GENESYS®
AN ALCATEL-LUCENT COMPANY

Genesys Voice Platform 7.6

VoiceXML 2.1

Reference Manual

About Genesys

Alcatel-Lucent's Genesys solutions feature leading software that manages customer interactions over phone, Web, and mobile devices. The Genesys software suite handles customer conversations across multiple channels and resources—self-service, assisted-service, and proactive outreach—fulfilling customer requests and optimizing customer care goals while efficiently using resources. Genesys software directs more than 100 million customer interactions every day for 4000 companies and government agencies in 80 countries. These companies and agencies leverage their entire organization, from the contact center to the back office, while dynamically engaging their customers. Go to 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.

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 following regional numbers:

Region	Telephone	E-Mail
North and Latin America	+888-369-5555 or +506-674-6767	support@genesyslab.com
Europe, Middle East, and Africa	+44-(0)-118-974-7002	support@genesyslab.co.uk
Asia Pacific	+61-7-3368-6868	support@genesyslab.com.au
Japan	+81-3-6361-8950	support@genesyslab.co.jp

Prior to contacting technical support, please refer to the *Genesys Technical Support Guide* for complete contact information and procedures.

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: 76gvp_ref_xml_11-2010_v7.6.401.00

Table of Contents

Preface

Welcome to the *Genesys Voice Platform 7.6 VoiceXML 2.1 Reference Manual*. This manual provides information about developing voice applications with Voice Extensible Markup Language (VoiceXML) 2.1 on the Genesys Voice Platform (GVP). It presents VoiceXML 2.1 concepts and describes the platform extensions on the GVP.

This document is valid only for the 7.6 release(s) of this product.

Note: For releases of this document created for other releases of this product, please 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 provides an overview of this document, identifies the primary audience, introduces document conventions, and lists related reference information:

Intended Audience

This guide, primarily intended for system integrators and administrators, assumes 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.

Chapter Summaries

In addition to this preface, this document contains the following chapters and an appendixes:

- Chapter 1, “Overview,” on page 13, introduces VoiceXML and describes the GVP platform specifics.
- Chapter 2, “Platform Extensions,” on page 33, describes the platform extensions for VoiceXML 2.1.
- Chapter 3, “Reference for Call Control Elements,” on page 63, describes the call control element extensions that the platform supports.
- Appendix A, “Transfer Scripts with DTMF Base,” on page 93, provides information on how to develop transfer scripts to support the TransferConnect feature.
- Appendix B, “UTF-8 Support for Attached Data,” on page 99, describes the UTF-8 support and provides a VoiceXML example.
- Appendix C, “Passing MRCP Vendor Specific Parameters,” on page 103, describes how to pass MRCP vendor-specific parameters.
- Appendix D, “Key Ahead,” on page 107, describes the Key Ahead feature and provides VoiceXML examples.
- Appendix E, “SIP Headers,” on page 117, describes how to propagate SIP Header values to VoiceXML applications.
- Appendix F, “Application Developer Note,” on page 119, provides additional information for application developers.
- Appendix G, “Best Practices,” on page 125, summarizes some of the techniques that can be used to develop efficient VoiceXML applications.

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:

75fr_ref_09-2006_v7.6.000.00

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

Type Styles

Italic

In this document, italic is used for emphasis, for documents' titles, for definitions of (or first references to) unfamiliar terms, and for mathematical variables.

Examples:

- Please consult the *Genesys Migration Guide* for more information.
- *A customary and usual practice* is one that is widely accepted and used within a particular industry or profession.
- Do *not* use this value for this option.
- The formula, $x + 1 = 7$ where x stands for . . .

Monospace Font

A monospace font, which looks like `teletype or typewriter text`, is used for 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; and code samples.

Examples:

- Select the `Show variables on screen` check box.
- Click the `Summation` button.
- In the `Properties` dialog box, enter the value for the host server in your environment.
- In the `Operand` text box, enter your formula.
- Click `OK` to exit the `Properties` dialog box.
- The following table presents the complete set of error messages T-Server® distributes 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.

Monospace is also used for any text that users must manually enter during a configuration or installation procedure, or on a command line:

Example:

- Enter `exit` on the command line.

Screen Captures Used in This Document

Screen captures from the product GUI (graphical user interface), as used in this document, may sometimes contain a minor spelling, capitalization, or grammatical error. 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.

Square Brackets

Square brackets indicate that a particular parameter or value is optional within a logical argument, a command, or some programming syntax. That is, the parameter's or value's presence is not required to resolve the argument, command, or block of code. The user decides whether to include this optional information. Here is a sample:

```
smcp_server -host [/flags]
```

Angle Brackets

Angle brackets indicate a placeholder for a value that the user must specify. This might be a DN or port number specific to your enterprise. Here is a sample:

```
smcp_server -host <confighost>
```

Related Resources

Consult these additional resources as necessary:

- *Genesys Voice Platform 7.6 Deployment Guide,* which provides detailed installation and configuration instructions for GVP.
- *Genesys Voice Platform 7.6 Reference Manual,* which provides instructions for the administration, provisioning, and configuring of GVP and its components.
- *Genesys Voice Platform 7.6 Troubleshooting Guide,* which provides trap and basic troubleshooting information for the GVP.
- *Genesys Voice Platform 7.6 Voice Application Reporter SDK Developer's Guide*, which provides examples on how to develop VoiceXML applications that interface with the Voice Application Reporter (VAR) database and generate application reports.
- *Genesys 7.6 Proactive Contact Solution Guide*, which consolidates information about the Genesys Proactive Contact solution. The Genesys Proactive Contact solution integrates Outbound Contact with GVP, and provides the ability to proactively initiate and handle outbound campaign calls using GVP.

- *Voice Extensible Markup Language (VoiceXML) Version 2.1, W3C Candidate Recommendation 13 June 2005.* The World Wide Web Consortium (W3C) publishes a technical report as a *Candidate Recommendation* to indicate that the document is believed to be stable, and to encourage its implementation by the developer community.
- *Genesys Technical Publications Glossary,* which ships on the Genesys Documentation Library DVD and which provides a comprehensive list of the Genesys and 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 operating systems 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*

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`.

Consult these additional resources as necessary:

- *Genesys Hardware Sizing Guide,* which provides information about Genesys hardware sizing guidelines for the Genesys 7.x and Genesys 8.x releases.
- *Genesys Interoperability Guide,* which provides information on the compatibility of Genesys products with various Configuration Layer Environments; Interoperability of Reporting Templates and Solutions; and Gplus Adapters Interoperability.
- *Genesys Licensing Guide,* which introduces you to the concepts, terminology, and procedures relevant to the Genesys licensing system.
- *Genesys Database Sizing Estimator 7.6 Worksheets,* which provides a range of expected database sizes for various Genesys products.

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.

Making Comments on This Document

If you especially like or dislike anything about this document, please 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 information in this document only and to the way in which the information is presented. Speak to 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.

Document Change History

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

Release 7.6.4

- TXML media elements are no longer supported and have been removed from Appendix A.

Chapter

1 Overview

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

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 *Voice Extensible Markup Language (VoiceXML) Version 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.

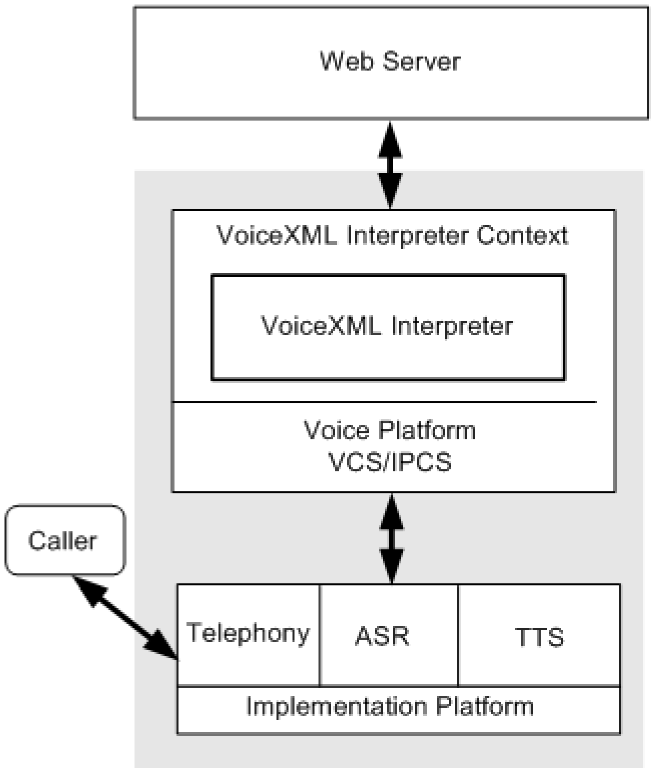


Figure 1: Platform for Voice Application

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), text-to-speech (TTS), and conferencing components. 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 CR 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 GVP supports them. Refer to the *Voice Extensible Markup Language (VoiceXML) Version 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
<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

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

Element	Purpose	New/Enhanced	GVP Support
<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.

Attributes

Table 2: <foreach> Attributes

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 Voice Communication Server/IP Communication Server (VCS/IPCS) 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.

Genesys supports using Nuance SWMS with RealSpeak for `<mark>`.

