WebSphere Voice Server

Speaker Verification guide

Version 6.1.1

Note: Before using this information and the product it supports, be sure to read the information in "Notices."

Second Edition (December 2008)

This edition applies to the IBM WebSphere Voice Server version 6.1.1. It applies to all subsequent releases and modifications until otherwise indicated in new editions.

Address your comments to:
International Business Machines Corporation
Department 6R4A
P.O. Box 12195
Research Triangle Park, North Carolina
27709-2195

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2007, 2008. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 2007,, 2008

Chapter 1. About this guide

This guide provides information about the speaker verification component, which is a part of IBM® WebSphere® Voice Server.

This chapter includes the following topics:

- Who should read this guide
- Related publications
- How this guide is organized
- Document conventions and terminology

Who should read this guide?

Read this guide if you:

- Want to find out more about IBM's speaker verification technology.
- Are working on a solution/application that utilizes the speaker verification component of the IBM® WebSphere® Voice Server.

Related publications

Reference, design, and programming information for creating voice applications is available from a variety of sources, as represented by the documents listed in this section.

NOTE: Guidelines and publications cited in this guide are for your information only and do not in any manner serve as an endorsement of those materials. You alone are responsible for determining the suitability and applicability of this information to your needs.

Specifications and standards

You may want to refer to the following sources for information about relevant specifications and standards:

- *High-Performance Text-Independent Speaker Verification*, available at **http://www.research.ibm.com/CBG/papers/icassp03_syspaper.pdf**
- *Voice Extensible Markup Language (VoiceXML) Version 2.1* specification, published by W3C and available at **http://www.w3.org/TR/voicexml21/**
- *Voice Extensible Markup Language (VoiceXML) Version 2.0* specification, published by W3C and available at **http://www.w3.org/TR/voicexml20/**
- *HTTP 1.1 Specification*, available at **http://www.ietf.org/rfc/rfc2616.txt**
- *HTTP State Management Mechanism (Cookie Specification)*, available at **http://www.w3.org/Protocols/rfc2109/rfc2109**
- *Speaker Identification and Verification Requirements for VoiceXML Applications*, available at **http://www.voicexml.org/resources/SIV_requirements_for_VoiceXML_20050914.pdf**

© Copyright IBM Corp. 2007,, 2008

Speech user interface design

The speech user interface guidelines presented in this guide are an evolving set of recommendations based on industry research and lessons learned in the process of developing our own VoiceXML and telephony applications. For more information, refer to speech industry literature and publications such as the following sources:

- *Audio System for Technical Readings (ASTeR)* by T. V. Raman, a Ph.D. thesis published by Cornell University, May 1994.
- *Auditory User Interfaces—Towards The Speaking Computer* by T. V. Raman, published by Kluwer Academic Publishers, August 1997.
- "Directing the Dialog: The Art of IVR" by Myra Hambleton, published in *Speech Technology*, Feb/Mar 2000.
- *Handbook of Human-Computer Interaction* by Thomas K Landauer, Martin Helander, and Prasad V. Prabhu, published by Elsevier Science, Amsterdam, North Holland, June 1997.
- *How to Build a Speech Recognition Application: A Style Guide for Telephony Dialogues* (Second Edition) by Bruce Balentine, David P. Morgan, and William S. Meisel, published by Enterprise Integration Group, San Ramon, CA, 2001.
- *Human Factors and Voice Interactive Systems* by Daryle Gardner-Bonneau, published by Kluwer Academic Publishers, Boston, MA, March 1999.

Server-side programming

Information about server-side programming is available from a number of sources, including the following:

- *Design and Implement Servlets, JSPs, and EJBs for IBM WebSphere Application Server* (IBM Redbooks) SG24-5754-00
- *Developing an e-business Application for the IBM WebSphere Application Server* (IBM Redpiece) SG24-5423-00
- *JavaServer Pages (JSP)* at **http://java.sun.com/products/jsp**
- *Java™ Servlet* at **http://java.sun.com/products/servlet/**
- *Servlet/JSP/EJB Design and Implementation Guide* (IBM Redbooks) SG24-5754-00

Deployment information

For information about deploying your voice applications, refer to the documentation provided with WebSphere Voice Server for Multiplatforms.

The following documentation is provided in softcopy only with the product, or can be downloaded from the IBM Publications Center:

- *IBM Text-to-Speech SSML Programming Guide*
- *WebSphere Voice Server for Multiplatforms Information Center*
- *WebSphere Voice Server for Multiplatforms: Administrator's Guide*, G210-1561

Document conventions and terminology

The following conventions are used to present information in this document:

Italic Used for emphasis, to indicate variable text, and for references to other documents.
Bold Used for configuration parameters, file names, URLs, and user-interface controls such as command buttons and menus.
Monospace Used for sample code.
<*text*> Used for editorial comments in scripts.

Chapter 2. Introduction

Speaker verification is the ability of verifying that a person is who they say they are, based on the analysis of their voice. Each person has unique features associated with their voice. The extraction of these features through the analysis of incoming audio allows the creation of a unique speaker model containing distinct characteristics associated with a person's voice. This model is known a voiceprint.

Note that speaker verification is different from speaker identification (the ability to identify any speaker from all known speakers by voice alone). The process of speaker verification consists of two distinct steps, enrollment and verification. Both steps require a unique identifier or claim of association with the person being enrolled or verified.

Enrollment is the process of creating a person's voiceprint. It involves someone speaking for a certain period of time and the speaker verification system processing the audio. During enrollment, a unique voiceprint identifier (claim) must also be provided by the user (or by the enrollment application itself). After enrollment is complete, the user can use the voiceprint identifier for further verification requests.

The actual process of verification takes place after someone is successfully enrolled with the system. This process, also referred to as *scoring*, also involves the same concepts used for enrollment. The user provides his or her voiceprint identifier, and the system fetches the enrolled voiceprint associated with it and starts analyzing the audio. During this process, the same features and characteristics analyzed during the voiceprint creation are now used to compare with the information stored in the voiceprint itself. The outcome of this analysis is a similarity score, which tells how close the analyzed voice is to the one that originated the voiceprint. This score can be used as a ***hint*** to the application, which can grant or deny access to the user.

IBM's speaker verification technology is text independent, therefore does not require any kind of speech grammars to function. Any text can be said for both enrollment and verification phases. Note that text independence naturally implies language independence. But, for best results, IBM recommends use of the same language for enrollment and verification.

