```

The VoiceXML application can also access these headers using the following `$variables`:

- P-Asserted-Identity—`$sip.p-asserted-identity`
- Call-ID—`$sip.call-id`

These `$variables` are available in the VoiceXML application's first page, and can be accessed by the `$start-ivr-url$` query string variable.

Appendix

F Best Practices

This appendix summarizes some of the techniques that can be used to develop efficient VoiceXML applications.

Note: This manual describes the VoiceXML 2.1 language as implemented by the Genesys Voice Platform (GVP) in versions 7.6 and earlier. For information about the GVP Next Generation Interpreter (NGI), see the *Genesys Voice Platform 8.1 Genesys VoiceXML 2.1 Reference Help.*

This appendix contains the following sections:

Overview

The following is a high-level list of recommended best practices:

- Careful caching of resources.
- Reduce the number of page transitions (that do not collect any caller inputs).
- Reduce the number of ECMAScripts used.
- Reduce the number of global variables.
- Reduce the usage of inline ECMAScripts.
- Reduce the time required to compile the VoiceXML page.
  - Keep the root document size small.
- Use `fetchhint` values (`safe` or `prefetch`) properly.
- During production deployment:
  - Disable `<log>` tag.
  - Disable ASR wave capture.

    - Use full duplex recording with care.
- Avoid using local resources on the GVP server.
- Ensure that `Expires` header is present for all external grammars used by the ASR server.
- Avoid using inline grammars.
- Pre-compile grammars when possible.
- Avoid frequently changing the application root.
- Avoid keeping audio files on the GVP servers.
- Down sample audio files to 8K samples, 1 byte Mono.
- Adjust timeout to a reasonable amount depending upon the data being collected.
- Resolve ambiguity—ideally one call input should match only one grammar.
- Renew the `Expires` time for resources when `If-Modified-Since` request made.
- Avoid using the `<break>` tag for pause between audio prompts.
- Avoid interleaving of small TTS prompts with audio prompts.
- Understand prompt queuing.
- Tune grammars and prompts for speech applications.
- Load test applications before going live.

Each of these recommendations is explained in detail in the following sections.

Note: Certain techniques will be more effective than others depending upon the nature of the application being developed. Genesys highly recommends that early benchmarking of smaller representative applications be conducted, which will indicate the kind of performance improvements that can be achieved.

Application Guidelines

This section explains the best practice recommendations in detail.

Careful Caching of Resources

Caching can improve application performance because it avoids sending extra HTTP fetch requests for resources that have not changed since the last time they were fetched.

Many times it is necessary during the development of applications to ensure that the resources on the application servers (for example, grammars, prompts, ECMAscripts, static VoiceXML scripts, and so on) are fetched every time

since they are likely to change. Once the application development is complete and has gone through rigorous testing, these resources are much less likely to change. This is the right time to examine the settings that control whether or not a resource is cached and for how long.

VoiceXML 2.0 (RFC 2616) describes the manner in which HTTP 1.1 caching behaves. The `maxage` and `maxstale` attributes of some of the selected VoiceXML tags control the manner in which cached copy is used. The `Expires:` header plays an important role indicating to the GVP VoiceXML platform about when this resource can be considered `stale`.

Genesys recommends that the `Expires:` header be set to a large value (for example, `86400`) for certain types of resource such as prompt files, grammar files (compiled or static source files), static VoiceXML files, and ECMAScript files. This ensures that the users of these resources avoid fetching potentially large resources from the network; therefore, improving application performance.

Reduce Number of Page Transitions

Each time a VoiceXML page is received by the VoiceXML interpreter, it is compiled and transformed into a state machine representation before it can be interpreted. It is important to avoid unnecessary page transitions, especially if the pages do not collect any inputs from the caller.

For example, consider the following VoiceXML page:

```
<?xml version="1.0"?>
<vxml version="2.0">
<form id="menu">
  <block>
<submit next= "http://1.2.3.4:8080/app/jsp/aLocal.jsp?key1= val1&key2=val2&key3=val3"/>
  </block>
</form>
</vxml>
```

It may be possible to invoke the `aLocal.jsp` while on the application server with the right parameters instead of delivering to the VoiceXML interpreter, having it compile the VoiceXML page and then eventually do a HTTP submit request. This not only saves valuable system resources on the GVP VoiceXML platform, but it also improve the latencies experienced by the caller.

Reduce Number of ECMAScripts

Although the ability to use ECMAScript inside VoiceXML is appropriate for executing data driven application logic without leaving the VoiceXML browser, it does create overhead because GVP uses a third party ECMAScript engine. Reduce the number of ECMAScripts used in an application, as overuse of ECMAScripts can adversely affect the performance of the VoiceXML interpreter impacting the application performance.

The application must be tuned to optimally balance the processing between the VoiceXML browser and the Web server. In order to use the VoiceXML browser resources efficiently, logic that can be handled on the server should be controlled in the dynamic web pages instead of executing them on the VoiceXML browser.

There is no additional ECMAScript overhead when the VoiceXML page is compiled; however, each individual request to an ECMAScript (either as a condition, expression or actual script) is sent to the ECMAScript engine for processing. The ECMAScript engine does not maintain internal cache of its own, and the script is compiled and executed each time it is invoked. For example, if an expression is executed more than once in an application flow, the expression will be evaluated each time.

When using ECMAScripts, it is recommended to define local variables to hold the expression variables. Use these variables if the same expression is computed multiple times. Use functions for commonly used operations instead of using inline scripts.

Reduce Number of Global Variables

All VoiceXML variables are ECMAScript definitions defined inside the ECMAScript engine. Memory is consumed for each variable created. Each time a variable is created, or a new value is assigned to it, ECMAScript overhead is incurred. Too many global variables defined in the application root increases the memory requirements which impacts application performance, and the total number of concurrent calls that GVP can handle.

Reduce Usage of Inline ECMAScripts

Internal and external ECMAScripts are executed in the same way. However, using `Expires` headers will help reduce the size of the VoiceXML page resulting in faster page fetches.

Reduce Time Required to Compile VoiceXML Page

When a VoiceXML page is fetched, the entire page is compiled even if a single line of a ten thousand line page is executed. Therefore, avoid having multiple forms within the same page as they may not all be used. However, having forms on separate pages can introduce fetch delays.

Genesys recommends that the maximum size of VoiceXML pages be under 100K for optimal performance of the platform.

The following code snippet shows an inefficient application design. There are six forms on the same page but only three will be used in a call. The rest are compiled but not used resulting in wasted CPU time.

```
<?xml version="1.0"?>
<vxml version="2.0">
<form id="menu">
  <field name="course">
    <prompt>
      Pick a course for its description.
      Example CS 100.
    </prompt>
                <grammar>
      CS 100 {course:cs100} | CS 121 {course:cs121} |
      CS 122 {course:cs122} | CS 123 {course:cs123} |
      CS 124 {course:cs124}
    </grammar>
    <filled>
      <goto expr="'#'+course"/>
    </filled>
  </field>
</form>
<form id="cs100">
  <block>
    CS 100.
    Introduction to Computer Usage. This course gives an introduction
                to using personal computer hardware and software...
    <!-- loop back -->
    <goto next="#menu"/>
  </block>
</form>
<form id="cs121">
  <block>
    CS 121.
    Introduction to Object-Oriented Programming. This course teaches
    the fundamental concepts of problem solving using a computer...
  </block>
</form>
<form id="cs122">
  <block>
    CS 122.
    Principles of Program Design. This course teaches the fundamental
    concepts of object-oriented analysis and design...
  </block>
</form>
<form id="cs123">
  <block>
    CS 123.
    Developing Programming Principles. This course gives a review of
    fundamental programming concepts and their application in Java...
  </block>
</form>
<form id="cs124">
  <block>
    CS 124.
```

```
    Introduction to Software Development. This course is an introduction
    to basic concepts of computer science, including the paradigms of
                theory, abstraction, and design...
  </block>
</form>
</vxml>
```

The following code snippet shows a more efficient application. By moving each form into its own page, a form will only be compiled if it is going to be executed. The other unused forms will not be compiled eliminating unnecessary CPU cycles.

It is more efficient to use <subdialog> and <return> tags rather than <goto> tags to avoid transitioning back to the main menu for it to be recompiled. With a <subdialog>, the main menu page is preserved and upon return will not be recompiled.

