VoiceGenie 7.2

Application Migration Guide (7.1 to 7.2)

About Genesys

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

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

Notice

Your Responsibility for Your System's Security

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

Trademarks

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	+81-3-5649-6871	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 7 Licensing Guide.

Released by

Genesys Telecommunications Laboratories, Inc. www.genesyslab.com

Document Version: 10-2007

Table of Contents

Chapter

1 Introduction

Overview

The VoiceGenie Media Platform includes a VoiceXML Interpreter component. This component is responsible for requesting Voice Extensible Markup Language (VoiceXML) pages from a web application server (via a caching proxy), compiling these pages into an internal representation, and then executing them in order to manage a dialog with a user. As part of this dialog management process, the VoiceXML Interpreter also requests resources (such as speech recognition and speech synthesis sessions) from other platform components.

The “legacy” VoiceGenie Interpreter is a mature software component that handles millions of calls every day in deployments around the world. However, this legacy interpreter has a number of limitations:

- Although it is highly tuned and delivers industry-leading performance, the legacy interpreter does not scale well with today’s multicore/multiprocessor systems.
- The architecture of the legacy interpreter is not well suited to forthcoming standards, such as VoiceXML 3.

Therefore, VoiceGenie has taken the opportunity to re-architect the VoiceXML Interpreter, producing a “next-generation” VoiceXML Interpreter software component, the NextGen Interpreter. The NextGen Interpreter is well positioned for the evolution of VoiceXML 3, and it leverages multicore/multiprocessor environments much more effectively than the legacy interpreter. The NextGen Interpreter also has a number of other improvements—for example, in the area of logging.

The VoiceGenie Media Platform can run one or both interpreters. In the latter case, application provisioning controls the selection of an interpreter for a particular application.

This document outlines the high-level application behavior and changes between the 7.0.x and 7.1.x VoiceGenie legacy interpreters and the 7.2 NextGen Interpreter/Media Platform. The purpose of this document is to help application developers migrate their VoiceXML applications from the legacy interpreter to the NextGen Interpreter.

Scope and Audience

This document describes the syntactic and semantic differences that an application developer will observe between the legacy interpreter and the NextGen Interpreter. It does not address configuration and management differences that may exist as a result of using a different VoiceXML interpreter.

This document is intended for anyone who writes VoiceXML applications for the VoiceGenie platform, including customers, partners, and professional services teams (both Genesys and third-party).

Chapter

2 General Changes

Strict XML Parser Implementation

In the legacy interpreter, the parser is very relaxed in terms of the documents that it accepts and executes.

Examples:

- The legacy interpreter will execute certain classes of malformed Extensible Markup Language (XML) documents. For example, if the closing `</vxml>` element is missing, the document will still execute.
- The legacy interpreter will accept special XML characters, such as `&` and `<`, even if they are not XML-escaped. In the NextGen Interpreter, any unescaped XML character causes a parsing error.
- The legacy parser will accept invalid attributes (that is, attributes that are not part of the VoiceGenie implementation of VoiceXML 2.0 or 2.1, or the VoiceGenie extension set)—the invalid attributes are simply ignored. In the NextGen Interpreter, any invalid attributes cause a parsing error.
- The legacy parser will accept an invalid element (that is, an element that is not part of the VoiceGenie implementation of VoiceXML 2.0 or 2.1, or the VoiceGenie extension set)—the invalid element is simply ignored, as are any child elements of it (whether valid or not). In the NextGen Interpreter, any invalid elements cause a parsing error.
- The legacy interpreter will, in some cases, accept attribute values that are not enclosed in quotation marks. In the NextGen Interpreter, any unenclosed attribute values cause a parsing error.
- The legacy interpreter will accept multiple grammars on the same page with the same id. In the NextGen Interpreter, this will cause a parse error.

In the NextGen Interpreter, parsing is done in strict accordance with the VoiceXML schema; any element or attribute that violates the schema will cause a parsing error.

Note: To avoid this issue, ensure that your VoiceXML documents are valid XML documents.

Time Designations

If an integer appears where a Time Designation is required (that is, if it does not have the `s` or `ms` suffix, as required by VoiceXML 2.0/2.1), the legacy interpreter logs a warning and treats the value as milliseconds. If the NextGen Interpreter encounters such a value, it throws an `error.semantic` event.

Note: To avoid this issue, ensure that all time measurements include the appropriate unit designation.

Access to Application Temporary Data

The `savetmpfiles` property was developed in the legacy interpreter to control whether the media platform permanently saves temporary files to the file system for debugging purposes. In the legacy interpreter, the `savetmpfiles` property takes a hexadecimal value, where each bit represents an application component that the application developer wants to save to the disk—for example, audio files or grammars. In practice, most applications use only `0x0` or `0xffffff` as the value for `savetmpfiles`, to completely enable or disable the `savetmpfiles` feature for all types of files.