IBM recommends the usage of its speaker verification technology as an extra (not the only) layer of security for speech applications/solutions. Robust applications should consider this technology in conjunction with other security mechanisms such as knowledge-based verification such as secret passwords.

Functionality Overview

The speaker verification component provided by the WebSphere® Voice Server allows speech applications (VoiceXML) to access IBM's speaker verification technology. VoiceXML applications can submit audio to be verified by the voice server through a simple Web interface.

In terms of voiceprint storage, this product includes a file-based voiceprint repository, which should be exclusively used for evaluation and/or application development and testing. Customers must ensure they implement a production voiceprint repository (e.g., RDB-based) prior to solution deployment. For more information about how to implement a voiceprint repository, see the Voiceprint API section.

Chapter 3. Developing

From a speech application point of view, speaker verification is performed in two distinct steps. The first step is referred to as the **designation phase**, where the application interacts with the caller to establish his or her identity (claim). Note that this phase does not necessarily involve any speaker verification resource as the claim is normally gathered using only speech (or DTMF) recognition. After the claim is established, the application enters the **execution phase** for actual interaction with the speaker verification resource through enrolling or verifying the caller.

The WebSphere® Voice Server component for speaker verification provides a Web interface for VoiceXML applications. Applications can perform speaker verification by submitting requests to a Web Connector module deployed during product installation. The Web Module is in charge of communicating locally with the voice server, but this communication is transparent to the application developer. Therefore, the Web interface is the only public interface exposed by the WebSphere® Voice Server for accessing its speaker verification resources.

Note that it is the application's responsibility to gather and submit the claim as well as the audio to be verified. As far as speaker verification is concerned, there are no restrictions in terms of how the application is instrumented to collect the audio. This is normally done though standard VoiceXML elements, such as the <*record*> tag or the *recordutterance* property.

The Web interface provides two modes of operation: **session bound** and **session unbound**. When operating in session unbound mode, every request is treated as an atomic operation. For example, you might want to have an application record someone's enrollment phrase in a single utterance, so you use the session unbound mode to send a single request to process the audio. Session bound mode allows multiple requests with audio to be submitted throughout the lifetime of a session. In this mode, an application may, for example, break the enrollment phrase into multiple utterances, using multiple requests to submit the audio. The web interface concatenates the audio samples from the multiple requests into a single audio sample for verification. You might choose to use a session bound operation if a single utterance is too short for verification but the total concatenated utterances would suffice.

As a rule of thumb, the session unbound mode is provided as more convenient for developers. But it is expected and recommended that robust applications opt for the use of session bound mode, since it allows better integration from a user experience point of view. Session bound instrumentation is also more likely to provide better results, as applications can be written to score/verify over multiple utterances, improving overall efficiency.

Audio Considerations

Input audio format

Speaker verification supports both G711-ulaw and G711-alaw audio input, both sampled at 8KHz rate, 8bit, mono. The audio format expected by the speaker verification engine is configurable (see Chapter 6. Configuring for more details).

NOTE: Converting audio from one format to another (a-law to u-law and vice versa) before submitting for verification is not supported as it impacts the accuracy. This kind of conversion normally takes place on the Interactive Voice Response (IVR) side and should be avoided.

Audio length required for enrollment

The amount of audio used to create a voiceprint has a direct effect on the system's overall accuracy. Ideally, applications should be designed to provide as much audio as possible during enrollment. The following table can be used as a reference.

Table 1: Audio quality

Enrollment Duration (s)	Accuracy Quality
30-45	AVERAGE
45-60	GOOD
>60	EXCELENT

Keep in mind, however, that several factors (e.g., background noise, audio quality, and volume) influence the amount of audio actually processed by the speaker verification engine. The engine tries to balance these discrepancies in signal quality but, when developing and testing a speaker verification application, be aware that these factors may have a significant impact on the numbers displayed in the table above.

Internally, the speaker verification engine counts the number of frames actually processed during enrollment. If, at the end of the enrollment process, this counter is not higher than its internal threshold, the enrollment fails with an 'undecided' return code (see the Speaker Verification Web API section for more details).

© Copyright IBM Corp. 2007,, 2008

Requirements and limitations

The following requirements must be met by all clients (applications, IVR platforms, etc.) in order to use the speaker verification API provided by the WebSphere® Voice Server.

Table 2: Requirements

Requirement	Reason
VoiceXML 2.1	All responses returned by the web connector use version 2.1 in the top-level document. Use of <*record.utterance*> allows applications to perform speech recognition and speaker verification in the same utterance.
HTTP Cookie support	Required for session bound mode. Cookie is used to store/retrieve the verification session to/from the HTTP session.

The product currently has the following limitations:

Table 3: Limitations

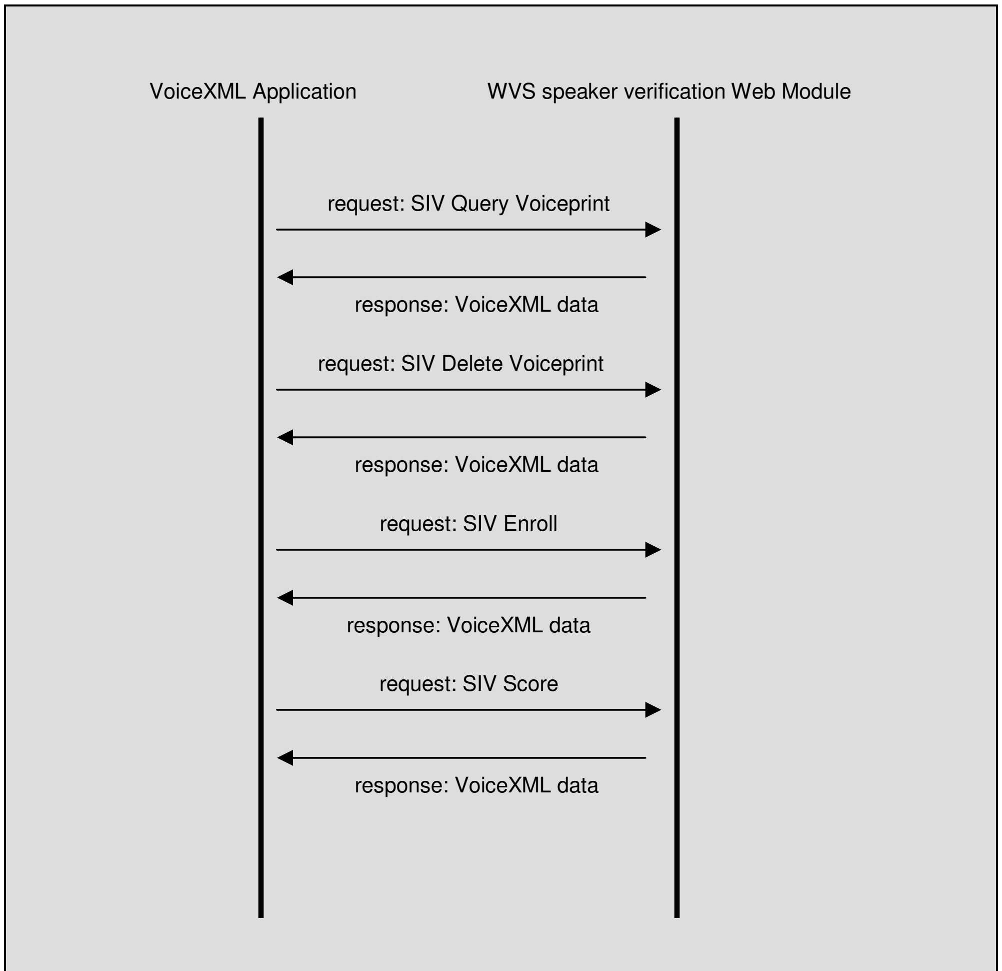
Limitation	Description
No HTTP support for input-wave-uri header.	This header only supports the use of file://. Therefore, all audio files submitted for verification using this header must be located in the WVS box where speaker verification has been deployed.
No UTF-8 support for voiceprint parameter	Valid voiceprint names are required to be in US-ASCII characters (character code 33 (decimal) to 126 (decimal))