```
<?xml version="1.0"?>
<vxml version="2.0">
<form id="menu">
  <field name="course">
    <prompt>
      Pick a course for its description.
      Example CS 100.
    </prompt>
    <grammar>
      CS 100 {course:cs100} | CS 121 {course:cs121} |
      CS 122 {course:cs122} | CS 123 {course:cs123} |
      CS 124 {course:cs124}
    </grammar>
    <filled>
      <if cond="course=='cs100'">
        <goto nextitem="loop"/>
      <else/>
        <goto expr="course+'.vxml'"/>
      </if>
    </filled>
  </field>
  <subdialog name="loop" src="cs100.vxml">
    <filled>
      <clear/>
      <goto next="#menu"/>
    </filled>
  </subdialog>
</form>
</vxml>

[ cs100.vxml ]

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

```
<vxml version="2.0">
<form id="cs100">
  <block>
    CS 100.
    Introduction to Computer Usage. This course gives an introduction
                to using personal computer hardware and software...
    <!-- loop back -->
    <return/>
  </block>
</form>
</vxml>

[ cs121.vxml ]

<?xml version="1.0"?>
<vxml version="2.0">
<form id="cs121">
  <block>
    CS 121.
    Introduction to Object-Oriented Programming. This course teaches
    the fundamental concepts of problem solving using a computer...
  </block>
</form>
</vxml>
```

Use fetchhint Values (safe or prefetch) Properly

Fetching unnecessary resources is expensive. You can greatly improve the performance of your system and applications by fetching resources that are actually used.

The default value for the `fetchhint` property (for audio/scripts/grammars) is `prefetch` which means that all resources on the page start to be fetched when the page starts. The page runs concurrently although certain fetched resources are not required due to the call flow. For example, error message audio files are rarely used since errors are not normally expected. In this case, setting `fetchhint` to `safe` would be a better choice.

When `fetchhint` is set to `safe`, resources are fetched when they are needed. In this case, fetching is not concurrent with the page, so there is the potential for latency. However, if the resource is not very large, this delay will not be noticeable to the caller. Using values of `prefetch` and `safe` appropriately can improve the performance of an application dramatically.

The following code snippet is an example of a inefficient application:

```
<?xml version="1.0"?>
<vxml version="2.0">
<!--All audio resources are loaded at the begining of the page-->
<property name="audiofetchhint" value="prefetch"/>
<form>
```

```
  <field name="question" type="digits">
    <prompt>
      <audio src="prompt.wav">
        Please say a number.
      </audio>
    </prompt>
    <!--Most of these audio files are not used-->
    <noinput>
      <audio src="noinput.wav">
        I did not hear you.
      </audio>
    </noinput>
    <nomatch>
      <audio src="nomatch.wav">
        That is not a number.
      </audio>
    </nomatch>
    <help>
      <audio src="help.wav">
        Please say a number.
      </audio>
    </help>
    <filled>
      You said <value expr="question"/>.
    </filled>
  </field>
</form>
</vxml>
```

Example 3b is an example of a more efficient application:

```
<?xml version="1.0"?>
<vxml version="2.0">
<form>
  <field name="question" type="digits">
    <prompt>
      <audio src="prompt.wav">
        Please say a number.
      </audio>
    </prompt>
    <!--Most of these audio files are not used-->
    <noinput>
      <audio src="noinput.wav" fetchhint="safe">
        I did not hear you.
      </audio>
    </noinput>
    <nomatch>
      <audio src="nomatch.wav" fetchhint="safe">
        That is not a number.
      </audio>
```

```
    </nomatch>
    <help>
      <audio src="help.wav" fetchhint="safe">
        Please say a number.
      </audio>
    </help>
    <filled>
      You said <value expr="question"/>.
    </filled>
  </field>
</form>
</vxml>
```

Production Deployment

Normally, during development logging levels will be configured in the platform or application to provide the most detailed information for debugging. This may be detrimental to the platform and/or application performance though is not considered an issue. In a production environment, the performance impact of those logging levels could be considered unacceptable as large amounts of data will be written to files. Genesys recommends that the `<log>` tag should be used minimally in production, and turned on only for collecting speech tuning metrics. In addition, the platform logging levels must be kept at the minimum level; no greater than `Standard` during normal operations of a live deployment (the default level is `Interaction`).

Sometimes, detailed logging is required for debugging, tuning, monitoring, or for other reasons. When using higher log levels, try to localize their use, rather than enabling them globally throughout an entire application. When using full-duplex recording for monitoring, use in a limited (spot check) fashion. In any case, collected files should be deleted when they are no longer required, or moved to another server or SAN as soon as possible if they need to be archived.