In the NextGen Interpreter, the `com.voicegenie.savetmpfiles` feature uses a list of keywords to represent the different application components to be saved. Specifying the keyword `all` causes the media platform to save everything, whereas specifying the keyword `none` causes it to save nothing. The values `0x0` and `0xffffff` (for any number of `f`s) are equivalent to the keywords `none` and `all`, respectively. For details about the available keywords, see NGI Developer Workshop.

Changes to Shadow Variables

application.lastresult$[i].rawresults

In the legacy interpreter, the `application.lastresult$[i].rawresults` variable is used to expose detailed information about the recognition results. It uses a cumbersome text format to convey the extra information, such as per-slot confidence and ambiguous results.

In the NextGen Interpreter, the `rawresults` variable is no longer available. Instead, several new variables have been added that indirectly correspond to `rawresults:`

- `application.lastresult$.xmlresult` exposes the NLSML document as a DOM object, in the same way that a `<data>` object exposes an arbitrary XML document.
- `application.lastresult$[i].ambig_interpretation` exposes ambiguous results returned from the ASR engine. It is an ECMAScript array, where each element represents an `<instance>` element within the `<interpretation>` element of the NLSML result. If there is only one `<instance>` element within the `<interpretation>` element, `application.lastresult$[i].ambig_interpretation` is set to ECMAScript `undefined.`
- Each descendent node in `application.lastresult$[0].interpretation` may have a corresponding `$.confidence` shadow variable, if the corresponding NLSML document node has a per-slot confidence.

If your application relies on the values in the `rawresults` shadow variable, it must be updated so that it uses ECMAScript to walk through and parse the DOM object returned in the new shadow variable representation.

application.lastresult$.activegrammar and application.lastresult$.triggeredgrammar

In the NextGen Interpreter, these variables are now set for DTMF grammars, as well as for ASR grammars.

Although the Developer Workshop has documented the variable application.lastresult$.triggeredgrammar.linenumber for a `<choice>` within a `<menu>` as the line number of the menu itself, the legacy interpreter actually used the line number of the choice. This has been corrected in the NextGen Interpreter.

application.lastresult$.triggeredgrammar.pageuri

In the legacy interpreter, the URI being exposed to the application in either the `application.lastresult$.triggeredgrammar.pageuri` or the `application.lastresult$.activegrammars[i].pageuri` shadow variable has its URI parameters removed. The NextGen Interpreter preserves the URI parameters and exposes the entire URI.

In the NextGen Interpreter, `application.lastresult$.triggeredgrammar.pageuri` is undefined when a hotkey (universal) grammar is matched.

Removal of saveutterance-Related Shadow Variables

Because VoiceXML 2.1 has standardized the `recordutterance` feature, the NextGen Interpreter no longer supports the VoiceGenie extension property `com.voicegenie.saveutterance`. As a result, the following `application.lastresult$` shadow variables are no longer available:

- `application.lastresult$.utteranceaudio`
- `application.lastresult$.duration`
- `application.lastresult$.size`

These should be replaced with the `recording`, `recordingduration`, and `recordingsize` shadow variables, respectively. For further details, see `http://www.w3.org/TR/voicexml21/#sec-reco_reco`.

Other Removed Variables

The following shadow variables have been removed in the NextGen Interpreter:

- `application.lastresult$.info`
- Input item–level shadow variable `<name>$.info`
- Input item–level shadow variable `<name>$.bargeinphrase`
- Input item–level shadow variable `<name>$.bargeinscore`

In the legacy interpreter, the bargein-related shadow variables were supported only with the Watson ASR engine.

Access to <record> Metadata

In the legacy interpreter, when a variable representing a recording (either from a `<record>` input item or from an utterance recording) is logged via the `<log>` tag, the recording's file path is logged. Due to the changed internal structure of the NextGen Interpreter, this information is no longer available from the application environment. The recording file location *is* logged to the metrics file.

Full Call Recording

In the NextGen Interpreter, the format of the Full Call Recording (FCR) instructions has changed. Semicolons are no longer used as command delimiters and are simply ignored (or treated as part of the command). Only line breaks are treated as command delimiters.

As a result, it is no longer possible to have multiple commands on a single line, delimited by semicolons.

Example:

```
<log vg:dest="calllog">directory
/usr/local/phoneweb/callrec; enable callrec recsrc=mixed
type=audio/x-wav; keep-files true;</log>
```

This must be changed to:

```
<log vg:dest="calllog">
   directory /usr/local/phoneweb/callrec
   enable callrec recsrc=mixed type=audio/x-wav
   keep-files true
</log>
```

Alternate Page Handling

In the legacy interpreter, when the initial application page cannot be loaded, the alternate page (as specified in the application provisioning) is fetched and parsed, in order to support failover to an alternate server or application. However, the page does not begin execution as if it were a new page. Instead, after the page has been loaded, the error handler for `error.badfetch` is selected and executed.