Speaker Verification Web API

All requests are tunneled through the HTTP protocol. Requests containing audio are expected to be in the multi-part message format defined by the HTTP specification (refer to Section 19.2 of the HTTP 1.1 specification listed in the reference section for further information). All HTTP responses include a message body containing VoiceXML data to be interpreted by the application.

Session unbound mode

When using this mode, all operations are performed independently of one another. Internally, the speaker verification Web Module establishes a session with the server only to handle a specific request, disconnecting from the server immediately after returning a response.



Available API calls: session unbound mode

SIV Query Voiceprint

Purpose

Checks if a given voiceprint is available in the repository. In other words, allows applications to check if a given user has already been enrolled.

URI

http://<hostname>[:port]/ibmsiv/queryVP

Parameters

voiceprint *(required)*

The unique ID associated with a voiceprint.

repository-uri *(optional)*

The location of the voiceprint repository (if not specified, defaults to configured repository).

Returns

HTTP response body contains the following VoiceXML excerpt:

```
<vxml>
      <form id="siv">
      <var name="result" expr="new Object();"/>
      <block>
      <assign name="result.exists" expr="true | false"/>
      <assign name="result.id" expr="<voiceprint>"/>
      </block>
      </form>
</vxml>
```

Where

result.exists – True if voiceprint exists, false otherwise.
result.id – The specified voiceprint identifier.

Example

The VoiceXML snippet below is an example of how to query the voiceprint repository.

```
<block name=”SIV_QUERY_VP”>

  <!-- assuming variable ‘claim’ containing voiceprint id has already -->
  <!-- been collected by the application                             -->
  <var name=”voiceprint” expr=”application.claim”/>

  <subdialog name="sivquery"
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-
name>:<port-number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/queryVP?"
             method="get"
             namelist=”voiceprint”>
  <filled>
      <if cond="sivquery.result.exists == 'false'">
         <log>
             SIV_QUERY_VP::voiceprint <value expr="voiceprint"/>
             not found.
         </log>
      <else/>
         <log>
             SIV_QUERY_VP::voiceprint <value expr="voiceprint"/>
             has been found in the repository
        </log>
    </if>
  </filled>

  <catch event="error.siv.failure">
    <log>
        SIV_QUERY_VP:: ev=<value expr="_event"/>
        msg=<value expr="_message"/>
    </log>
  </catch>

  </subdialog>

</block>
```

SIV Delete Voiceprint

Purpose

Deletes the specified voiceprint from the repository. This call has no effect if the voiceprint does not exist.

URI

http://<hostname>[:port]/ibmsiv/deleteVP

Parameters

voiceprint *(required)*

The unique ID associated with a voiceprint.

repository-uri *(optional)*

The location of the voiceprint repository.
(If not specified, defaults to configured repository.)

Returns

HTTP response body contains the following VoiceXML excerpt:

```
<vxml>
       <form id=”siv”>
       <var name=”result” expr=”new Object();”/>
       <block>
       <assign name=”result.deleted” expr=”true | false“/>
       <assign name=”result.id” expr=”<voiceprint>”/>
       </block>
       </form>
</vxml>
```

Where

result.deleted – True if voiceprint exists and has been deleted. False if voiceprint exists and could not be deleted. True if voiceprint does not exist.
result.id – The specified voiceprint identifier.

Example

The VoiceXML snippet below is an example of how to delete a voiceprint from the repository.

```
<block name=”SIV_DELETE_VP”>

  <!-- assuming variable ‘claim’ containing voiceprint id has already -->
  <!-- been collected by the application                             -->
  <var name=”voiceprint” expr=”application.claim”/>

  <subdialog name="sivdelete"
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-
name>:<port-number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/deleteVP’”
             method="get"
             namelist=”voiceprint”>
  <filled>
     <log>
         SIV_DELETE_VP::voiceprint <value expr="sivdelete.result.id"/>
         has been deleted.
     </log>
  </filled>

  <catch event="error.siv.failure">
    <log>
        SIV_DELETE_VP:: ev=<value expr="_event"/>
        msg=<value expr="_message"/>
    </log>
  </catch>

  </subdialog>

</block>
```

© Copyright IBM Corp. 2007,, 2008

SIV Enroll

Purpose

Enrolls by creating a new voiceprint or updating (adapting) an existent voiceprint, based on the audio provided with the request. Audio data MUST be provided (except if the input-wave-uri parameter is specified), encapsulated in the request body (multipart type), according to HTTP 1.1 specifications.

URI (POST)

http://<hostname>[:port]/ibmsiv/enroll

Parameters

voiceprint *(required)*

The unique ID associated with a voiceprint.

repository-uri *(optional)*

The location of the voiceprint repository.
(If not specified, defaults to configuration property.)

input-wave-uri *(optional)*

If available, specifies the location of the audio to be used for processing.
NOTE: Only supports *file* protocol.