Note: RealSpeak 4.0 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. Support for the `<transfer>` element is optional with VoiceXML 2.1, and is not supported by GVP. VoiceXML 2.1 extends these elements so that the Interpreter can conditionally enable recording while simultaneously gathering input from the user.

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 VCS/IPCS.
2. The Interpreter posts them to the VoiceXML application.

The following sections provide more detail about each step, as well as the controls that are available to the VoiceXML application to regulate the Interpreter behavior during each step.

<submit> or <submit mode="sync"> or <subdialog>

After each recognition, the Interpreter downloads the captured utterance from the ASR server to the VCS/IPCS. 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.

<submit mode="async">

After each recognition, the Interpreter downloads the captured utterance from the ASR server to the VCS/IPCS. During `<submit mode="async">` 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 queues it to the Bandwidth Manager.

Note: In both scenarios, the ASR server to the VCS/IPCS download always happens synchronously. This might adversely affect the caller-perceived latency. If the VoiceXML application wants to improve the caller-perceived latency, the Interpreter makes available a custom property—`com.genesys.utterancefetchmode`—that can be used to control the Interpreter behavior during the download from ASR Server to the VCS/IPCS.

com.genesys.utterancefetchmode="sync"

If the property is unspecified, it defaults to `sync`. In both cases (unspecified and specified as `sync`), the Interpreter behavior is to download the utterances from the ASR server to the VCS/IPCS synchronously (that is, the same behavior as the two preceding scenarios).

com.genesys.utterancefetchmode="async"

During `<submit mode="async">` execution, the Interpreter spawns off an independent thread to download the utterances for all recognitions that have happened so far. The Interpreter thread also hands off all of the field data (including any `<record>` data) to this independent thread. The Interpreter then proceeds with the VoiceXML execution. When the independent thread has finished downloading the utterances, it packages all of the utterances and any field data (including the regular `<record>`, for example) into one multipart/form-data package, and queues it to the Bandwidth Manager. Because everything happens `async`, the shadow variables (`lastresult$.recording`, `lastresult$.recordingsize`, and `lastresult$.recordingduration`) are not filled in. The `<fieldname>$.recording` is still valid, and it can be used in the namelist to post the recording (see the example below).

Table 4 captures all combinations.

Table 4: Sync / Async Combinations

com.genesys. utterance fetchmode	<submit> Mode	Interpreter Behavior
sync (or unspecified)	sync (and <subdialog>)	The Interpreter downloads the utterance after each recognition. During `<submit>`/`<subdialog>`, the Interpreter packages all utterances and field data into one multipart/form-data, and posts it to the VoiceXML application.
sync (or unspecified)	async	The Interpreter downloads the utterance after each recognition. During <submit>, the Interpreter packages all utterances and field data into one multipart/form-data, and queues it to the Bandwidth Manager.
async	sync (and <subdialog>)	This is an invalid combination, and it is not supported. An `error.unsupported.utterancefetchmode` is thrown.
async	async	During `<submit>`, the Interpreter spawns off a thread to download all utterances so far. The Interpreter also hands off all field data to the new thread. The Interpreter proceeds with normal execution. The new thread, after downloading the utterances, packages everything (utterance plus field data) into one multipart/form-data and queues it to the Bandwidth Manager. The shadow variables (`recording`, `recordingsize`, and `recordingduration`) are not available to the VoiceXML application.

Example of VoiceXML Application with Async Utterance Fetch

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
      xmlns:telera="http://www.telera.com/vxml/2.0/ext/20020430"
      xmlns:conf="http://www.w3.org/2001/vxml-conformance" version="2.1">

<property name="recordutterance" value="true"/>
<property name="com.genesys.utterancefetchmode" value="async"/>

<form id="utterance_test">
    <field name="saywhat">
        <grammar version="1.0" root="city" type="application/srgs+xml">
            <rule scope="public" id="city">
                <one-of>
                    <item>chicago</item>
                </one-of>
            </rule>
        </grammar>
        <prompt count="1">
            Say 'Chicago'.
        </prompt>
        <prompt count="2">
            Say 'Chicago' again.
        </prompt>
        <prompt count="3">
            Say 'Chicago' one more time.
        </prompt>
    </field>
    <filled>
        <var name="the_recording" expr="saywhat$.recording"/>
        <telera:submit method="post"
            mode="async"
       enctype="multipart/form-data"
       next="http://10.10.10.245:9810/upload_log/capture1.asp"
       asyncposturl="http://10.10.10.245:9810/upload_log/capture1.asp"
       namelist="the_recording"/>
    </filled>
</form>
</vxml>
```

Specifying the Media Format of Utterance Recordings

In this release, GVP 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.

Interaction with Existing Controls

GVP supports the following controls for utterance capture:

- The VCS/IPCS controls how many simultaneous ports can do utterance recording.
- The Policy Manager controls how many total calls per day per application can do utterance recording.

  Note: This applies to GVP multi-tenancy only. For GVP single-tenancy, there is no upper limit.

- The `appid.xml` has the `$asrwavfilelog$` flag, which instructs the VCS/IPCS to turn on/off utterance capture (no VoiceXML application intervention is necessary).

All of these controls apply to utterance captures on MRCP ASR.

Table 5 shows how `recordutterance` property and `$asrwavfilelog$` interact.

Table 5: Interaction of recordutterance and $asrwavfilelog$

recordutterance	$asrwavfilelog$	Comments
true	X	The utterance recording occurs subject to port/Policy Manager controls. Recordings will be posted to the VoiceXML application, as described in the preceding sections.
false	X	The utterances will not be recorded.
unspecified	true	The utterance recording occurs, subject to port/Policy Manager controls. The recordings remain on the ASR server (GVP will not retrieve them).
	false/unspecified	The utterances will not be recorded.

If the application turns on the utterance recording (via `recordutterance`), but the recording does not occur because of the port/Policy Manager controls, the value of the recording variable is undefined (see Table 3 on page 19).

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.

Notes: For consultation transfer, `transferaudio` is not supported; however, `transferaudio` is supported for bridge transfer.

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, GVP 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.

While GVP supports VoiceXML consultative transfer requests over IP SIP and also TDM in carrier connected environments, it has not been validated with TDM connections, in behind-PBX environments, using the consultative transfer tag.

GVP supports consultative transfer for the following configurations:

- SIP REFER with Replaces
- Two B-Channel Transfer (TBCT)
- Explicit Call Transfer (ECT)
- Release Link Trunking (RLT)

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`.

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>
```

Support Notes

<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.

<grammar> 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 grammar support SRGS support is via a third-party MRCP ASR server. ABNF support is on the IBM WVS only.

DTMF grammar support 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 optional ABNF format support or optional semantic interpretation support. Native support will terminate processing of user input when the maximum allowed DTMF string is entered, when the interdigit timeout has expired, or when a termination character has been entered.

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 6: 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 7: Fetching Properties

Property	Supported	Default
fetchaudio	yes	n/a
fetchtimeout	yes	3s

Table 7: Fetching Properties (Continued)

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

Table 8: Miscellaneous Properties

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

Table 9: Prompt and Collect Properties

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

Table 10: Generic DTMF Recognizer Properties

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

VoiceXML 2.1

Table 11: <data> Fetching Properties

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

Table 12: Recording User Utterances While Attempting Recognition

Property	Supported	Default
recordutterance	yes	false
recordutterancetype **Note:** The MRCP protocol does not support this property. 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	

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:

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:

- Asynchronous Post of Recorded Files
- 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

Asynchronous Post of Recorded Files

The VoiceXML `<submit>` element has been enhanced to allow the asynchronous posting of recordings so that the application does not have to wait for the transmission of the recording to be completed before proceeding to the next element. This is enabled by the use of the `mode` and `asynchposturl` attributes.

Table 13 lists the extended attributes.

Extended Attributes

Table 13: Extended Attributes

mode	Specifies the mode for sending the recording. This attribute is checked if the namelist of the `<submit>` element contains record field variables. It can take one of the following values: • `sync`—The namelist is sent to the `next` URL. • `async`—The namelist is sent to both the `asyncposturl` and the `next` URL.
asyncposturl	The URL to which the namelist is sent in case of `async` mode. Data is sent to `asyncposturl` via HTTP POST using multi-part/form-encoding.

Here is an example:

```
<submit xmlns="http://www.telera.com/vxml/2.0/ext/20020430"
[next="URL" | expr="script to evaluate the URL"]
namelist="variablelist"
mode="async"
method="post"
asyncposturl="URL to process the recording"
/>
```

Note: The `encoding` and `method` attributes can be used to control how the data is sent to the next URL.

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.

For the VCS, this feature depends on the TDM trunk card vendor—for example, Dialogic—to supply the appropriate CPA detection. Enhanced CPA results are available when the Call Progress Detection (CPD) library is enabled. You can enable CPD through the Element Management Provisioning System (EMPS). Refer to the *Genesys Voice Platform 7.6 Reference Manual* for instructions.

For the IPCS, this feature depends on the VoIP Media Gateway having the appropriate signaling message, which it obtains by performing the CPA detections. With native RTP, 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. Host Media Processing (HMP) integration provides a rich set of CPA results through HMP technology. Operator/Network Intercept support is dependent on Early Media being provided by the far end. With Media Sessions Markup Language (MSML) integrations (that is, Convedia and MRP), CPA is limited to detection of Answer and Ring No Answer. Far End

Busy support is dependent on the far end providing the appropriate SIP messages. The CPA is done before the call is answered, and is dependent on the appropriate SIP protocol messages being provided for CPA.