Throwing an error immediately is not a good choice for the behavior in this case, because it is not clear:

- Which VoiceXML scopes have been initialized before the error is thrown.
- Which scope the error is thrown in.
- What happens when an error is encountered during scope initialization.

In the NextGen Interpreter, when the initial page cannot be loaded, the alternate page is fetched and parsed, and the page is executed from the beginning.

Pre-Fetching

In the current version of the NextGen Interpreter, pre-fetching functionality is not available. This capability may be made available in a future release. If pre-fetching is required, use the legacy interpreter for your application.

Prompt Queue Fetching

In the legacy interpreter, the `<audio>` files in the entire prompt queue are fetched from the HTTP server (and the alternate audio is enclosed within the `<audio>` elements) before the first audio file is played. In the NextGen Interpreter, the architecture has been changed so that the `<audio>` files are fetched from the HTTP server as each of them is played. This change improves the time-to-first-audio, because the NextGen Interpreter no longer needs to wait for the audio files from the entire prompt queue to be fetched before the first audio file can be played to the caller.

The legacy interpreter supports the value `stream` for the `fetchhint` property/attribute. As a result of the preceding architectural change in the NextGen Interpreter, this value for `fetchhint` is no longer supported.

TDD

In the current release of the NextGen Interpreter, support for TTD grammars is not available. This capability will be made available in a future release. If TDD is required, use the legacy interpreter for your application.

Page Fetch Changes

Initial Page

In the NextGen Interpreter, any errors in the initial page (that is, anything that would generate an `error.badfetch`) cause an incall rejection. By contrast, the legacy interpreter throws an `error.badfetch` to the same page or possibly the default page. In other words, the legacy interpreter allows a page that should have thrown an `error.badfetch` to begin execution, and throws the error only if the offending code is encountered.

Transitions with fetchaudio

When fetching pages, scripts, or data with `fetchaudio` specified, the NextGen Interpreter begins fetching the target page while any queued prompts are playing. The legacy interpreter waits until these prompts have been played before beginning the fetch.

Submitting Objects

If an ECMAScript object is used in a namelist to fetch something from a web server (for example, `<submit>` or `<data>`), the legacy interpreter is capable of resolving the subproperties of the object and adding the individual properties as parameters to the HTTP request.

Example:

```
<block>
  <script>
    var a = new Object();
    a.foo1 = “hey”;
    a.foo2 = “you”;
  </script>
  <submit next=”next.html” namelist=”a” method=”get”/>
</block>
```

For this example, the legacy interpreter creates an HTTP request with the following Request-URI (because the method is `get`, the parameters are added to the URI):

```
http://<server>/next.html?a%2Efoo1=hey&a%2Efoo2=you
```

To achieve similar functionality in the NextGen Interpreter, the application would need to use the `toSource()` function of ECMAScript objects. The preceding example may be translated to:

```
<block>
  <script>
    var a = new Object();
    a.foo1 = “hey”;
    a.foo2 = “you”;
    var toSend = a.toSource();
  </script>
  <submit next=”next.html” namelist=”toSend”
method=”get”/>
</block>
```

From this, the NextGen Interpreter creates an HTTP request with the following Request-URI:

```
http://10.0.0.12/next.html?toSubmit=%28%7Bfoo1%3A%22hey%22%2C%20foo2%3A%2
   2you%22%7D%29
```

After unescaping the `toSubmit` URI parameter, this becomes:

```
({foo1:"hey", foo2:"you"})
```

Note: Application developers may need to change their HTTP server code to accommodate these differences.

Submitting Arrays

If an array is used in a namelist to fetch something from a web server (for example, `<submit>` or `<data>`), the legacy interpreter is capable of resolving each element of the Array and adding the elements as individual parameters to the HTTP request.

Example:

```
<block>
  <script>
    var foo = [ 1, 2, 3 ];
  </script>
  <submit next=”next.html” namelist=”foo” method=”get”/>
</block>
```

For this example, the legacy interpreter creates an HTTP request with the following Request-URI (because the method is `get`, the parameters are added to the URI):

```
http://<server>/next.html?foo1=1&foo2=2&foo3=3
```

To achieve similar functionality in the NextGen Interpreter, the application would need to use the `toSource()` function (which serializes the Array into JavaScript Object Notation [JSON]). The preceding example may be translated to:

```
<block>
  <script>
    var foo = [ 1, 2, 3 ];
    var toSend = foo.toSource();
  </script>
  <submit next=”next.html” namelist=”toSend”
method=”get”/>
</block>
```

From this, the NextGen Interpreter creates an HTTP request with the following Request-URI:

```
http://10.0.0.12/next.html?toSubmit=[1,%202,%203]
```

Note: Application developers may need to change their HTTP server code to accommodate these differences.