In a production environment, the following is also recommended to avoid use of significant system resources that may adversely affect system performance:

- Disable ASR wave capture.
- Use full duplex (transactional) recording with caution.

Ensure That Expires Header is Present

ASR engines are responsible for fetching all of the external grammars. They maintain their own cache and depend on the `Expires` header to determine how long to use the cached copy. If the `Expires` header is not set, or set incorrectly, the ASR engine will continue to re-fetch the external grammar causing unnecessary use of system and network resources.

Avoid Using Inline Grammars

When using an inline grammar a VoiceXML interpreter has to generate a `temporary` grammar that is eventually passed on to the ASR for recognizing caller inputs. Even though the VoiceXML page may be cached, the generation of the temporary grammar has to be repeated each time that page is executed.

Construct an external grammar and use that instead of the inline grammar. This will not only reduce the load on VoiceXML interpreter, but also will cause the ASR to use a cached copy (if the `Expires` header is set correctly).

Precompile Grammars When Possible

Before a grammar can be executed, the ASR engine first compiles the grammar and caches for all subsequent requests for that grammar. Most ASR engines provide an option for pre-compiling the grammar files. If the application is using large static grammar files, it will be better to pre-compile the grammars and place the pre-compiled version on the web server. This will help in speeding up the response time from the ASR engine when the calls are made.

Do not generate unnecessary dynamic grammars. If the grammar will not change, develop it as a static grammar (and precompile if large) and reference via URL.

Avoid Frequent Modification of Application Root Document

When the application root of the VoiceXML dialog is modified, a new application context is created. To reduce the overhead of creating this context, change the application root document only when absolutely required, (for example, when a subdialog is created or when control is transferred from one application to another).

If the application design is using a root document, ensure that it is applied to all pages. A compilation penalty occurs if the application fetches pages that have application root document and pages that do not.

Avoid Keeping Audio Files on GVP Servers

GVPi does not offer a performance improvement when the audio files are kept locally on the GVP platform. GVPi uses HTTP to fetch the audio files and fetching audio files from the same GVP machine will place additional burden on the platform due to processor resource competition.

Reduce Sample Audio Files to 8K Samples/Sec, 8bit Mono

Reducing the audio file type to 8000 samples/second, 8 bits Mono reduces the amount of data that needs to be fetched by the GVP. Using this format, less

work needs to be done by GVP for processing the audio file due to the headerless format. For all audio formats with headers, GVP first reads the header for each audio file before playback, thereby adding more processing overhead.

Adjust Timeout to a Reasonable Amount Depending on Data Being Collected

It may be necessary to allow callers extra time to say long input strings such as credit card numbers. It is therefore a good practice to use a different timeout value for such cases. Make sure that timeout is not set to zero for user input fields, as in this case the caller will not be able to enter user input at all. This could result in difficult to troubleshoot problems especially for speech applications.

Resolve Grammar Ambiguity

The ASR engine will work efficiently if the grammars are designed in a manner where there is no ambiguity, that is, a caller utterance matches only one item in the active grammar at any given point in time. This will also simplify the VoiceXML application since it does not have to handle any ambiguous recognition results.

Renew Expires Time for Resources When If-Modified-Since Request Made

It may be possible to renew the `Expires` header value when an `If-Modified-Since` query is made for the proxy after a resource is considered `STALE`. This will extend the time after which the resource is considered `STALE`.

Avoid Using <break> tag for Pause Between Audio Prompts

The `<break>` tag is used for introducing pauses in prompts. It uses the TTS engine to generate a small silence or cadence break. If the break tag is interspersed with audio file playback, each request is sent as a separate request to the TTS engine and this increases the processing overhead. Use a prerecorded silence audio file instead of `<break>` tag for pauses between audio prompts.

Avoid Interleaving of Small TTS Prompts with Audio Prompts

SSML tags inside a `<prompt>` are batched and sent to the TTS engine. If a prompt or set of consecutive prompts contain interleaving of SSML and `<audio>` tag for playing pre-recorded audio files, each section of SSML is sent

to TTS engine separately. This will result in processing latencies. Avoid intermixing small TTS and pre-recorded audio prompts.