Returns

HTTP response body contains the following VoiceXML excerpt:

```
<vxml>
       <form id=”siv”>
       <var name=”result” expr=”new Object();”/>
       <block>
       <assign name=”result.decision”
               expr=”undecided | accepted‟/>
       <assign name=”result.id” expr=”<voiceprint>”/>
       </block>
       </form>
</vxml>
```

Where

result.decision – The enrollment result; one of the following:

undecided – application did not provide enough audio to complete enrollment
accepted – enrollment completed successfully

result.id – The specified voiceprint identifier.

Example

The VoiceXML snippet below is an example of how to submit audio to enroll a voiceprint.

```
<block name=”SIV_ENROLL”>

  <!-- assuming variable ‘claim’ containing voiceprint ID has already -->
  <!-- been collected by the application                              -->
  <var name=”voiceprint” expr=”application.claim”/>

  <!-- also assuming that utterance has already been recorded and     -->
  <!-- the audio is available through the variable ‘utterance’        -->
  <subdialog name="sivenroll"
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-
name>:<port-number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/enroll?voiceprint='+voiceprint"
             method="post"
             enctype=”multipart/form-data”
             namelist=”application.utterance”>
  <filled>
     <if cond=”sivenroll.result.decision == ‘accepted’”>
        <log>
         SIV_ENROLL::done enrolling <value expr="sivenroll.result.id"/>
        </log>
     <else />
        <log>
         SIV_ENROLL::Unable to create voiceprint. Not enough audio.
        </log>
     </if>
  </filled>

  <catch event="error.siv.failure">
    <log>
        SIV_ENROLL:: ev=<value expr="_event"/>
        msg=<value expr="_message"/>
    </log>
  </catch>

  </subdialog>

</block>
```

SIV Score

Purpose

Perform speaker verification using the specified voiceprint as the claim and the provided audio for processing. The voiceprint associated with the claim must be available (enrolled) in the repository. The returned result contains the similarity score and a decision based on the configured acceptance threshold. Audio data MUST be provided (except if the input-wave-uri parameter is specified), encapsulated in the request body (multipart type), according to HTTP 1.1 specifications. Note that it is up to the application to honor the decision returned by the server.

URI (POST)

http://<hostname>[:port]/ibmsiv/score

Parameters

voiceprint *(required)*

The unique ID associated with a voiceprint.

repository-uri *(optional)*

The location of the voiceprint repository.
(If not specified, defaults to configuration property.)

input-wave-uri *(optional)*

If available, specify the location of the audio to be used for processing.
(**NOTE**: Only supports *file* protocol.)

Returns

HTTP response body contains the following VoiceXML excerpt:

```
<vxml>
       <form id=”siv”>
       <var name=”result” expr=”new Object();”/>
       <block>
       <assign name=”result.decision”
               expr=”undecided | rejected | accepted“/>
        <assign name=”result.score” expr=”<value>”/>
       <assign name=”result.id” expr=”<voiceprint>”/>
       </block>
       </form>
</vxml>
```

Where

result.decision – The suggested decision; one of the following:

undecided – application did not provide enough audio to produce a valid score
rejected – the generated score is lower than the acceptance threshold
accepted – the generated score is higher than the acceptance threshold

result.score - The similarity score produced by the engine, ranging from -1 to 1.
result.id – The specified voiceprint identifier.