Genesys Voice Platform supports the following CPA features:

- Normal Answer
- Answering Machine
- Fax/Modem
- Ring No Answer
- Far End Busy
- Operator/Network Intercept
- No Ringback
- Not In Service
- No Dialtone
- Unallocated Number
- Vacant Circuit
- SIT Unknown

The return result for CPA is through a dollar variable called `$_cparesult$`. Table 14 lists the various values:

Table 14: Values of CPA Features

CPA Feature	$cparesult$	VCS	VCS with CPD	IPCS Native	IPCS HMP	IPCS MSML	MRP
Normal Answer	CPA_NORMAL	✓	✓	✓	✓	✓	✓
Answering Machine	CPA_ANSWERMACHINE	✓	✓		✓	✓	✓
Fax/Modem	CPA_FAXMODEM	✓	✓		✓	✓	✓
Ring No Answer	CPA_NOANSWER	✓	✓	✓	✓	✓	✓
Far End Busy	CPA_BUSY	✓	✓	✓	✓	✓	✓
Operator/Network Intercept	CPA_OPERATOR INTERCEPT		✓		✓		
No Ringback	CPA_NO_RINGBACK		✓				
Not In Service	CPA_NOT_INSERVICE		✓				
No Dialtone	CPA_NODIALTONE		✓				
Unallocated Number	CPA_UNALLOCATED		✓				

Table 14: Values of CPA Features (Continued)

CPA Feature	$cparesult$	VCS	VCS with CPD	IPCS Native	IPCS HMP	IPCS MSML	MRP
Vacant Circuit	CPA_VACANT_CIRCUIT		✓				
SIT Unknown	CPA_SIT_UNKNOWN		✓				

To support CPA, GVP extended the `<transfer>` element under the Telera namespace with the attribute `AFTERCONNECTTIMEOUT` (refer to "<CREATE_LEG_AND_DIAL>" on page 68 for additional information on this attribute). This attribute defines the timeout, `afterconnecttimeout`, which starts after the outbound call is connected. The transfer will not return until this timeout expires.

When performing CPA with VoiceXML <transfer> on a bridge transfer, unless there is positive voice detection on the outbound call, the outbound call will be dropped and control returned to the application.

The following outcomes for the transfer are set by the platform:

- far_end_disconnect
- near_end_disconnect
- maxtime_disconnect
- busy
- noanswer
- answermachine
- faxmodem

The following outcomes are not set:

- network_disconnect
- network_busy
- unknown

You can also use `<CREATE_LEG_AND_DIAL>` to perform CPA. (See "<CREATE_LEG_AND_DIAL>" on page 68 for more information.)

The following is a VoiceXML example with the `AFTERCONNECTTIMEOUT` attribute in the `transfer` element under the Telera namespace.

```
<form id="transfer">
       <block>
           <audio src="voxfiles/transfer_title_EN_US.vox">I will now attempt to transfer you
to the operator</audio>
       </block>

       <transfer xmlns="http://www.telera.com/vxml/2.0/ext/20020430"
       name="operator" dest="4085553658" connecttimeout="20s"
       afterconnecttimeout="10s" bridge="true">
```

```
            <filled xmlns="http://www.w3.org/2001/vxml">

               <if cond="operator == 'busy'">
                  <audio src="voxfiles\operator_busy.wav">The operator is
                  busy</audio>
                  <audio src="voxfiles\return_mainmenu.wav">Returning to main
                  menu</audio>
                  <goto next="#menu_options" />
               </if>
            </filled>
         </transfer>
</form>
```

Append Attribute in <param> Element

A new attribute, `append`, is added to `<param>` with Genesys namespace. This attribute indicates whether the `header` should be appended to the existing header. Refer to "Classid telephonydata:put" on page 54 for more details.

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 GVP 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. For information about using the Genesys namespace, refer to the extension of "Classid telephonydata:put" on page 54.

Note: The namespace that GVP 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 43.

Table 15 lists the platform-specific variables.

Table 15: Platform-Specific Variables

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.
$ivr-root-dir$	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).

Table 15: Platform-Specific Variables (Continued)

$last-error$	The last error encountered by the VCS/IPCS. Possible values are: • `BAD_XMLPAGE` • `XMLPAGE_NOTFOUND` • `XMLPAGE_TIMEOUT` • `RESOURCE_NOTFOUND` • `RESOURCE_TIMEOUT` • `BAD_XMLTAGNAME` • `BAD_XMLTAGVALUE` • `OPSERVER_ERROR` The VCS/IPCS 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 VCS/IPCS 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.
$application-name$	The name of the voice application for this call. The Call Flow Assistant uses this to accept or reject the call. Also used for reporting purposes.
$customer-name$	The name of the customer to bill for this call.
$lata-name$	The local calling area where the call originated, for example, LSAN for a call from the Los Angeles area.
$ccerror-telnum$	The outbound telephone number that the VCS/IPCS code dials if the Call Flow Assistant and voice applications are not accessible (because of a problem with the data network).
$callerhup$	The VCS/IPCS 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 15: Platform-Specific Variables (Continued)

$voicefile-format $	The `ACCEPT_CALL` page sets this variable, or the first XML page sent back by the voice application sets it. Genesys recommends that you explicitly set this variable in the voice application's first XML page and that you not depend on the Call Flow Assistant to set it. The VCS/IPCS code needs this value to properly interpret the `.vox/.wav` files to be played in the voice application. This is available in TXML only.
	These values are available on the VCS:
	• Mu Law vox formats
	`VOX_MULAW_6KHZ`
	`VOX_MULAW_8KHZ`
	• Mu Law wav formats
	`WAV_MULAW_6KHZ`
	`WAV_MULAW_8KHZ`
	• A Law vox formats
	`VOX_ALAW_6KHZ`
	`VOX_ALAW_8KHZ`
	• A Law wav formats
	`WAV_ALAW_6KHZ`
	`WAV_ALAW_8KHZ`
	• ADPCM vox formats
	`VOX_ADPCM_6KHZ`
	`VOX_ADPCM_8KHZ`
	• ADPCM wav formats
	`WAV_ADPCM_6KHZ`
	`WAV_ADPCM_8KHZ`

$voicefile-format $
(continued)
	These values are available on the IPCS:
	• Mu Law vox formats
	`VOX_MULAW_8KHZ`
	• Mu Law wav formats
	`WAV_MULAW_8KHZ`
	• A Law vox formats
	`VOX_ALAW_8KHZ`
	• A Law wav formats
	`WAV_ALAW_8KHZ`
	• ADPCM vox formats
	`VOX_ADPCM_8KHZ`
	• ADPCM wav formats
	`WAV_ADPCM_8KHZ`

$_toneinput$	The VCS/IPCS code sets this variable to contain the tone that was pressed by the caller to exit out of a `TONEMAP`. A voice application can get the value of this variable by including it as part of the target URL for a subsequent form.

Table 15: Platform-Specific Variables (Continued)

$telephony-port$	The VCS/IPCS code sets this variable, which contains the channel number of the inbound call. A voice application can get the value of this variable by including it as part of the target URL for a subsequent form. **Note:** This variable will not be available for the application that is executed from the outbound leg
$_cparesult$	The VCS/IPCS 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` If an outbound ISDN PRI call is busy, the ISDN disconnect message indicates the cause; however, if the far end is busy and ISDN signaling does not propagate a correct cause value, Busy will not be detected. If an outbound call is busy on a T1/E1 trunk, the far end plays a busy tone. SIP should be able to differentiate the various busy signals used worldwide. • `CPA_NOANSWER` This indicates that an outbound call was initiated but the far end did not answer the call. Currently, in ISDN, the call continues to ring and control is returned to the application so it can decide whether to continue to wait. In T1/E1, the call is dropped and the application must issue a hang up to clean up the call. Here is another possible value if the line is ISDN: • `CPA_OPERATORINTERCEPT`
$ivr-error-url$	This variable points to a URL on the backup voice application. If the VCS/IPCS encounters any error (not just timeout) when making an `HTTP get` request from the voice application, it performs an `HTTP get` request to the URL at `$ivr-error-url$`. The query string for this URL contains the new `NextAction = CALLINPROGRESS`. Voice applications handling this `NextAction` can perform a graceful transfer by playing an appropriate message, and then transferring the call to a live agent or an alternative voice application. Existing voice applications that do not handle this request will get the same behavior as the current behavior—that is, the call gets dropped. If VCS/IPCS cannot access this URL or there is an error, it dials-out `$cc-error-telnum$`.
$sid$	The Call Flow Assistant sets this variable while issuing the `<ACCEPT_CALL>` element. It contains the value of the `ScriptID` as generated by the Queue Adapter or the IVR Server Client.
$scriptdata$	This variable contains any data specific to the ScriptID. Similarly, the Queue Adapter or the IVR Server Client provides the value to the Call Flow Assistant when the CFA informs the Queue Adapter or the IVR Server Client about an incoming call.