The following is an example of a non-efficient application:

```
<?xml version="1.0"?>
<vxml version="2.0">
<block>
    <prompt>
      <audio src="TransactionHistory.vox"/>
   <audio src="credit.vox"/>
   500 dollars
   <audio src="on.vox"/>
   Jan 20, 2007.
   <audio src="debit.vox"/>
   400 dollars
   <audio src="on.vox"/>
   Jan 21, 2007.
   <audio src="debit.vox"/>
   700 dollars
   <audio src="on.vox"/>
   Jan 22, 2007.
   <audio src="credit.vox"/>
   345 dollars
   <audio src="on.vox"/>
   Jan 25, 2007.
    </prompt>
  </block>
</form>
</vxml>
```

In this case each SSML snippet like `700 dollars Jan 25, 2007` is sent as separate request to the TTS Engine.

The following is an example of a more efficient application:

```
<?xml version="1.0"?>
<vxml version="2.0">
<block>
    <prompt>
      <audio src="TransactionHistory.vox"/>
   Credit 500 dollars on Jan 20, 2007.
   Debit 400 dollars on Jan 21, 2007.
   Debit 700 dollars on Jan 22, 2007.
   Credit 345 dollars on Jan 25, 2007.
    </prompt>
  </block>
</form>
</vxml>
```

Understand Prompt Queuing

At any given time, the VoiceXML interpreter is in one of two states, waiting for input in a field item, or transitioning between field items in response to an input. The interpreter is in the transitioning state during a <block>, <catch>, <filled> or other non-field element as well as while it is transitioning between elements (even across pages).

Prompts are queued while the interpreter is in the transitioning state.

Queued prompts are played either:

- when the interpreter reaches the next waiting state (for example, a <field>, <initial>, or <choice>/<menu>), at which point all queued prompts are played and the interpreter listens for input simultaneously—PromptAndCollect.
- when the interpreter reaches an <exit>, <transfer>, <record>, or <object>, at which point all queued prompts are played before the element is executed—PromptOnly.
- when the interpreter begins fetching a resource (such as a document) using <submit>, <goto>, or <subdialog>, with fetchaudio specified. In this case, all queued prompts are played—PromptOnly—and then the fetchaudio is played until the fetch completes.

Note: In the PromptOnly state, no speech input is accepted. DTMF input is allowed, provided that prompt barge-in is enabled.

If multiple prompts are queued before a field input, the timeout/bargeintype of the last prompt is used.

Tune Grammars and Prompts for Speech Applications

Compared to DTMF applications, speech applications are considerably more complex and require different techniques. For successful voice applications, grammar tuning is essential after representative call data has been collected. If the grammars are not properly tuned, it can result in poor customer experience with high opt-out or drop rates. Use utterance capture recording for the call data capture to identify application areas with high amount of misrecognitions. The utterances may uncover input that callers are frequently saying but are not covered in the grammars. You may also have to take caller accents into account and add alternative pronunciations for some grammar items. Additionally, confidence levels may need to be adjusted to reduce both false recognitions and high rejection rates.

Similarly prompts also need to be well designed so that they guide the callers to say the correct input. If callers are getting stuck in the call-flow causing high misrecognition rates, evaluate the prompts to make sure they are intuitively directing the callers to speak the words as expected by the grammar.

Load Test Applications Before Production Operations

A GVP server is configured to service a calculated number of concurrent callers. This affects the number of ports that are in a single server before callers perceive latency or the expected duration of the call time lengthens. The more the latencies and durations increase, the less servicing ports available can deliver the expected call volumes. Undersized ports could lead to busy signals turning away business opportunity or affecting customer satisfaction.

The port calculations take into consideration the application design, component distribution, platform characteristics, web-server performance and other external dependencies. Each of these variables could greatly vary the ports supported by an individual GVP server.

All calculations are theory and produce theoretical results. There is no replacement for not evaluating the true potential of the GVP server by load testing. Use a bulk call generation testing service, such as Empirix, that will exercise both the GVP and the Web server application design and configuration.

Additionally, monitor the memory and CPU utilization on the GVP platform, web server, and if implemented, the speech server. For most application you will have to tune the web server for performance. Typically you will have to tune the worker threads/process pool, memory size, session timeout, connection recycle time.