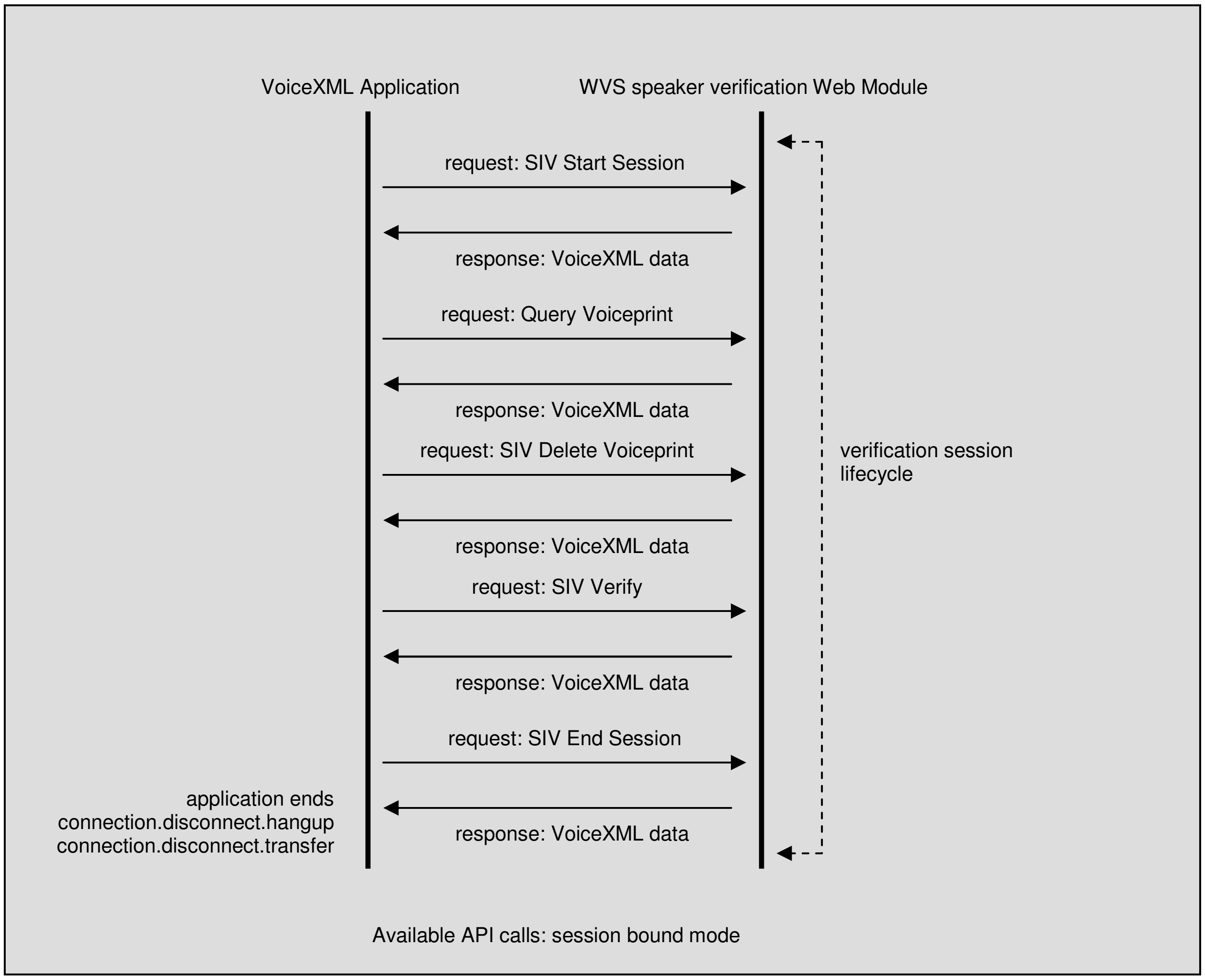
Example

The VoiceXML snippet below is an example of how to verify (score) someone's voice.

```
<block name=”SIV_SCORE”>

  <!-- assuming variable ‘claim’ containing voiceprint ID has already -->
  <!-- been collected by the application                              -->
  <var name=”voiceprint” expr=”application.claim”/>

  <!-- also assuming that utterance has already been recorded and     -->
  <!-- the audio is available through the variable ‘utterance’        -->

  <subdialog name="sivscore"
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-
name>:<port-number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/score?voiceprint='+voiceprint"
             method="post"
             enctype=”multipart/form-data”
             namelist=”application.utterance”>
  <filled>
     <log>
         SIV_SCORE::score <value expr=”sivscore.result.score”/>
         SIV_SCORE::decision <value expr=”sivscore.result.decision”/>
     </log>
     <if cond=”sivscore.result.decision == ‘accepted’”>
        <log>
         SIV_SCORE::access granted.
        </log>
     <elseif cond=”sivscore.result.decision == `undecided’”/>
        <log>
         SIV_SCORE::Need more audio to make a decision.
        </log>
     <elseif cond=”sivscore.result.decision == ‘rejected’”/>
        <log>
         SIV_SCORE::access denied.
        </log>
     </if>
  </filled>

  <catch event="error.siv.failure">
    <log>
        SIV_SCORE:: ev=<value expr="_event"/>
        msg=<value expr="_message"/>
    </log>
  </catch>

  </subdialog>

</block>
```

Session bound mode

When operating in this mode, the speaker verification Web module establishes a session with the server and keeps that session open until the client explicitly tells it to disconnect or an internal timeout expires. In this mode, multiple operations can be performed, targeting the same session on the server side. The verification process may be split transparently into multiple utterances, with the final outcome being the same as having a single (longer) utterance. There is no need for the application to concatenate the individual audio streams as this is handled by the API. Note that this mode of operation requires HTTP cookie support on the client side.


VoiceXML Application
WVS speaker verification Web Module
request: SIV Start Session
response: VoiceXML data
request: Query Voiceprint
response: VoiceXML data
request: SIV Delete Voiceprint
response: VoiceXML data
request: SIV Verify
response: VoiceXML data
request: SIV End Session
response: VoiceXML data
verification session lifecycle
application ends
connection.disconnect.hangup
connection.disconnect.transfer
Available API calls: session bound mode

SIV Start Session

Purpose

Establish an enrollment or verification session with the server. The parameters specified are persistent throughout the lifetime of the session. If called during an ongoing verification session, the request automatically causes the previous session to be aborted.

URI

http://<hostname>[:port]/ibmsiv/startsession

Parameters

voiceprint *(required)*

The unique ID associated with a voiceprint.

repository-uri *(optional)*

The location for the voiceprint repository.
(If not specified, defaults to configured repository.)

mode *(optional)*

The verification mode ('score' | 'enroll').
(If not available, defaults to 'score'.)

Returns

The HTTP response body contains the following VoiceXML excerpt:

```
<vxml>
       <form id="siv">
       <var name="result" expr="new Object();"/>
       <block>
       </block>
       </form>
</vxml>
```

Example

The VoiceXML snippet below is an example of how to establish a verification session.

```
<block name=”SIV_START_SESSION”>

  <!-- assuming variable ‘claim’ containing voiceprint ID has already -->
  <!-- been collected by the application                              -->

  <var name=”voiceprint” expr=”application.claim”/>
  <var name=”mode” expr=”’score’”/>

  <subdialog name="startsession"
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-
name>:<port-number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/startsession'
             method=”get”
             namelist=”voiceprint mode">
  <filled>
      <log>
         SIV_START_SESSION::session established for
         voiceprint <value expr="voiceprint"/>
      </log>
  </filled>

  <catch event="error.siv.failure">
    <log>
        SIV_START_SESSION:: ev=<value expr="_event"/>
        msg=<value expr="_message"/>
    </log>
  </catch>

  </subdialog>

</block>
```

SIV End Session

Purpose

End a previously established verification session. All resources associated with the session are released. This request has no effect if there is no active session established for the caller. **NOTE**: In case of successful enrollment, the associated voiceprint is only created or updated by the server when the session is terminated. Therefore, applications MUST end the enrollment session before verifying the enrolled/adapted voiceprint.

URI

http://<hostname>[:port]/ibmsiv/endsession

Parameters

None

Returns

The HTTP response body contains the following VoiceXML excerpt:

```
<vxml>
       <form id="siv">
       <var name="result" expr="new Object();"/>
       <block>
       </block>
       </form>
</vxml>
```

Example

The VoiceXML snippet below is an example of how to end a verification session.

```
<block name=”SIV_END_SESSION”>

  <subdialog name="endsession"
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-
name>:<port-number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/endsession'”
             method=”get”>
  <filled>
      <log>
         SIV_END_SESSION::session ended normally.
      </log>
  </filled>

  <catch event="error.siv.failure">
    <log>
        SIV_START_SESSION:: ev=<value expr="_event"/>
        msg=<value expr="_message"/>
    </log>
  </catch>

  </subdialog>

</block>
```

SIV Query Voiceprint

Purpose

Checks if a given voiceprint is available in the repository. In other words, allow applications to check if a given user has already been enrolled.

URI

http://<hostname>[:port]/ibmsiv/queryVP

Parameters

voiceprint *(required)*

The unique ID associated with a voiceprint.

repository-uri *(optional)*

The location for the voiceprint repository.
(If not specified, defaults to the session repository.)

Returns

The HTTP response body contains the following VoiceXML excerpt:

```
<vxml>
      <form id=”siv”>
      <var name=”result” expr=”new Object();”/>
      <block>
      <assign name=”result.exists” expr=”true | false“/>
      <assign name=”result.id” expr=”<voiceprint>”/>
      </block>
      </form>
</vxml>
```

Where

result.exists – True if voiceprint exists, false otherwise.
result.id – The specified voiceprint identifier.