Chapter

3 VoiceXML Behavior Changes

VoiceXML 1.0

The legacy interpreter implements the following VoiceXML 1.0 features, which are no longer supported in the NextGen Interpreter:

- Documents where the `version` attribute of the `<vxml>` element is set to `1.0`. These will be rejected and cause a parsing error.
- `session.telephone.*` session variables. The application should now use the `session.connection.*` session variables instead.
- `telephone.*` events (controlled by the platform configuration).
- Use of the `<dtmf>` element to define a DTMF grammar. The application should now use the `<grammar>` element with the `mode` attribute.
- Playback of a recorded input from a `<value>` element. By specifying `<value>` with an expression that references an input item collected with `<record>`, the legacy platform could play back the recorded audio. The application should now use the `<audio>` element with the `expr` attribute instead. For further details, see `http://www.w3.org/TR/voicexml20/#dml4.1.3.`
- The `<sentence>` and `<paragraph>` speech markup elements. These have been replaced by the `<s>` and `<p>` elements, as specified by a more recent release of the Speech Synthesis Markup Language (SSML) specification.
- SSML `<audio>` and `<value>` elements as children of the `<choice>` element. These child elements are no longer supported within `<choice>.`

VoiceXML 2.0

Element Changes

<catch>

The legacy interpreter use the anonymous variable `_disconnect_reason` to provide the disconnect reason. The NextGen Interpreter offers this information `_message` variable.

<choice>

The legacy interpreter supports the `fetchaudio` and `fetchaudiominimum` attributes of the `<choice>` element. In the NextGen Interpreter, these attributes are no longer supported. If the application needs to specify different `fetchaudio` properties on a per-choice basis, use `<form>` and `<field>` elements instead.

The NextGen Interpreter does not support SSML content inside `<choice>`.

<grammar>

When a built-in grammar such as `<grammar src="builtin:grammar/builtin_name" />` is matched, the legacy interpreter exposes the matched rulename in the `application.lastresult$.triggeredgrammar.rulename` variable, by setting it to the entire grammar URI `builtin:grammar/builtin_name.` In the NextGen Interpreter, the information is exposed via the same variable, but it is now exposed simply as the built-in grammar name—that is, as simply `builtin_name.`

As part of the strict parser changes in NGI, multiple id's on the same page with the same value are not permitted. This is in accordance with the `xml:id` schema restriction. In other words, for any given VoiceXML page, no two <form>s or `<grammar>`s can have the same id.

<link>

The behavior of `<link>` has been corrected so that it more closely conforms to the VoiceXML specification. Specifically, the NextGen Interpreter resolves the URI specified in a link according to section 2.5 of the specification, which states, “any URIs in an attribute of the link element are resolved dynamically, i.e. according to the base URI in effect when the link's grammar is matched.” By contrast, the legacy interpreter resolves the URI according to the base URI of the page that contains the link.

<log>

In NGI all log data is written in UTF-8 format.

<record>

Default type Value

If there is no `type` attribute in the `<record>` element to specify the desired recording format, and if the platform is set up to use ulaw by default, the legacy interpreter exposes the field item shadow variable `name$.filetype` as `audio/x-vox.` In the NextGen Interpreter, this value is exposed as `audio/basic.`

Invalid type Value

If the `<record>` element contains an invalid `type` attribute, the legacy interpreter uses `audio/vox` as the default, and the recording will be made. The NextGen Interpreter logs a warning message indicating that the MIME type is bad, and the recording will fail.

/timeout

The legacy interpreter supports the `beginsilence` attribute for the `<record>` element. In the NextGen Interpreter, attribute has been renamed `timeout,` because it has semantics identical to the `timeout` extension attribute of the `<field>` element, and it overrides the `timeout` property. Additionally, the order of precedence is now:

1. `timeout` attribute of the last prompt
2. `timeout` attribute of `<record>`
3. `timeout` property

<script>

Charset Support

The legacy interpreter supports the UTF-8, UTF-16, and ASCII charsets for the `<script>` element. The NextGen Interpreter supports only the UTF-8 and UTF-16 charsets. If any other charset is specified for the `<script>` element, an `error.unsupported.scriptcharset` event is generated when the `<script>` element is executed.

<subdialog>

Passing Objects by Using <param>

When `<param>` is used to pass data to a subdialog, the NextGen Interpreter copies variables by value. For object types, the NextGen Interpreter attempts to duplicate the object in order to pass it by value.

Record variables (created when the `<record>` tag is used) are allowed to be passed to a subdialog. The metadata stored in the copied record object is the *same* as in the original object.

DOM objects created as a result of `<data>,` or from `application.lastresult$.xmlresult,` are not supported. The DOM data will *not* be accessible within the subdialog.

When an ECMAScript object is duplicated within the subdialog, the attributes for array item properties are *not* copied. Function properties are *not* copied either.