If the application is experiencing memory leaks, consider recycling the threads/processors of the web server at periodic intervals. Do not enable excessive log levels on the web server side as that will impact overall performance of the web server. For logging, if possible, write the logs on a separate disk to reduce overall impact on disk I/O for the primary disk from which the web server pages will be served. When using Tomcat with Apache, take special care to tune the Apache Tomcat Connector (`mod_jk` or `AJP`) as the default settings for the connector do not perform well under load. Refer to the third-party web server documentation for details on performance tuning aspects.

If using databases or backend enterprise services, consider the performance of the database and Enterprise servers under load. Typically you will have to tune the connection pool size for the databases. If the database requests are taking long time, consider SQL query and database indexing optimizations. Work with your database administrator for these changes.

In addition to the automated load tests, perform actual human load tests with live people making simultaneous calls or do both. This will help in identifying areas in the application where caller perceived latencies are high and pin point locations that need to be optimized.

Related Documentation Resources

The following resources provide additional information that is relevant to this software. Consult these additional resources as necessary.

Management Framework

- *Framework 8.0 Deployment Guide,* which provides information about configuring, installing, starting, and stopping Framework components.
- *Framework 8.0 Genesys Administrator Deployment Guide,* which provides information about installing and configuring Genesys Administrator.
- *Framework 8.0 Genesys Administrator Help,* which provides information about configuring and provisioning contact center objects by using the Genesys Administrator.
- *Framework 8.0 Configuration Options Reference Manual,* which provides descriptions of the configuration options for Framework components.

SIP Server

- *Framework 8.0 SIP Server Deployment Guide,* which provides information about configuring and installing SIP Server.

Genesys Voice Platform

- *Genesys Voice Platform 8.1 Deployment Guide,* which provides information about installing and configuring Genesys Voice Platform (GVP).
- *Genesys Voice Platform 8.1 User's Guide,* which provides information about configuring, provisioning, and monitoring GVP and its components.
- *Genesys Voice Platform 8.1 Troubleshooting Guide,* which provides troubleshooting methodology, basic troubleshooting information, and troubleshooting tools.

- *Genesys Voice Platform 8.1 SNMP and MIB Reference,* which provides information about all of the Simple Network Management Protocol (SNMP) Management Information Bases (MIBs) and traps for GVP, including descriptions and user actions.
- *Genesys Voice Platform 8.1 Genesys VoiceXML 2.1 Reference Help,* which provides information about developing Voice Extensible Markup Language (VoiceXML) applications. It presents VoiceXML concepts, and provides examples that focus on the GVP Next Generation Interpreter (NGI) implementation of VoiceXML.
- *Genesys Voice Platform 8.1 Application Migration Guide,* which provides detailed information about the application modifications that are required to use legacy GVP 7.6 voice and call-control applications in GVP 8.1.
- *Genesys Voice Platform 8.1 CCXML Reference Manual,* which provides information about developing Call Control Extensible Markup Language (CCXML) applications for GVP.
- *Genesys Voice Platform 8.1 Configuration Options Reference,* which replicates the metadata available in the Genesys provisioning GUI, to provide information about all the GVP configuration options, including descriptions, syntax, valid values, and default values.
- *Genesys Voice Platform 8.1 Metrics Reference,* which provides information about all the GVP metrics (VoiceXML and CCXML application event logs), including descriptions, format, logging level, source component, and metric ID.

Voice Platform Solution

- *Voice Platform Solution 8.1 Integration Guide,* which provides information about integrating GVP, SIP Server, and, if applicable, IVR Server.

Composer Voice

- *Composer 8.0 Deployment Guide,* which provides installation and configuration instructions for Composer.
- *Composer 8.0.3 Help,* which provides online information aobut using Composer, an Integrated Development Environment used to develop applications for GVP and Universal Routing.

Open Standards

- *W3C Voice Extensible Markup Language (VoiceXML) 2.1, W3C Recommendation 19 June 2007,* which is the World Wide Web Consortium (W3C) VoiceXML specification that GVP NGI supports.