Example

The VoiceXML snippet below is an example of how to query the voiceprint repository.

```
<block name=”SIV_QUERY_VP”>

  <!-- assuming verification session previously established           -->
  <!-- also assuming variable ‘claim’ containing voiceprint ID has    -->
  <!-- already been collected by the application                      -->
  <var name=”voiceprint” expr=”application.claim”/>

  <subdialog name="sivquery"
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-
name>:<port-number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/queryVP?"
             method="get"
             namelist=”voiceprint”>
  <filled>
      <if cond="sivquery.result.exists == 'false'">
         <log>
             SIV_QUERY_VP::voiceprint <value expr="voiceprint"/>
             not found.
         </log>
      <else/>
         <log>
             SIV_QUERY_VP::voiceprint <value expr="voiceprint"/>
             has been found in the repository
        </log>
    </if>
  </filled>

  <catch event="error.siv.failure">
    <log>
        SIV_QUERY_VP:: ev=<value expr="_event"/>
        msg=<value expr="_message"/>
    </log>
  </catch>

  </subdialog>

</block>
```

SIV Delete Voiceprint

Purpose

Deletes the specified voiceprint from the repository. This call has no effect if the voiceprint does not exist.

URI

http://<hostname>[:port]/ibmsiv/deleteVP

Parameters

voiceprint *(required)*

The unique ID associated with a voiceprint.

repository-uri *(optional)*

The location for the voiceprint repository.
(If not specified, defaults to session repository.)

Returns

The HTTP response body contains the following VoiceXML excerpt:

```
<vxml>
       <form id=”siv”>
       <var name=”result” expr=”new Object();”/>
       <block>
       <assign name=”result.deleted” expr=”true | false“/>
       <assign name=”result.id” expr=”<voiceprint>”/>
       </block>
       </form>
</vxml>
```

Where

result.deleted – True if voiceprint exists and has been deleted. False if voiceprint exists and could not be deleted. True if voiceprint does not exist.
result.id – The specified voiceprint identifier.

Example:

The VoiceXML snippet below shows an example of how to delete a voiceprint from the repository.

```
<block name="SIV_DELETE_VP">

  <!-- assuming verification session previously established           -->
  <!-- also assuming variable ‘claim’ containing voiceprint ID has    -->
  <!-- already been collected by the application                      -->
  <var name="voiceprint" expr="application.claim"/>

  <subdialog name="sivdelete"
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-
name>:<port-number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/deleteVP"
             method="get"
             namelist="voiceprint">
  <filled>
     <log>
         SIV_DELETE_VP::voiceprint <value expr="sivdelete.result.id"/>
         has been deleted.
     </log>
  </filled>

  <catch event="error.siv.failure">
    <log>
        SIV_DELETE_VP:: ev=<value expr="_event"/>
        msg=<value expr="_message"/>
    </log>
  </catch>

  </subdialog>

</block>
```

SIV Verify

Purpose

Enrolls or gets the score for the session voiceprint. Audio data MUST be provided (except if input-wave-uri parameter is specified) and encapsulated in the request body (multipart type), according to HTTP 1.1 specifications.

URI (POST)

http://<hostname>[:port]/ibmsiv/verify

Parameters

input-wave-uri *(optional)*

If available, specify the location of the audio to be used for processing.
NOTE: Only supports *file* protocol.

Returns

The HTTP response body contains the following VoiceXML excerpt.

```
<vxml>
       <form id=”siv”>
       <var name=”result” expr=”new Object();”/>
       <block>
       <assign name=”result.decision”
               expr=”undecided | rejected | accepted“/>
        <assign name=”result.score” expr=”<value>”/>
       <assign name=”result.id” expr=”<voiceprint>”/>
       </block>
       </form>
</vxml>
```

Where

result.decision – The suggested decision; one of the following:

undecided – If enrolling, the application did not provide enough audio to create a voiceprint. If scoring, there is not enough audio to generate a valid score.
rejected – If enrolling, not applicable. If scoring, the generated score is lower than the acceptance threshold (score only).
accepted – If enrolling, the voiceprint has been enrolled successfully. If scoring, the generated score is higher than the acceptance threshold.

result.score – the similarity score produced by the engine, ranging from -1 to 1. If enrolling, ignore (always 0.0).

result.id – The specified voiceprint identifier.

Example

The VoiceXML snippet below is an example of how to use the verify request. Note that this snippet could be used for both enrollment and score verification. The mode parameter specified by the preceding session start (not shown here) dictates the behavior.

```
<block name=”SIV_VERIFY”>

<!-- assuming variable ‘claim’ containing voiceprint ID has already -->
<!-- been collected by the application                              -->
<var name=”voiceprint” expr=”application.claim”/>

<!-- assuming session has already been started for score            -->
<!-- also assuming that utterance has already been recorded and     -->
<!-- the audio is available through the variable ‘utterance’        -->

  <subdialog name="sivscore”
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-
name>:<port-number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/verify’”
             method="post"
             enctype=”multipart/form-data”
             namelist=”application.utterance”>
  <filled>
     <log>
         SIV_VERIFY::voiceprint <value expr=”sivscore.result.id”/>
         SIV_VERFIY::score <value expr=”sivscore.result.score”/>
         SIV_VERIFY::decision <value expr=”sivscore.result.decision”/>
     </log>
     <if cond=”sivscore.result.decision == ‘accepted’”>
        <log>
         SIV_VERIFY::access granted.
        </log>
     <elseif cond=”sivscore.result.decision == `undecided’”/>
        <log>
         SIV_VERIFY::Need more audio to make a decision.
        </log>
     <elseif cond=”sivscore.result.decision == ‘rejected’”/>
        <log>
         SIV_VERIFY::access denied.
        </log>
     </if>
  </filled>

  <catch event="error.siv.failure">
    <log>
        SIV_VERIFY:: ev=<value expr="_event"/>
        msg=<value expr="_message"/>
    </log>
  </catch>
  </subdialog>

</block>
```

Sample VoiceXML applications

The following directories contain sample applications that can be used for further reference:

- Linux *WVS_ROOT/siv/samples/vxml*
- Windows *WVS_ROOT*\siv\samples\vxml

In the application root document (*sivapp.vxml*) you can find more information about the available samples, including usage of both session-unbound (*sivscore.vxml*) and session-bound (*sivscore1stutt.vxml*) modes of the speaker verification Web API.