VoiceXML 2.1

With VoiceXML 2.1, the World Wide Web Consortium (W3C) has made a number of previously proprietary extensions standard. As a result, the following old extensions are no longer supported in the NextGen Interpreter:

- `com.voicegenie.saveutterance <property>.` Use the input item shadow variable `recording` instead. For further details, see `http://www.w3.org/TR/voicexml21/#sec-reco_reco.`
- The `expr` attribute of `<grammar>.` Use the `srcexpr` attribute instead. For further details, see `http://www.w3.org/TR/voicexml21/#sec-grammar_expr.`
- The `expr` attribute of `<script>.` Use the `srcexpr` attribute instead. For further details, see `http://www.w3.org/TR/voicexml21/#sec-script_expr.`

VoiceGenie Extensions

Namespace Prefixes

In the legacy interpreter, there are several extension elements and attributes that are used without XML namespace prefixes—for example, the `<send>` element, and the `volume` attribute of the `<audio>` element. In the NextGen Interpreter, it is a requirement that all extensions have XML namespace prefixes. The namespace URI is configurable, and it is set to `http://www.voicegenie.com/2006/vxml21-extension` by default.

The following is an example of how to use namespaces in the extension element and attribute:

```
<?xml version="1.0"?>
<vxml version="2.0" xmlns=http://www.w3.org/2001/vxml
xmlns:vg="http://www.voicegenie.com/2006/vxml21-
extension" >
    <form>
        <var name="offset" expr="1+2" />
        <block>
            <vg:send ... />
            <audio src="123" vg:volume="5" />
        </block>
    </form>
</vxml>
```

Call-Control Extensions

In the NextGen Interpreter, the call-control-related extensions are no longer available. Application developers should now use the CCXML capability instead. As a result, the following call-control features are no longer supported in the NextGen Interpreter:

- The call-control extension elements `<call>`, `<fork>`, `<join>`, and `<release>`. Multiple sessions and conferencing are supported through CCXML.
- Conference features. Nevertheless, a conference may be invoked by other means when a new call goes into the media platform. The media platform remains capable of mixing conferences.
- Talk-Listen-Toggle (TLT).
- The `name` and `chan` attributes of `<disconnect>`.
- The call-control events `com.voicegenie.call.*`.
- The call-control session variables.

If these call-control features are required, you must continue to use the legacy interpreter. It is highly recommended that you migrate to use of the CCXML capability, which provides all of these capabilities by using a standards-based markup language.

Note that the `<send>` and `<receive>` elements continue to be supported in the NextGen Interpreter, to support asynchronous messaging.

endbeep

The `endbeep` attribute requests a beep to be played at the end of prompts, just before recognition or recording begins. In the legacy interpreter, if an endbeep is requested, and the prompts have bargein enabled, the beep is not played. In the NextGen Interpreter, the beep is played unless the prompt has been interrupted due to a bargein.

Transfer Events

The event error.connection.nolicense is now replaced by error.connection.noresource.nolicense. This event is thrown when the platform could not establish the outbound call due to a lack of licenses.

_VGGetInfo Function

In the legacy interpreter, the `_VGGetInfo` ECMAScript function is used to provide values for certain information, such as the IP address of the running platform or the version of the media platform. This function has not been implemented in the NextGen Interpreter.

VoiceXML Properties

Initial Setup

In the legacy interpreter, properties specified in the `defaults.vxml` page are set before the application starts. This means that extensions that operate based on property values may behave differently in the NextGen Interpreter. In particular, such extensions include metrics and `savetmpfiles` properties that are used before the main page is initially loaded/started.

Property Prefixes

In the legacy interpreter, several VoiceGenie extensions have been implemented, some of which do not require a company domain prefix. In the NextGen Interpreter, all the extension properties require the company domain as the prefix. The following are the extension properties that do not require a prefix in the legacy interpreter:

- `asrengine`
- `ttsengine`
- `endbeep`
- `loglevel`
- `metricslevel`
- `savetmpfiles`

Note that, in the NextGen Interpreter, the extension prefix for all of the extension properties is configurable; in the Genesys/Voicegenie Kingston release, the default extension is `com.voicegenie.`

Therefore, suppose that you use the following property for the legacy interpreter:

```
<property name=”ASRENGINE” value=”REALSPEAK”/>
```

For the NextGen Interpreter, this property must be changed to:

```
<property name=”com.voicegenie.asrengine”
value=”REALSPEAK”/>
```

Removed Properties

The following properties are no longer supported in the NextGen Interpreter:

- `asrinittimeout`
- `bargeinlevel`
- `vgasrcalllog`
- `vgasrconfidencialutterance`
- `com.voicegenie.asr.nuance.native`
- `com.voicegenie.asrrawambresult`
- `com.voicegenie.wakeupwordranges`
- `com.voicegenie.serverselect`
- `com.voicegenie.fieldobject`
- `com.voicegenie.xmlencoding`
- `com.voicegenie.saveutterance`

expr Attribute of <property>

In the legacy interpreter, the following properties supported the `expr` attribute, so that the property values could be evaluated at runtime:

- `fetchaudio`
- `timeout`

This capability is no longer supported in the NextGen Interpreter.

asrinittimeout

In the legacy interpreter, the `asrinittimeout <property>` is specified as a Time Designation, and the media platform uses it in a formula to calculate the grammar loading timeout. The media platform uses the smaller value of the `timeout <property>` and the `asrinititmeout <property>` to timeout a grammar loading request that is taking too long.