- *W3C Voice Extensible Markup Language (VoiceXML) 2.0, W3C Recommendation 16 March 2004,* which is the W3C VoiceXML specification that GVP supports.
- *W3C Speech Synthesis Markup Language (SSML) Version 1.0, Recommendation 7 September 2004,* which is the W3C SSML specification that GVP supports.
- *W3C Voice Browser Call Control: CCXML Version 1.0, W3C Working Draft 29 June 2005,* which is the W3C CCXML specification that GVP supports.
- *W3C Semantic Interpretation for Speech Recognition (SISR) Version 1.0, W3C Recommendation 5 April 2007,* which is the W3C SISR specification that GVP supports.
- *W3C Speech Recognition Grammar Specification (SRGS) Version 1.0, W3C Recommendation 16 March 2004,* which is the W3C SRGS specification that GVP supports.

Genesys

- *Genesys Technical Publications Glossary,* which ships on the Genesys Documentation Library DVD and which provides a comprehensive list of the Genesys and computer-telephony integration (CTI) terminology and acronyms used in this document.
- *Genesys Migration Guide*, which ships on the Genesys Documentation Library DVD, and which provides documented migration strategies for Genesys product releases. Contact Genesys Technical Support for more information.
- Release Notes and Product Advisories for this product, which are available on the Genesys Technical Support website at `http://genesyslab.com/support`.

Information about supported hardware and third-party software is available on the Genesys Technical Support website in the following documents:

- *Genesys Supported Operating Environment Reference Manual*
- *Genesys Supported Media Interfaces Reference Manual*

For additional system-wide planning tools and information, see the release-specific listings of System Level Documents on the Genesys Technical Support website, accessible from the `system level documents by release` tab in the Knowledge Base `Browse Documents` Section.

Genesys product documentation is available on the:

- Genesys Technical Support website at `http://genesyslab.com/support`.
- Genesys Documentation Library DVD, which you can order by e-mail from Genesys Order Management at `orderman@genesyslab.com`.

Document Conventions

This document uses certain stylistic and typographical conventions—introduced here—that serve as shorthands for particular kinds of information.

Document Version Number

A version number appears at the bottom of the inside front cover of this document. Version numbers change as new information is added to this document. Here is a sample version number:

80fr_ref_06-2008_v8.0.001.00

You will need this number when you are talking with Genesys Technical Support about this product.

Screen Captures Used in This Document

Screen captures from the product graphical user interface (GUI), as used in this document, may sometimes contain minor spelling, capitalization, or grammatical errors. The text accompanying and explaining the screen captures corrects such errors *except* when such a correction would prevent you from installing, configuring, or successfully using the product. For example, if the name of an option contains a usage error, the name would be presented exactly as it appears in the product GUI; the error would not be corrected in any accompanying text.

Type Styles

Table 18 describes and illustrates the type conventions that are used in this document.

Table 18: Type Styles

Type Style	Used For	Examples
Italic	• Document titles • Emphasis • Definitions of (or first references to) unfamiliar terms • Mathematical variables Also used to indicate placeholder text within code samples or commands, in the special case where angle brackets are a required part of the syntax (see the note about angle brackets on page 121).	Please consult the *Genesys Migration Guide* for more information. Do *not* use this value for this option. A *customary and usual* practice is one that is widely accepted and used within a particular industry or profession. The formula, $x + 1 = 7$ where x stands for . . .
Monospace font (Looks like `teletype` or `typewriter text`)	All programming identifiers and GUI elements. This convention includes: • The *names* of directories, files, folders, configuration objects, paths, scripts, dialog boxes, options, fields, text and list boxes, operational modes, all buttons (including radio buttons), check boxes, commands, tabs, CTI events, and error messages. • The values of options. • Logical arguments and command syntax. • Code samples. Also used for any text that users must manually enter during a configuration or installation procedure, or on a command line.	Select the `Show variables on screen` check box. In the `Operand` text box, enter your formula. Click `OK` to exit the `Properties` dialog box. T-Server distributes the error messages in `EventError` events. If you select `true` for the `inbound-bsns-calls` option, all established inbound calls on a local agent are considered business calls. Enter `exit` on the command line.
Square brackets ([ ])	A particular parameter or value that is optional within a logical argument, a command, or some programming syntax. That is, the presence of the parameter or value is not required to resolve the argument, command, or block of code. The user decides whether to include this optional information.	`smcp_server -host [/flags]`
Angle brackets (< >)	A placeholder for a value that the user must specify. This might be a DN or a port number specific to your enterprise. **Note:** In some cases, angle brackets are required characters in code syntax (for example, in XML schemas). In these cases, italic text is used for placeholder values.	`smcp_server -host <confighost>`

Index

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W