These *VoiceXML* documents are also deployed during product installation and should also be available through the same URI used to access the API:

http://<hostname>[:port]/ibmsiv/samples

Tips for speaker verification applications

The next sections provide some tips for developing speaker verification applications.

Define 'blanket' event catchers when running in session-bound mode

When using session-bound mode it is good practice to define global catch events to avoid exiting applications without properly ending a verification session. Global event catchers are normally defined in the root document of a VoiceXML application.

The example below illustrates the usage of global event catchers to end a verification session. Since calling an end session without previously establishing a verification session has no effect on the server side, it is OK to define the constructs below at the application scope.

```
<!-- *************************************************************** -->
<!-- These should be defined in root document of the VoiceXML app    -->
<!-- *************************************************************** -->

<catch event="error.siv.failure">
  <log>
       siv error - Ev=<value expr="_event"/>
       Msg=<value expr="_message"/>
  </log>
  <data <!--  var name="sivbaseurl" expr="'http://<wvs-host-name>:<port-
number>'" /-->
        <var name="sivbaseurl" expr="''" />
       srcexpr="sivbaseurl+'/ibmsiv/endsession'" />
  <exit />
</catch>

<catch event="connection.disconnect.hangup">
   <log>
       Call hangup - Ev=<value expr="_event"/>
       Msg=<value expr="_message"/>
   <log>
   <data <!--  var name="sivbaseurl" expr="'http://<wvs-host-name>:<port-
number>'" /-->
        <var name="sivbaseurl" expr="''" />
       srcexpr="sivbaseurl+'/ibmsiv/endsession'"/>
</catch>

<catch>
  <log>
      Generic - Ev=<value expr="_event"/>
      Msg=<value expr="_message"/>
  </log>
  <data <!--  var name="sivbaseurl" expr="'http://<wvs-host-name>:<port-
number>'" /-->
        <var name="sivbaseurl" expr="''" />
       srcexpr="sivbaseurl+'/ibmsiv/endsession'" />
  <exit />
</catch>
```

Use session bound mode to concatenate audio streams

You can use the session bound mode of the Web API to ‘concatenate’ different audio streams from multiple utterances into a single verification procedure. This allows applications more flexibility when balancing the speaker verification requirements in terms of audio duration, with the usability requirements for each speech application.

The example below shows the usage of session bound mode to handle enrollment across multiple turns.

```
<block name=”SIV_ENROLL”>

  <!-- assuming variable ‘claim’ containing voiceprint ID has already -->
  <!-- been collected by the application                              -->
  <var name=”voiceprint” expr=”application.claim”/>
  <var name=”mode” expr=”’enroll’”/>

  <!-- ************************************************************** -->
  <!-- Establishing a verification session first.                     -->
  <!-- ************************************************************** -->
  <subdialog name="startsession"
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-name>:<port-
number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/startsession'
             method=”get”
             namelist=”voiceprint mode">
  <filled>
      <log>
         SIV_ENROLL::enrollment session established for <value expr="voiceprint"/>
      </log>
  </filled>
  </subdialog>

  <!-- ************************************************************** -->
  <!-- Record the first utterance                                     -->
  <!-- ************************************************************** -->
  <record name="clip1" beep="true" maxtime="300s"
              dtmfterm="true" type="audio/basic">
    <prompt>Please, say your enrollment phrase now.</prompt>
    <filled>
      <prompt>Please, wait while we process your request.</prompt>
    </filled>
  </record>

  <!-- ************************************************************** -->
  <!-- Send first audio clip for enrollment.                          -->
  <!-- ************************************************************** -->
  <subdialog name=”utterance1”
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-name>:<port-
number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/verify’”
             method="post"
             enctype=”multipart/form-data”
             namelist=”clip1”>
  <filled>
     <log>SIV_ENROLL::utterance 1:
          decision <value expr=”utterance1.result.decision”/>
     </log>
     <if cond=”utterance1.result.decision == ‘accepted’”>
        <log>SIV_ENROLL::enrollment accepted.</log>
        <prompt>Congratulations! You have been enrolled</prompt>
        <data <!--  var name="sivbaseurl" expr="'http://<wvs-host-name>:<port-
number>'" /-->
            <var name="sivbaseurl" expr="''" />
            srcexpr="sivbaseurl+'/ibmsiv/endsession'"/>
        <exit/>
     <elseif cond="utterance1.result.decision == `undecided'"/>
        <log>
           SIV_ENROLL::Need more audio to make a decision.
           Proceeding to second utterance.
        </log>
     </if>
  </filled>
  </subdialog>

  <!-- ************************************************************ -->
  <!-- Record the second utterance                                  -->
  <!-- ************************************************************ -->
  <record name="clip2" beep="true" maxtime="300s"
              dtmfterm="true" type="audio/basic">
    <prompt>Please, speak your second enrollment phrase.</prompt>
    <filled>
      <prompt>Please, wait while we process your request.</prompt>
    </filled>
  </record>

  <!-- ************************************************************ -->
  <!-- Send second audio clip for enrollment.                       -->
  <!-- ************************************************************ -->
  <subdialog name="utterance2"
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-name>:<port-
number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/verify'"
             method="post"
             enctype="multipart/form-data"
             namelist="clip2">
  <filled>
     <log>SIV_ENROLL::utterance 2:
          decision <value expr="utterance2.result.decision"/>
     </log>
     <if cond="utterance1.result.decision == 'accepted'">
        <log>SIV_ENROLL::enrollment accepted.</log>
        <prompt>Congratulations! You have been enrolled</prompt>
        <exit/>
     <elseif cond="utterance1.result.decision == `undecided'"/>
        <log>SIV_ENROLL::still not enough audio.</log>
        <prompt>I'm sorry, please try again later.</prompt>
     </if>
  </filled>
  </subdialog>

  <!-- ************************************************************ -->
  <!-- Ending the session.                                          -->
  <!-- ************************************************************ -->
  <subdialog name="endsession"
             <!--  var name="sivbaseurl" expr="'http://<wvs-host-name>:<port-
number>'" /-->
             <var name="sivbaseurl" expr="''" />
             srcexpr="sivbaseurl+'/ibmsiv/endsession'
             method="get">
  <filled>
      <log>
         SIV_ENROLL::enrollment session ended"/>
      </log>
  </filled>
  </subdialog>

</block>
```

Always check for ‘undecided’ decision when performing enrollment

As stated before, the speaker verification engine has an internal threshold controlling the amount of audio processed during enrollment. This threshold must be reached for the engine to declare the voiceprint ‘*accepted*’. Therefore, it is good practice for the application to check for an ‘*undecided*’ decision (i.e., more audio needed) before considering that the enrollment has been completed. If the speaker verification engine does not report an accepted decision, the voiceprint will not be enrolled (created).