In the NextGen Interpreter, the `com.voicegenie.asr.loadgrammartimeout <property>` is used for this purpose, and this property takes an integer in milliseconds as its value. The `timeout <property>` is no longer involved in calculating the grammar loading timeout.

bargein

In the legacy interpreter, the application can set the value of the `bargein` property to `dtmf.` When this value is set, only DTMF input will stop the prompt, whereas a start-of-speech detected by the speech recognizer will not. The NextGen Interpreter does not currently support this feature.

bargeintype

In the legacy interpreter, in addition to the standard values `hotword` and `speech,` the `bargeintype` property allows the values `recognition` and `energy.` These values were included in previous drafts of VoiceXML 2.0, but they have been dropped from the W3C Recommendation version of VoiceXML 2.0. As a result, setting the `bargeintype` property to these unsupported values will generate an `error.semantic` event when the property is used.

The `bargeintype` property is used to instruct the speech/DTMF recognizers whether they should perform hotword bargein. In the legacy interpreter, the `bargeintype` for a particular recognition session is determined by the property that was in effect when the last prompt in the prompt queue was added. In the NextGen Interpreter, the recognition mode used for a `<field>`

or `<initial>` is the one specified by the `bargeintype` property that was in scope during the execution of the input item. For `<transfer>` and `<record>`, the recognition mode is always `hotword.` As a result, the `bargeintype` property/attribute of a `<prompt>` is ignored.

fetchtimeout

If the application specifies a negative value for the `fetchtimeout` attribute/property, the legacy interpreter ignores the negative value and uses a configured default value. The NextGen Interpreter generates an `error.semantic` event when it uses this property.

inputmodes

In the legacy interpreter, the value of the `inputmodes` property can be set to `both`, as shorthand for `dtmf voice.` In the NextGen Interpreter, this value is no longer supported, and the application should now use a value of `dtmf voice` instead. Invalid values cause an `error.semantic` to be thrown.

Additionally, if the value of `inputmodes` is `voice`, any DTMF input will be ignored in the NextGen Interpreter. In the legacy interpreter, DTMF input will trigger a nomatch.

recordutterance

The `recordutterance` property has been introduced in VoiceXML 2.1. In the legacy interpreter, this property can be used only when the `<vxml>` version is set to `2.1` (otherwise, it is ignored). In the NextGen Interpreter, this property can be used even when the `<vxml>` version is not set to `2.1.`

wakeupwordminimum/wakeupwordmaximum

In the legacy interpreter, if the wakeupwordminimum was greater than the wakeupwordmaximum, the following would be logged as a metric:

```
COM.VOICEGENIE.WAKEUPWORDMINIMUM mustbe less than
COM.VOICEGENIE.WAKEUPWORDMAXIMUM.Disabling wakeupword
feature.
```

In NGI, an error.asr will be thrown.

session.com.voicegenie.transfer.allowed

According to the VoiceXML specification, all session variables should be read-only. Therefore, in the NextGen Interpreter, this function has been replaced by the ECMAScript functions `session.com.voicegenie.setTransferAllowed()` and `session.com.voicegenie.getTransferAllowed()`.

session.connection.answeredby

In the legacy interpreter, if the VoiceXML session is associated with an outbound call, this session variable in a VoiceXML session is used to indicate what type of entity answered the call. It takes one of the following values: `human`, `fax`, `machine`, or `modem`.

This variable is not yet supported in the NextGen Interpreter.

VG Properties

com.voicegenie.maxrecordtime

In the legacy interpreter, this property can be specified only in the platform defaults file. In the NextGen Interpreter, the normal rules of property scoping apply.

Note: No platform default or maximum can be specified.

com.voicegenie.disablerecord

This property is not supported in the NextGen Interpreter.

Miscellaneous Differences

CDATA in SSML

In the legacy interpreter, markup that is contained within a `CDATA` block intended for TTS is not escaped. It is used exactly as it appears within the VoiceXML page.

This means that the following prompt will *not* generate `<mark>`s, but will in fact begin speaking “mark name equals beg slash”:

```
<prompt>
   <![CDATA[
      <mark name = " beg "/>
      12
      <mark name = " xxx "/>
      Go from
      <mark name ="here"/>
      hear to
      <mark name="there and here"/>
      <mark name="end"/>
      there!
      <mark name="endend"/>
   ]]>
</prompt>
```

Sending a Recording to Document to a Server

When a `recording` variable appears as one of the variables in the namelist of a `<subdialog>`, `<submit>`, or `<data>`, if the `method` is not `post`, or if the `enctype` is not `mulitpart/form-data`, the NextGen Interpreter will generate an `error.badfetch`. The legacy interpreter will simply proceed with the `<subdialog>` in these situations.