Table 15: Platform-Specific Variables (Continued)

$scripturl$	The voice application sets this variable before sending the `<SCRIPT_RESULT>` element. Its value will be the top HREF for the page that the CFA generates in response to a `RUN_SCRIPT_REQ` from the Queue Adapter or the IVR Server Client.
$playfilesize$	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.
$badxmlpageposturl$	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.
$dialed-number$	GVP 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>
```

Controlling TTS Prefetch

The TTS prefetch uses the VoiceXML property extension `com.genesys.ttsfetchhint`. By default, the TTS prefetch is set to `safe`.

- To turn off the TTS prefetch, use the value `safe` as shown in the following example:

```
<property name="com.genesys.ttsfetchhint" value="safe"></property>
```

- To turn on prefetch, use the value `prefetch`, as shown in the following example:

```
<?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">
<property name="termchar" value="#"></property>
<property name="com.genesys.ttsfetchhint" value="prefetch"></property>
   <menu id="sample_menu" dtmf="true">
      <prompt>
         <enumerate>
            For <value expr="_prompt"/>, Press <value expr="_dtmf"/>
         </enumerate>
      </prompt>
      <prompt bargein="false">
         Please press the number
      </prompt>
      <choice next="#choiceTwo_one">
         Billing Enquiry
      </choice>
   </menu>
   <form id="choiceTwo_one" >
      <field name="fld1">
         <prompt>
            Please enter your digit
         </prompt>
      <grammar src="http://.../main-menu1.grxml" />
      <filled>
```

```
        <prompt>you have said <value expr="fld1"/>
      </prompt>
      <clear/>
      <goto next="grtest.xml" />
      </filled>
   </field>
</form>
</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 ICM or 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.genesys.returntermchar`

  Use this property to indicate whether to return the termination character in addition to the other entered digits. For example, `<property name="com.genesys.returntermchar" value="true"/>` causes the termination character to be included in the result. The default value is "`false`".

  When setting this property to `true`, you must ensure the following:

  - You must set the VoiceXML `termchar` property to the termination character(s) that are to be accepted and returned.

    If multiple termination characters are specified, any of the characters specified could be used to terminate the input. For example, `<property name="termchar" value=*#">` allows either `*` or `#` to mark the end of the input.
  - The grammar to be matched must allow the termination characters as valid input. For example, if a four digit input is to be accepted, `com.genesys.returntermchar` is true, and `termchar` is set to `*#`, the grammar must also be capable of accepting `*` or `#` as its final digit. Failure to have the grammar allow the termination character(s) results in a `nomatch` being thrown. Refer to the following grammar as an example:

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

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

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

<rule id="pin" scope="public">
 <one-of>
```

```
   <item>
     <item repeat="4"><ruleref uri="#digit"/></item>
     <item repeat="0-1">*</item>
   </item>
   <item>
     <item repeat="4"><ruleref uri="#digit"/></item>
     <item repeat="0-1">#</item>
   </item>
</one-of>
</rule>
</grammar>
```

- `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`.

- `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 VCS:

  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)

  These values are available in VoiceXML on the IPCS:

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

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

A Law .wav formats:
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 16 on page 49.

Table 16: Attributes

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.)
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 VCS/IPCS 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`—to get data from Framework
- `CRData:put`—to send attached data to Framework
- `CRData:genericAction`—to perform an action on Framework
- `Telephonydata:put`—to send data via SIP INFO in case of IPCS
- `transactionalrecord:start`—to start transactional recording
- `transactionalrecord:stop`—to stop transactional recording
- `asr:freeResource`—to free 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

Note: This object class is supported on the IPCS only.

In the case of the IPCS, you can send user data using SIP INFO 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="<DATA>" />
<genesys:param name="header" append="<true/false>" value="<DATA>" />
<genesys:param name="body" value="<DATA>" />
</object>
</form>
</vxml>
```

Table 17 details the contents of the `telephonydata:put` class.

Table 17: Telephonydata:put Class Contents

Parameter	Values	Description
protocol	sip	Indicates telephony call control protocol. If not provided, then the platform default is `sip`.
msgtype	INFO	Indicates the SIP request/response to send. If not provided, then defaults to `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 a 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`.
body	Entire body of SIP message	Contents of the entire body portion of the SIP message. Maximum size is 1024 bytes. Binary data is not supported.

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

Receiving User Data

The IPCS 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 an XML `object` element to send customized SIP messages.

The `Genesys.protocol` object indicates the protocol that is being used for call control. For SIP, this parameter will always be set to `sip`. To get data from an incoming SIP message, the VoiceXML application uses the `Genesys.sip` object.

The `Genesys.sip` object will be an array of SIP messages. As each new SIP message is received, it appends to the array. The application keeps track of which messages it has processed. GVP appends each new message to the end of the array of messages. GVP does not maintain a history, nor does it reference new updates. Table 18 summarizes the objects:

Table 18: Objects

Field	Data Type	Values	Description
Genesys.protocol	string	SIP	Indicates call control protocol. Currently only SIP is supported.
Genesys.sip	array of sip objects		This is an array of sip objects. Each inbound SIP message is appended to the array.
sip.msgtype	string	INFO	Indicates the SIP message type. Currently only INFO messages will be added to the array.
sip.header	array of strings	SIP header with their values	Contents of the entire header portion of the SIP message. Each header has a separate entry in the array. The first index (zero) contains the request URI followed by individual headers, in the order in which they appear in the message.
sip.body	string	Entire body of SIP message	Contents of the entire body portion of the SIP message. Maximum size is 1024 bytes. Binary data is not supported.

The following example shows how the `Session` object represents a given SIP INFO message.

Note: Currently, only SIP INFO messages are appended to the `Session` objects.

```
INFO sip:10.10.10.113:5060 SIP/2.0
From: <sip:6261395@10.10.16.28>;tag=ds-29-59c189c8
To: <sip:301681@10.10.10.113>;tag=2289469511802061817
Contact: <sip:User_Name@10.10.16.28:5060;transport=udp>
Max-Forwards: 70
Call-ID: A9AC79A0-1DD1-11B2-A433-916D5613A99E@10.10.10.113
CSeq: 4 INFO
Content-Length: 30
Via: SIP/2.0/UDP 10.10.16.28:5060;branch=z9hG4bKc5b0352a-05f4-11da-
8029-8f471417ffe9
Content-Type: application/text
Supported:  timer

My account number is 123457890
```

The preceding SIP message is represented using application objects as follows:

```
session.genesys.protocol="sip"
session.genesys.sip[0].msgtype="INFO"
session.genesys.sip[0].header[0] = "INFO sip:10.10.10.113:5060 SIP/2.0"
session.genesys.sip[0].header[1] = "From:
<sip:6261395@10.10.16.28>;tag=ds-29-59c189c8"
session.genesys.sip[0].header[2] = "To:
<sip:301681@10.10.10.113>;tag=2289469511802061817"
session.genesys.sip[0].header[3] = "Contact:
<sip:User_Name@10.10.16.28:5060;transport=udp'
session.genesys.sip[0].header[4] = "Max-Forwards: 70"
session.genesys.sip[0].header[5] = "Call-ID: A9AC79A0-1DD1-11B2-A433-
916D5613A99E@10.10.10.113"
session.genesys.sip[0].header[6] = "CSeq: 4 INFO"
session.genesys.sip[0].header[7] = "Content-Length: 30"
session.genesys.sip[0].header[8] = "Via: SIP/2.0/UDP
10.10.16.28:5060;branch=z9hG4bKc5b0352a-05f4-11da-8029-8f471417ffe9 "
session.genesys.sip[0].header[9] = "Content-Type: application/text"
session.genesys.sip[0].header[10] = "Supported:  timer"

session.genesys.sip[0].body="
My account number is 123457890
```

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 `appid.xml`), 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. If the mode of `post` is specified as `async`, the posting is done using BWM, and the application is not guaranteed to get the recording during the call. If, for some reason, the post to BWM is not successful (for example, if the BWM is down), the recording is posted synchronously. If the mode is `sync`, the recording is posted directly to the application (without using BWM) 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="async" />
<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 19 could be thrown by GVP to the application depending on the scenario:

Table 19: 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 IPCS.	`error. unsupported.object.transrec`	Transactional recording is not supported in IP Communication Server.
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 (that is, VoiceXML input of `type="voice"`) 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'"/>
<submin 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 VoiceXML applications to free an ASR resource on demand. You can use the `asr:freeResource` `classid` in the VoiceXML `<object>` element to instruct the VCS/IPCS 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 MRCP Server. Check with your MRCP ASR vendor for details on how licenses are freed 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:freeResource` 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:freeResource`.

```
<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>` throws `error.semantic.ASRFreeResource.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 VCS/IPCS shall implicitly free the MRCP session when bridging a call if there are no active grammars. The MRCP session is freed from both legs of the bridged calls.

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:

```
GenesysLab_<ResellerName>_<CustomerName>_<ApplicationName>_<ConnectionID>_<SessionID>
```

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

Redirecting Number

When an incoming call that is to be transferred lands on GVP, and the Redirecting Number IE is available from the network, GVP captures the Redirecting Number, stores it as part of the call set-up data, and makes it available to the VoiceXML applications for further processing.

GVP stores the information in the following VoiceXML extension object:

```
session.connection.inboundcalldata.redirectingnumber
```

The VoiceXML application can retrieve the information as follows:

```
<prompt>
   <expr="session.connection.inboundcalldata.redirectingnum"/>
</prompt>
```

Note: The information is read-only and cannot be set by the VoiceXML application.

When the Redirecting Number IE is not available from the network, GVP returns an empty string to the VoiceXML application.

Presentation and Screening Indicators

GVP extracts the presentation and screening indicators from the Calling Party Number IE of an inbound leg, and transfers them to the VoiceXML application as call set-up data. The VoiceXML application then transmits the indicators to Calling Party Number IE of the ISDN set-up message of the outbound leg during call transfer.

GVP stores the information in the following VoiceXML extension objects:

```
session.connection.inboundcalldata.screeningIndicator
session.connection.inboundcalldata.presentationIndicator
```

The VoiceXML application can retrieve the information as follows:

```
<prompt>
   <expr="session.connection.inboundcalldata.screeningIndicator "/>
   <expr="session.connection.inboundcalldata.presentationIndicator "/>
</prompt>
```

Note: The information is read-only and cannot be set by the VoiceXML application.

Chapter

3

Reference for 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. Note that the elements are described according to their implementation in Genesys Voice Platform only.

The Call Control Elements described in this chapter are the only TXML elements that can be used in a GVP 7.6 VoiceXML application. Note that the TXML elements cannot be mixed with VoiceXML elements in a VoiceXML page, and must exist on a separate XMLPage as described in "TXML" on page 49. The VoiceXML page can transition to a XMLPage and a XMLPage can transition back to a VoiceXML page during the execution of an IVR application.

This chapter covers the following topics:

Call Control Elements

Table 20 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 20: Call Control Elements

Element	Attribute	Non-settable Attributes
<ALERT_LEG>	LEG_ID IVRURL	
<BRIDGE_CALL>	LEG_ID	
<CREATE_LEG_AND_DIAL>	TELNUM IVRURL BRIDGE ENDSESSIONONHUP URL_ONLEG2HUP CPATIMEOUT AFTERCONNECTTIMEOUT ANI CALLTRIGGEREVENT	
<END_SESSION>		
<HANGUP_AND_DESTROY_LEG>	REASON	
<LEG_WAIT>	TIMEOUT HREF	
<ON_LEGHUP>	ENDSESSION OTHER_LEG_URL	
<QUEUE_CALL>	AGENTGRP USR_PARAMS AGENT_URL	
<REXFER>	TELNUM IVRURL LEG_ID	
<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

Table 21: 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

Table 22: 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

Table 23: Attributes

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. Bridging three or more calls (a conference) requires specific hardware (a conference board) on the VCS.

Child/Parent Elements

Table 24: 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 45 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"?>
<XMLPageTYPE= "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"?>
< XMLPageTYPE= "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"
CALLTRIGGEREVENT="connected,alerting,callproceeding,100-199,1xx"
/>
```

Attributes

Table 25: 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.

Table 25: Attributes (Continued)

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.
ENDSESSIONON HUP	(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 VCS/IPCS 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.`
AFTERCONNECT TIMEOUT	(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.

Table 25: Attributes (Continued)

ANI	This is the Caller ID sent out for the outbound call. The T1 or E1 carrier must also support this feature.
CALLTRIGGER EVENT	This attribute specifies at which call state the voice application regains control when generating outbound calls. The call trigger can be a comma separated list of the following values: • `connected`—The voice application gains control after the outbound call has been answered. This is the default value. • `alerting`—The voice application gains control after the far end sends the VCS/IPCS alerting indication or the VCS/IPCS detects ring back tones. `Alerting` in the IPCS is the same as `180`. • `callproceeding`—The voice application gains control after the far end sends the VCS/IPCS a call proceeding message. `Callproceeding` in the IPCS is the same as `100`. • `100-199`—The voice application gains control after the IPCS receives the specific provisional message. Note that the 100 response and 180 response is the same as callproceeding. • `1xx`—The voice application gains control after the VCS/IPCS receives any 1xx message. The far ends sends the VCS/IPCS a call proceeding message.

CallTriggerEvent

Table 26 on page 71 indicates the value of the `CPA_RESULT` that will be available in the application under different scenarios. The voice application can access this `CPA_RESULT` using the `$_cparesult$` variable, for example, as query string variable in the `IVRURL`. The transient `CPA_RESULT` indicates the value when `CREATE_LEG_AND_DIAL` gives control back to the application after receiving the specified call trigger event. Based on the final disposition of the call, this value can change. The application can check the final cap result at the end of the call (the hangup event handler URL can carry the `$_cparesult$` variable).The final value will change only in case of `BUSY` when the `CALLTRIGGER` event is not "connected." If the voice application wants to access the `CPA_RESULT` prior to the end of the call, it can do so by having a `LEG_WAIT` with a timeout and HREF with `CPA_RESULT $` variable.

Table 26: CPA_RESULT with CallTriggerEvent

CallTriggerEvent	Final Call Result	Transient CPA_RESULT	Final CPA_RESULT
connected	answered	CPA_NORMAL	CPA_NORMAL
	busy	CPA_BUSY	CPA_BUSY
alerting callproceeding 100–199 1xx	answered	CPA_NORMAL	CPA_NORMAL
	busy	CPA_NORMAL	CPA_BUSY

Child/Parent Elements

Table 27: 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 45 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 75).

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"?>
<XMLPageTYPE= "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>
```

Route-Based Dialing

GVP supports route-based dialing on the IPCS. The following is an example of the TXML control:

`<CREATE_LEG_AND_DIAL TELNUM="ROUTE:N<number>" BRIDGE="YES"/>`

Where *N* is the route number. The key word `ROUTE` is case sensitive, and it must be in uppercase. The route number specified must have been configured with outbound ports.

<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

Table 28: Attribute

REASON	Reserved for future use.

Child/Parent Elements

Table 29: 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/>
```

The <END_SESSION> element has no attributes.

Child/Parent Elements

Table 30: 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"?>
<XMLPageTYPE= "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

Table 31: 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

Table 32: 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"?>
<XMLPageTYPE= "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
<XMLPageTYPE= "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:`

```
<XMLPageTYPE= "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

Table 33: 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 voice application executes this element, it sends a `QUEUE_CALL_REQ` request to the Interpreter Context. The Interpreter Context calls the Call Router to queue the call.

Syntax

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

Attributes

Table 34: 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 Queue Adapter or IVR Server Client forwards the values of the subparameters to the Call Router.
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

IVR Server Client Details

If you are using the IVR Server Client, then `USR_PARAMS` must consist of one parameter called `GenesysRouteDN`. Therefore, `USR_PARAMS` is mandatory.

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

Queue Adapter Details

If Queue Adapter is used for Cisco, then `USR_PARAMS` is limited to 10 variables with 40 characters each. Variables are named `CallVar1, CallVar2, ... CallVar10`. These variables can be populated using `ICMVarMappingMode` in `USR_PARAMS`.

1. `<QUEUE_CALL USR_PARAMS="ICMVarMappingMode:MAP_INORDER AcctNum:1234, Amount: 500" />`

   In this example, `CallVar1` will be populated with `1234` and `CallVar2` will be populated with `500`.

2. `<QUEUE_CALL USR_PARAMS="ICMVarMappingMode:MAP_SPECIFICVARS, CallVar1:1234, CallVar6:500, CallVar7:Valid />`

   In this example, `CallVariable1` will be set to `1234`, `CallVariable6` set to `500` and `CallVariable7` set to `Valid`. All other ICM variables will not be set.

Note: The variable name must be one of `CallVar1`, `CallVar2,...CallVar10`.

3. `<QUEUE_CALL USR_PARAMS= AcctNum:1234, Amount: 500 />`

   In this Example, since `ICMVarMapping` mode is not used, `CallVar3` will be populated with `1234` and `CallVar4` will be populated with `500`. `CallVar1` and `CallVar2` are reserved for internal use.

4.
   ```
   <QUEUE_CALL
   USR_PARAMS="ICMVARMappingMode:MAP_INORD,ECC.AcstNum:123,ECC.Amount:
   500" />
   ```

   In this example the, the `AcctNum` and `Amount` variables are passed as ECC variables to the ICM.

Encoding Data

If a '%', ':' or ',' is present as part of the data, that data needs to be encoded using the URL encoding scheme. The Genesys component Voice Platform Queue Adapter decodes the data and presents it to the ICM. Table 35 gives the details.

Table 35: Encoding Data

User Data	Encoded Data to Interpreter Context	Encoded Data to Queue Adapter	Data Passed to Call Router
CallVariable1: 123,456	CallVariable1: 123%2C456	CallVariable1: 123%2C456	CallVariable1: 123,456
CallVariable1: 123,456:78%9	CallVariable1: 123%2C456%3A78%259	CallVariable1: 123%2C456%3A78%259	CallVariable1: 123,456:78%9
CallVariable1: 123,456, CallVariable2: 1234	CallVariable1: 123%2C456, CallVariable2:1234	CallVariable1: 123%2C456, CallVariable2:1234	CallVariable1: 123,456, CallVariable2:1234

Child/Parent Elements

Table 36: 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 45, 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"?>
<XMLPageTYPE= "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>
```

<REXFER>

The `<REXFER>` element transfers a call from one agent to another. This involves unbridging the inbound and outbound legs, dropping the current outbound leg, creating a new outbound leg, dialing a new phone number on the new outbound leg, and, finally, bridging the inbound and new outbound legs.