Form Item Properties

All form item properties and shadow variable properties that belong to a dialog scope are removed when the corresponding `<form>` has ended processing.

Note: This change will not affect your VoiceXML applications; however, if you have external applications/tools that process your logs, it may affect them.

GENESYS
AN ALCATEL-LUCENT COMPANY

Chapter

4 DTMF Recognition

New Implementation

A new DTMF recognizer has been integrated into the media platform. This DTMF recognizer supports only the standard Speech Recognition Grammar Specification (SRGS) grammars (both XML and ABNF grammars; see `http://www.w3.org/TR/speech-grammar/`). Furthermore, it supports only semantic interpretation that conforms to the SISR 1.0 Candidate Recommendation (see `http://www.w3.org/TR/semantic-interpretation/`).

It has been recognized that the changes in semantic interpretation may present a problem for application upgrades; therefore, as an aid to the developer, we attempt to support some simple ABNF grammars, without requiring the full ABNF declaration. The rules are as follows:

The inline DTMF grammar will be prepended by `#ABNF 1.0;$a =` and appended by `;` if an inline DTMF grammar meets all of the following conditions:

- It does not provide a grammar type, or if it provides a type of `application/x-abnf`.
- It does not start with the required SRGS ABNF declaration of `#ABNF 1.0;`.
- The configuration option `vxmli.legacy.simple_dtmf_grammars` has been set to `true`.

This enables the most common shorthand grammar formats to be automatically processed. Note, however, that applications should migrate existing grammars to the formal SRGS XML or ABNF formats.

Note: Extended timing properties have been preserved in the new DTMF recognizer, including critical digit timeout.

Hotword Recognition

In the legacy interpreter, hotword bargein is supported only for speech recognition. In the NextGen Interpreter, both speech *and* DTMF recognition support hotword bargein.

Error Handling

For DTMF grammars containing content that is not in `[0-9abcd*#]`, the NextGen Interpreter throws `error.badfetch`. The legacy interpreter throws `error.grammar.dtmf`.

Chapter

5 Grammars

DTMF Interpretation

In the NextGen Interpreter, DTMF recognition results might have a different formatting when compared with results from the legacy interpreter. For example, using the grammar:

```
<grammar xmlns="http://www.w3.org/2001/06/grammar"
mode="dtmf" version="1.0" root="my_root2"
type="application/srgs+xml">
        <rule id="my_root2" scope="public">
        <one-of>
            <item>12345</item>
        </one-of>
        </rule>
    </grammar>
```

With input ‘12345’, this grammar generates a result of “12345” in the legacy interpreter. In the NextGen Interpreter, the result is “1 2 3 4 5”.

External Grammars

Error Handling

In the legacy interpreter, `error.asr.grammar` could be thrown in some cases, such as:

- Fetching the grammar failed due to a timeout.
- The grammar does not exist.
- A referenced rule is not defined.
- A grammar syntax error is detected.

In the NextGen Interpreter, `error.badfetch` will be thrown instead, as specified in the VoiceXML specification.

External Grammar Fetching

The legacy interpreter includes a configuration option that instructs the interpreter to fetch external grammars, and to pass the local address of these grammars to the speech engine. This capability is no longer supported in the NextGen Interpreter.

All currently supported speech engines are fully capable of directly fetching and caching external grammars, as required by the SRGS.

Inline Grammars

Strict Parsing

One consequence of the NextGen Intepreter's strict parsing is that it will reject some malformed inline XML grammars that were accepted by the legacy interpreter.

In particular, some of the elements/attributes in older versions of the SRGS standards (such as the `<count>` element and the `tag` attribute) are no longer supported.

To avoid this issue, ensure that inline grammars are properly formed. Also ensure that grammars are compliant with the current release of the SRGS.

Attributes

Type

To improve conformance with the VoiceXML specification, the NextGen Interpreter requires that inline grammars specify the `type` attribute.

Chapter

6 Changes to Metrics and Logging

Every effort has been made to replicate the format of the legacy interpreter's metrics file as closely as possible in the NextGen Interpreter. Nevertheless, there are several differences between the two interpreters. For example, metrics are not guaranteed to be logged in the same sequence as before. Additionally, the formatted output may have changed.

Note: These changes should be taken into account for external applications that perform metrics file processing. If you currently rely on the parsing of the metrics file data for reporting or billing purposes, your parsing application may need to be updated.

Tools Related INFO Messages

Script Operations

In the NextGen Interpreter, the Script Operations `INFO` message is logged only when:

- Executing a `<field>.`
- Executing a `<transfer>.`
- Executing a `<record>.`
- Ending a call.

Compile Time

In the NextGen Interpreter, the Compile Time `INFO` message is logged immediately after a page is compiled. No message is logged if the page was cached. Additionally, for the first pages of a session (defaults, main page, and root page), the message is logged prior to initialization, and therefore the value of `com.voicegenie.metricslevel` used at that time is specified in the configuration via the SMC (or `callmgr.cfg`).

Metrics

In the NextGen Interpreter, the level of a metric is configurable via the SMC/configuration. This differs from the legacy interpreter, where each metric had an assigned level that could not be changed. The provided default values are the same as those found in the legacy interpreter. Note that changing these values may disable some of the functionality of the VoiceGenie Tools, such as the Quality Advisor and Call Analyst.

appl_begin

This metric is now logged before starting the execution of the first page.

Note: The properties that are currently in effect are the properties specified in the interpreter configuration, *not* in `default.vxml.`

appl_end

This metric is now logged after all pages have completed execution.

Note: The properties that are currently in effect are the properties specified in the interpreter configuration.

asr_trace

The format of the result string differs, in that the NextGen Interpreter provides the full raw result returned from the ASR engine.

Examples:

```
asr_trace ASR_DONE:results:<?xml version='1.0'?><result><interpretation
   grammar="session:0x0026" confidence="90"><input
   mode="speech">tea</input><instance/></interpretation></result>

asr_trace ASR_DONE:results:<?xml version='1.0'?><result><interpretation
   grammar="session:0x007b6" confidence="97"><input
   mode="speech">No</input><instance><SWI_meaning>{SWI_literal:No}</SWI_me
   aning></instance></interpretation></result>
```

call_appl

This metric is now logged before starting the execution of the first page.

Note: The properties that are currently in effect are the properties specified in the interpreter configuration, *not* in `default-ng.vxml`.

dtmf

The parameter `action` is no longer specified.

Examples:

```
dtmf :3
dtmf :99
```

dtmf_end

This metric is no longer recorded.

exec_error

The legacy format is not precisely followed. Instead, what is a logged is a free-form message.

Example:

```
exec_error TypeError: session.transfer has no properties
```

exec_warning

This metric is no longer recorded.

form_exit

The NextGen Interpreter logs only `normal` as a `reason`; it never logs `internal_error`.

input_end

This metric now contains much more information. The format is:

```
input_end
[reason]|[mode]|[grammar_scope]|[grammar_url]|[phrase]|[conf
idence]
```

Where `reason` can be one of the following: `DISCONNECTED`, `FAILED`, `NO_INPUT`, `ERROR`, `NO_MATCH`, `MAX_SPEECH_TIMEOUT`, `ASR_MAXSPEECHTIMEOUT`, `RECORD_END`, or `TRANSFER_END`.

Examples:

```
input_end MATCHED|dtmf|Field|inline|991058|1.000000
input_end NO_INPUT|||||
input_end
   MATCHED|dtmf|Field|file:///usr/local/phoneweb/samples/testapp/test.vxml
   |2|1.000000
```

log

The format is now:

```
log <label>:<message><expression>
```

parse_error

In the NextGen Interpreter, the `application` name and `line` number are always empty.

Example:

```
parse_error (http://138.120.72.51/testapp/test3.vxml,, line ):Invalid
   child element Block in element Catch at line 15
```

parse_warning

In the NextGen Interpreter, the `application` name and `line` number are always empty.

Example:

```
parse_warning (http://10.0.0.1/property_vgaudiostop.vxml,,):Unknown
   property name: VGSTOP
```

prompt

This metric now has no data. It simply provides an indication that one or more prompts will be played. Each individual prompt is identified from the metric `prompt_play`.

prompt_play

This metric has the same data as the original `prompt` metric, but for only one item of a prompt queue. Also, `filename` is no longer provided. The new format is:

```
<type>|<data>
```

Examples:

```
prompt_play audio|http://127.0.0.1/audio/goodbye.vox
prompt_play tts|<?xml version="1.0" encoding="UTF-8"?><speak
   version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xml:lang="en-
   US">Please press a number between 1 and 5.</speak>
```

Note: This metric is new for 7.2

prompt_end

In the NextGen Interpreter, `input` is always empty, and `hangup` is no longer offered as a value for `reason`.

Examples:

```
prompt_end done
prompt_end dtmfbargein
prompt_end aborted
```

record_end

In the NextGen Interpreter, there are two changes:

- A local grammar match is not logged as an `outcome;` `RECORD SUCCESS` is logged instead.
- `MSG_INTERRUPT` is not logged as a `term reason.`

Example:

```
record_end :RECORD
   SUCCESS|MAXTIME|2000|audio/basic|/usr/local/phoneweb/tmp/00020069-
   100000A9/3262_2884936863.00020069-100000A9.ulaw
```

subdialog_return

The format of this metric is dependent on the return type, and it has changed in the NextGen Interpreter.

For `event`, the format is:
`event|<event name>`

For `namelist`, the format is:
`namelist|<the evaluated namelist>`

subdialog_start

The `param_name` and `param` values are now listed as one entry, encoded in JavaScript Object Notation (see `http://json.org/`).

submit

The `namelist` value is now encoded in JavaScript Object Notation (see `http://json.org/`).

Index