The `<REXFER>` element provides the voice application with a convenient macro to perform all these tasks with one element. However, you can also execute this operation by issuing all the elements separately.

Upon executing a `<REXFER>` element, the voice application sends a `REXFER_REQ Next action` to the Interpreter Context. The `REXFER_REQ` is divided into several subtasks, as listed above. To perform the required subtasks, `REXFER_REQ` then issues interrupts or elements to the different legs.

Syntax

```
<REXFER
TELNUM="telephone number"
LEG_ID="name"
IVRURL="URL
/>
```

Attributes

Table 37: Attributes

TELNUM	The telephone number to be dialed on the new leg.
IVRURL	(Optional) The URL from which the document for the new outbound leg is executed. This element can be used to “whisper” a message to the new agent. If this attribute is missing, the inbound and new outbound legs are bridged as soon as the dialed call is answered on the outbound leg, otherwise, the XMLPage generated by `IVRURL` bridges the call after the “whispering” is complete.
LEG_ID	(Optional) The identity of the inbound leg to which the new outbound leg should be bridged. A value of `ALL` bridges the new leg with all other legs associated with the call/session.

Child/Parent Elements

Table 38: Child/Parent Elements

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

After the agent interrupts the outbound leg (for example, by pressing a specific key for which a global tonemap exists on the outbound leg), the outbound leg must issue the `<REXFER>` element.

Notes About the <REXFER> Element

1. `<REXFER>` must be followed by `LEG_WAIT` in the XMLPage. Otherwise, after executing the `<REXFER>` element, control passes to the `HREF` specified at the top of the XMLPage.
2. Using `<REXFER>` has this limitation: no message can be played on the inbound leg while transferring the agent leg. If control is given to the voice application on the inbound leg, it can result in a race condition where the Interpreter Context and the voice application both try to issue elements/interrupts to the inbound leg. The subsequent order of execution is unpredictable.
3. For the inbound leg to have full control while the agent leg is being destroyed and transferred, do not use the `<REXFER>` element.

In the following example, the `<REXFER>` element transfers the caller to an agent. The `TELLNUM` attribute specifies the number to dial. The `<IVURL>` element executes the `agent1.vxml` file. This document "whispers" to the agent that a call is coming. The `wait.xml` file allows the agent to wait before the call is transferred.

Examples

Document 1

```
<?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://customer_service/rexfer.xml"/>
      </field>
   </form>
</vxml>
```

Document 2 rexfer.xml

```
<?xml version="1.0"?>
<XMLPageTYPE= "IVR" CUSTID="rexfer" VERSION="2.5"
   SESSIONID="$sessionid$" HREF="http://Telera.net/
   cust_service/agent1.asp">

   <REXFER   TELNUM="<%= NumToDial%>" LEG_ID="ALL"
      IVURL="http://Telera.net/cust_service/agent1.vxml"/>

   <LEG_WAIT />
</XMLPage>
```

Document 3 Agent1.vxml

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

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

         <prompt>
            This is an incoming call from
            <var name="customer" expr="name"/>
         </prompt>

         <goto next="http://customer_service/wait.xml"/>
      </field>
   </form>
</vxml>
```

Document 4 Wait.xml

```
<?xml version="1.0"?>
<XMLPageTYPE= "IVR" CUSTID="rexfer" VERSION="2.5"
   SESSIONID="$sessionid$">
   <LEG_WAIT/>
</XMLPage>
```

<SCRIPT_RESULT>

The voice application executes this element in response to the Call Router's `RUN_SCRIPT_REQ` to the voice application. When the voice application executes this element, it sends a `SCRIPT_RESULT` request to the Interpreter Context. The Interpreter Context presents the data to the Call Router as a response to the previous request.

Syntax

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

Attribute

Table 39: 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 Queue Adapter or IVR Server Client 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 Queue Adapter or IVR Server Client.

USR_PARAMS

IVR Server Client Details

None.

Queue Adapter Details

If the Queue Adapter is used for Cisco, then `USR_PARAMS` is limited to 10 variables with 40 characters each. Variables are named `CallVar1`, `CallVar2`,...`CallVar10`. These variables can be populated using `ICMVarMappingMode` in `USR_PARAMS`.

1. `<SCRIPT_RESULT USR_PARAMS="ICMVarMappingMode:MAP_INORDER AcctNum:1234, Amount: 500" />`

   In this example, `CallVar1` will be populated with `1234` and `CallVar2` will be populated with `500`.

2. `<SCRIPT_RESULT USR_PARAMS="ICMVarMappingMode:MAP_SPECIFICVARS, CallVar1:1234, CallVar6:500, CallVar7:Valid />`

   In this example, `CallVariable1` will be set to `1234`, `CallVariable6` set to `500` and `CallVariable7` set to `Valid`. All other ICM variables will not be set.

Note: The variable name must be one of `CallVar1`, `CallVar2`, ...`CallVar10`.

3. `< SCRIPT_RESULT USR_PARAMS= AcctNum:1234, Amount: 500 />`

   In this example, since `ICMVarMapping` mode is not used, `CallVar3` will be populated with `1234` and `CallVar4` will be populated with `500`. `CallVar1` and `CallVar2` are reserved for internal use. Table 40 provides the details.

Table 40: Encoding Data

User Data	Encoded Data to Interpreter Context	Encoded Data to Queue Adapter	Data Passed to Call Router
CallVariable1: 123,456	CallVariable1: 123%2C456	CallVariable1: 123%2C456	CallVariable1: 123,456
CallVariable1: 123,456:78%9	CallVariable1: 123%2C456%3A78%259	CallVariable1: 123%2C456%3A78%259	CallVariable1: 123,456:78%9
CallVariable1: 123,456, CallVariable2: 1234	CallVariable1: 123%2C456, CallVariable2:1234	CallVariable1: 123%2C456, CallVariable2:1234	CallVariable1: 123,456, CallVariable2:1234

Child/Parent Elements

Table 41: 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>
```

<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

Table 42: 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

Table 43: 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 VCS/IPCS. The <SET> element tells the VCS/IPCS to associate a particular value with a variable name. When the voice application uses the specified variable in a document, the VCS/IPCS 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

Table 44: 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 45 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

Table 45: 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.

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://bridge_call.xml"/>
      </field>

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

Bridge_Call.xml

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

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>
```

Unbridge_Call.xml

```
<?xml version="1.0"?>
<XMLPageTYPE= "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 non interruptible.

Parameters

Table 46: 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).
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

Table 47: 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

Table 48: Parameters

MUSIC_DN	Directory number of the music source.
DURATION	Music duration in seconds. This parameter is optional.

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.

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

Appendix

A Transfer Scripts with DTMF Base

This appendix provides information on how to develop transfer scripts to support the TransferConnect feature.

This appendix includes the following sections:

Overview

This appendix explains how to write VoiceXML control scripts to support TransferConnect with DTMF base. It contains sample scripts that support ATT TransferConnect. You can use these sample scripts as templates to create your own scripts to support MCI TakeBackAndTransfer (TNT), Sprint Transfer release, or GVP Hook-flash Transfer.

Configuring TransferConnect in GVP

To use the TransferConnect feature, you need to provision both the VCS/IPCS and a voice application. Refer to the GVP Transfers section in the *Genesys Voice Platform 7.6 Reference Manual*, which describes transfer configurations.

Examples of Transfer Scripts

This section provides example transfer scripts. These scripts are found in the `<GVP Install Folder>\CN\Web\XferConnect` directory.

Converted XML Script for XferConnect

GVP provides default scripting for these transfer options:

- "ATTCourtesy"
- "ATTConference"
- "ATTConsultative"

ATTCourtesy

```
<?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:9810/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:9810/XferConnect/hangup.xml" />
  <elseif cond="returncode == '**5'" />
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  <elseif cond="returncode == '**7'" />
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  <elseif cond="returncode == '**8'" />
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  <elseif cond="returncode == '**1'" />
  <goto next="http://localhost:9810/XferConnect/hangup.xml" />
  </if>
</filled>
<noinput>
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  </noinput>
  </field>
  </form>
  </vxml>
```

ATTConference

```
<?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:9810/XferConnect/error.xml" />
  </catch>
<form id="att_conference">
<block>
<prompt bargein="false">
  <telera:value mode="dtmfplay" expr="'*8'" />
  <audio src="silence1000ms.wav" />
  </prompt>
  </block>
<field name="returncode">
<prompt bargein="false" timeout="5s">
  <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:9810/XferConnect/conf.xml" />
  <elseif cond="returncode == '**5'" />
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  <elseif cond="returncode == '**7'" />
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  <elseif cond="returncode == '**8'" />
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  <elseif cond="returncode == '**1'" />
  <goto next="http://localhost:9810/XferConnect/hangup.xml" />
  </if>
  </filled>
<noinput>
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  </noinput>
  </field>
  </form>
</vxml>
```

ATTConsultative

```
<?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" />
<catch>
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  </catch>
<form id="att_consultative">
<block>
<prompt bargein="false">
  <telera:value mode="dtmfplay" expr="'*8'" />
  <audio src="silence1000ms.wav" />
  </prompt>
  </block>
<field name="returncode">
<prompt bargein="false" timeout="3s">
```

```
  <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:9810/XferConnect/return.xml" />
  <elseif cond="returncode == '**5'" />
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  <elseif cond="returncode == '**7'" />
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  <elseif cond="returncode == '**8'" />
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  <elseif cond="returncode == '**1'" />
  <goto next="http://localhost:9810/XferConnect/hangup.xml" />
  </if>
  </filled>
<noinput>
  <goto next="http://localhost:9810/XferConnect/error.xml" />
  </noinput>
  </field>
  </form>
</Vxml>
```

Appendix

B UTF-8 Support for Attached Data

This appendix describes UTF-8 support for attached data.

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.

Application to IVR Server

The Call Flow Assistant accepts the following `HTTP POST` requests from the Conversation Controller.

- `QUEUE_CALL_REQ`
- `EXECUTE_IVR_SCRIPT_RESULT`
- `ATTACH_CRDATA`
- `GET_CRDATA`
- `EXECUTE_GENERIC_ACTION_REQ`

IVR Server to Application

To support UTF-8, the following occurs:

- When setting the `$script-data$`, the Call Flow Assistant specifies the `enctype` as `utf8` in the XML page.
- The Conversation Controller accepts the `$script-data$` UTF-8 data from the Call Flow Assistant. The Conversation Controller 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.

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 can enable hotword support by using the `Hotword Support` parameter on the EMPS > Servers > MRCP ASR server. Set the value of the `Hotword Support` parameter to `Vendor Specific Parameters` for MRCP servers (for example, Nuance SWMS) that support hotword by the vendor-specific parameter `Recognition-Mode` on MRCPv1. For all other servers, set the value of the `Hotword Support` parameter to `None`.

When configuring a SWMS server in GVP, you can use the `SampleASRServerSWMS` sample server, in which `Hotword Support` is already set to `Vendor Specific Parameters` by default.

Using Nuance SWMS, 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 `Hotword Support` 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 documentation for details on the parameters that you can optionally set on SWMS to fine-tune hotword recognitions.

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

Bridged Transfers

On IPCS/VCS, if the `com.telera.speechenabled` parameter is set to `true`, and if the ASR server supports hotwords, the termination of the called party leg during a bridged transfer by matching the caller's voice or DTMF input to a grammar is supported.

On VCS only, if `com.telera.speechenabled` is set to `false`, matched DTMF input can terminate the called party leg of the call.

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).

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>
```

Appendix

E SIP Headers

This appendix describes how to propagate SIP Header values to the VoiceXML application on IP Communication Server (IPCS).

This appendix contains the following section:

Propagation of Headers

IPCS 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, IPCS passes the value to the VoiceXML application.

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

Call-ID

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

Accessing Header Values

IPCS 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 IPCS 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.

For information about configuring the IPCS for SIP headers, see the *Genesys Voice Platform 7.6 Deployment Guide*.

Appendix

F Application Developer Note

This appendix contains information that is useful to application developers.

This appendix has one section:

Transferring Calls Using GVP

When transferring Voice over IP (VoIP) calls using IPCS, the destination number must be entered in the SIP URI format:

`sip:<application-name>@<destination-host-ip-address>:<port-number>`

For example:

```
sip:bankingapplication@10.10.10.100:9810
```

where

`bankingapplication` is the `<application-name>`

`10.10.10.10` is the `<destination-host-ip-address>`

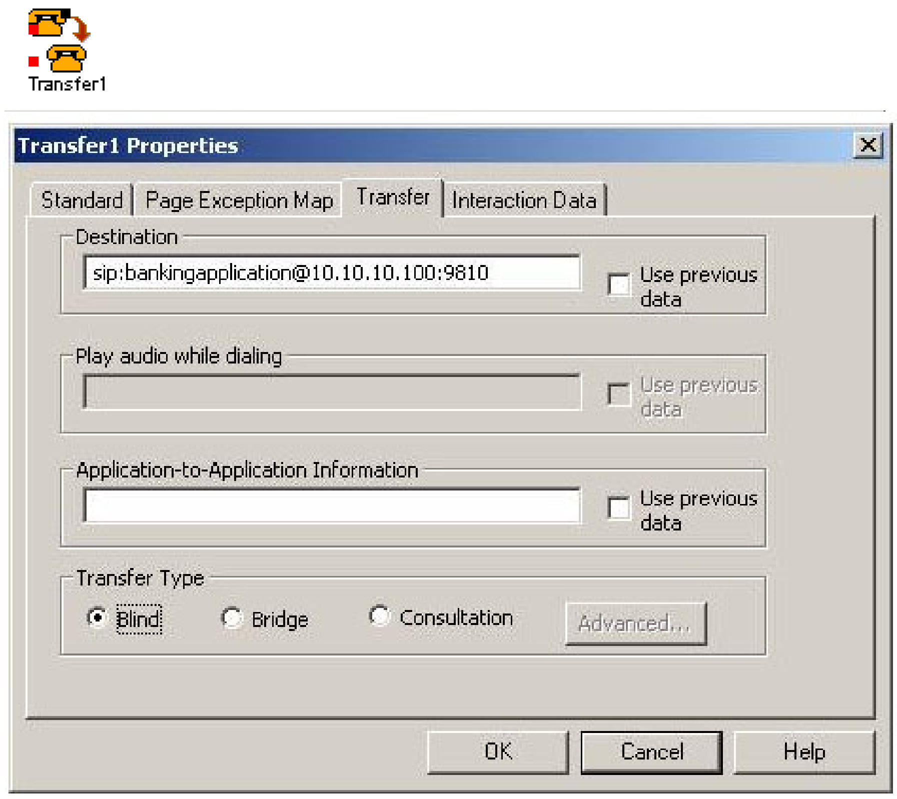
`9810` is the `<port-number>`

The port number is required only if the SIP Server on the destination host is listening on a port other then 5060. If the voice application in this example was listening on port 5060, the SIP URI would be `sip:bankingapplication@10.10.10.100`. The destination host can be the local GVP server or a remote server. The destination host also can be any SIP Server—for example, a softphone or a Media Gateway.

When you use Genesys Studio to create your voice applications, there are two blocks that you can use to enable call transfers in a voice application: CONNECT block and TRANSFER block.

Using TRANSFER Block

When using the TRANSFER block, enter the destination number in SIP URI format in the `Destination` field under the `Transfer` tab (see Figure 2).



Destination: sip:bankingapplication@10.10.10.100:9810 [ ] Use previous data

Play audio while dialing: [ ] Use previous data

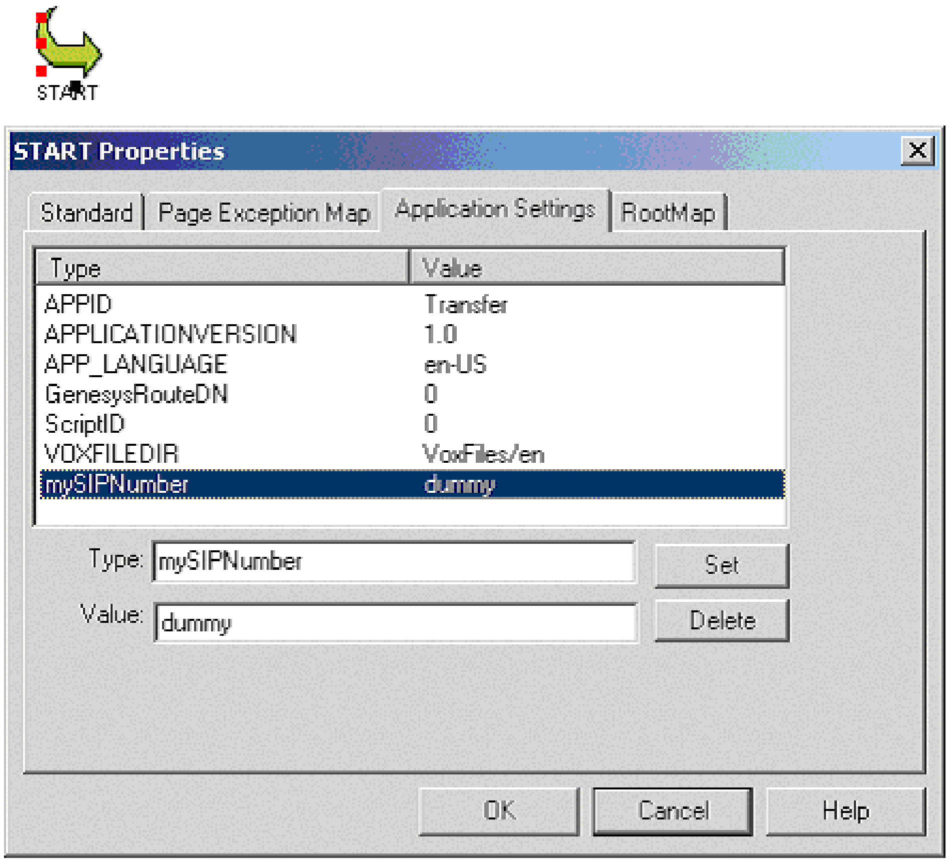
Application-to-Application Information: [ ] Use previous data

Transfer Type: [✓ Blind] Bridge Consultation

Figure 2: TRANSFER Block

Using CONNECT Block

When using the CONNECT block, in the START block of the voice application, a new variable under the Application Settings tab must be defined. The new variable should have a name such as `MySIPNumber` with either an explicit SIP address or as a placeholder value such as `dummy` (see Figure 3).

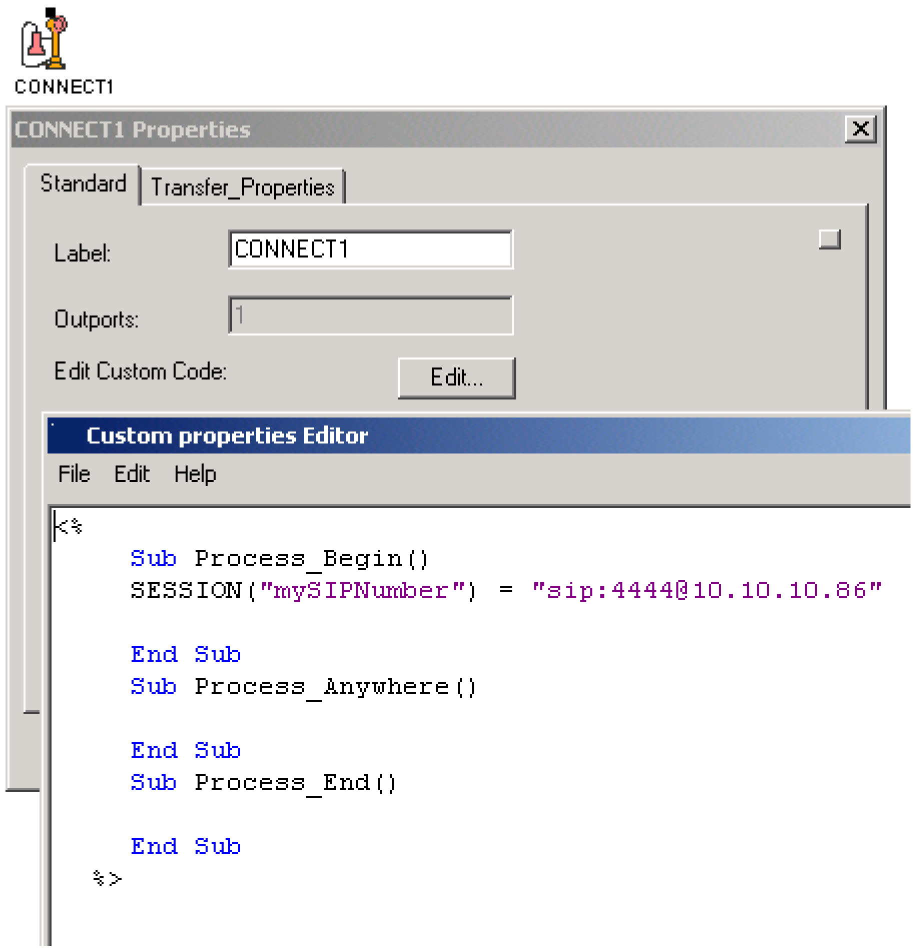


Type	Value
APPID	Transfer
APPLICATIONVERSION	1.0
APP_LANGUAGE	en-US
GenesysRouteDN	0
ScriptID	0
VOXFILEDIR	VoxFiles/en
mySIPNumber	dummy

Type: mySIPNumber
Value: dummy

Figure 3: START Block

If the variable is a place holder, in the CONNECT block under the Standard tab, the Custom Code must be modified to include a new value for the `mySIPNumber` variable defined in the START block (see Figure 4 on page 122). The value of the `mySIPNumber` must be the destination number in SIP URI format.



Label: CONNECT1

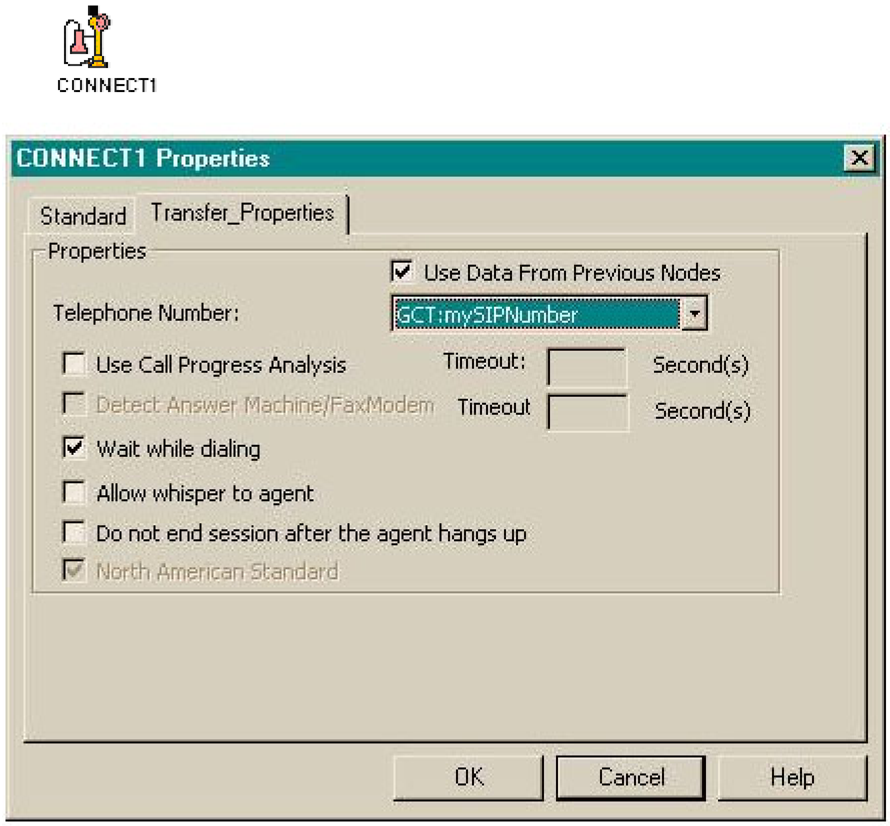
Outports: 1

Edit Custom Code:

```
<%
     Sub Process_Begin()
     SESSION("mySIPNumber") = "sip:4444@10.10.10.86"

     End Sub
     Sub Process_Anywhere()

     End Sub
     Sub Process_End()

     End Sub
  %>
```

Figure 4: CONNECT Block Custom Code

In either the case of an explicit SIP address or a placeholder replaced by custom coding, in the CONNECT block under the Transfer_Properties tab, select the `Use Data From Previous Nodes` check box and select the variable defined in START block (in this example `mySIPNumber`) in the pull-down list, see Figure 5 on page 123.



CONNECT1 Properties

Properties

- [✓] Use Data From Previous Nodes
- Telephone Number: GCT:mySIPNumber
- [ ] Use Call Progress Analysis — Timeout: Second(s)
- [ ] Detect Answer Machine/FaxModem — Timeout Second(s)
- [✓] Wait while dialing
- [ ] Allow whisper to agent
- [ ] Do not end session after the agent hangs up
- [✓] North American Standard

Figure 5: Connect Block Transfer Properties

Appendix

G Best Practices

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

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 Nuance OSR.
- 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.
- Use asynchronous mode for posting recordings.
- 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.
- Use of grammars inside transfers.
- 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 `WARNING` during normal operations of a live deployment.

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.

In case of whole-call recording and using Bandwidth Manager (BWM) for asynchronous posting, make sure that failure in posting of the recorded audio files is monitored. Failure in the receiving web server could cause the queuing of file to reach the maximum directory size (especially Windows) which will either degrade system performance or cause undefined behavior in the application.

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

GVP does not offer a performance improvement when the audio files are kept locally on the GVP platform. GVP 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.

Use Asynchronous Mode for Posting Audio Recordings

Posting of recorded files to the web application server can have an impact on the network bandwidth especially for applications that need to post large recording files such as the full duplex recording of long calls. GVP provides an option for asynchronous posting for recordings done by the `<record>` tag, VoiceXML 2.1 utterance capture feature, and for the full-duplex transactional recording.

The Bandwidth Manager (BWM) component of GVP manages the posting of asynchronous recordings from the Voice Communication Server (VCS) or IP Communication Server (IPCS) to the web server. The BWM queues the recordings and posts them to the web server at a fixed transfer rate. This is so that one application does not take too much of the network resources impacting other applications running on the same network. The recordings will remain queued with BWM until they are successfully posted to the web server.

Use the asynchronous mode of posting by default. For GVP deployments that do not contain the BWM component, the recording will get posted synchronously during the call.

In the event of failures, BWM will continuously retry to post the recordings. If you are using the asynchronous mode of posting and do not see the recording files on the web application server side after a reasonable period, review the BWM logs for transmission errors.

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 may result in a larger number of TTS licenses being used than expected and 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 and eight TTS Engine licenses will be used for converting text to audio for playback.

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>
```

Licenses for TTS engine can be further tuned by setting the GVP property `com.genesys.ttsfetchhint` to `safe` or `prefetch`. In the case of `prefetch`, all SSML text inside a prompt and alternative text inside audio are sent to TTS engine immediately and multiple TTS license can get used as seen in the preceding example. As in case of VoiceXML `fetchhint` property, setting the `ttsfetchhint` to `safe`, would result in TTS being generated only when the prompt will be actually played back to the caller.

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.

Grammars Inside Transfers

All editions of GVP may not support grammar recognition on bridged conversations. Use this feature carefully being cognizant of any limitations for your specific configuration.

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 like 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.

Index