End verification session after enrollment is complete

When enrolling using the session bound mode, keep in mind that the voiceprint is only created when the session terminates. Therefore, it is good practice for your application to check for the decision and end the session as soon as it gets an ‘accepted’ result from the server.

Voiceprint repository API

The speaker verification component relies on a persistent repository to keep all enrolled voiceprints. The location of the voiceprint repository is abstracted by the Voiceprint Repository (VPR) API, which makes the speaker verification component agnostic to the backend used to store the voiceprints. **NOTE:** The VPR API is used internally by the speaker verification component and it is not directly exposed to the application developer.

The repository location is specified by the repository-uri parameter (either through configuration or as one of the speaker verification Web API parameters). Through the Voiceprint Repository abstraction, customers can provide centralized access to their own voiceprint repository by providing an implementation of the HTTP interface defined below for the Voiceprint Repository abstraction.

HTTP GET

The speaker verification component sends an HTTP GET request to the VPR Servlet to perform one of the operations specified below.

Parameters

voiceprint (required)

The voiceprint associated with the request.

op (required)

The type of operation to perform.

Expected values:

query: detects whether or not the specified voiceprint is available
delete: deletes the specified voiceprint from the repository
fetch: retrieves the specified voiceprint from the repository

Response

The speaker verification component relies on the HTTP response return code to indicate whether the operation completed successfully. Note that the return codes may assume slightly different meanings, depending on the type of operation being performed. The following return codes are expected:
200 – The voiceprint exists (or has been successfully deleted by a delete operation)
400 – The request is missing a required parameter
404 – The voiceprint does not exist (or could not be deleted by a delete operation)
Any other status code is handled as a generic error.

The HTTP response body must contain the voiceprint binary data (from a successful fetch operation).

HTTP POST

The SV Resource Adapter sends an HTTP POST request to the VPR Servlet when it needs to save a voiceprint in the repository. The body of the request contains the binary data for the specified voiceprint. If the specified voiceprint already exists, the contents are overwritten.

Parameters

voiceprint (required)

The voiceprint associated with the request.

Response

The SV Resource Adapter relies on the HTTP response return code to indicate whether the operation completed successfully. The following return codes are expected:

200 – The voiceprint was saved successfully
400 – The request is missing a required parameter
Any other status code is handled as a generic error.

NOTE: The implementation of the Voiceprint Repository Servlet and the instrumentation to access the underlying database are not provided by this product. A reference implementation is provided as an example and can be found in the installation directory, under

- Linux *WVS_ROOT/siv/samples/vpr/sivvpr.war*
- Windows *WVS_ROOT*\siv\samples\vpr\sivvpr.war

Chapter 4. Planning

This section explains the considerations when you plan to add speaker verification to your existing WebSphere® Voice Server system. The additional requirements for speaker verification vary, depending on the following issues:

- The number of telephone lines or channels supported that will use speaker verification.
- When to verify the speaker.
- The size of your voice print repository.
- The length of the audio from a user to verify.

Planning your WebSphere Voice Server system involves the following steps:

- Understanding the speech application using speaker verification.
- Estimating speaker verification CPU requirements.

Understanding the speech application using speaker verification

When planning a voice system using WebSphere® Voice Server speaker verification, you must determine:

- The number of telephony lines requiring speaker verification support.
- When will you perform speaker verification in the application? Only at the beginning of the call to verify the caller's identify or continually during the call?
- How many seconds of audio will be collected and used to verify the speaker?
- How many active telephony lines will perform speaker verification concurrently?

NOTE: This document can only provide approximate guidelines regarding the exact size and number of machines you need for your WebSphere Voice Server system. It is essential that you test any implementation with realistic call volumes before you put it into production. For guidance about capacity planning for your specific configuration, contact your IBM® representative.

In terms of persistent storage, you must determine:

- What kind of voiceprint repository to use? Understand the requirements, so you can plan and develop your own implementation of the VPR Servlet to access your persistent storage backend.
- How many voiceprints are expected to be in the repository? What size space is needed to store all the voiceprints (each voiceprint is approximately 43K bytes)?

Application load

To estimate the application load on the system, you need to know the following:

- The number of telephony channels required to handle the number of expected calls for applications using WebSphere Voice Server speaker verification.

Your WebSphere Voice Server system should be able to handle the maximum demand for speech resources. The maximum demand is the resources needed at the peak calling hour, rather than a day's average number of hourly calls. The primary speech resource is the speaker verification engine. The demand for engines is influenced both by the frequency of calls and how they are distributed. If all the incoming calls use the same application and start at the same time, each call will need an engine at the same time so the demand will be high. If, on the other hand, calls are distributed normally, the number of engines needed simultaneously may be considerably smaller.

For your applications, you must determine the acceptable performance or desirability of an engine being available for a call without a significant delay. Delays can cause performance degradation, such as not recognizing speech input or stuttering output. If degradation of performance is acceptable during peak utilization, fewer engines will be required.

The following variables affect application load:

- The time that engines are allocated or assigned as a proportion of a call. This is known as the *allocation duty cycle*. The allocation model for speaker verification engine depends on the API usage. If session unbound mode is used, the engines are allocated dynamically to perform a single verification operation and immediately freed. If session mode is used, an engine is allocated for the lifecycle of the session.
- The time that engines are active (in processing the verification audio) as a proportion of a call. This is known as the *active duty cycle*. This can vary considerably, depending on the design of your voice applications and their complexity. These factors determine the number of concurrent speaker verification sessions that are required.

The number of concurrent speaker verification sessions, in turn, determines the number of processors required and how powerful they must be. Similarly, these two variables—the number and speed of the processors—dictate the number and size of the machines needed for your WebSphere Voice Server installation.

For example, an application using speaker verification at the start of the call with five seconds worth of audio is likely to be actively engaged for only a short proportion of the length of a call. It will have a short active speaker verification duty cycle and require less CPU.

An application continually verifying the user with longer amounts of audio is likely to spend more time actively engaged. In this case, the active duty cycle for speaker verification is longer and more CPU processing is required.

Application design

The amount of CPU processing required for speaker verification is highly dependent upon the amount of audio to process for each verification request and the number of other concurrent verification requests in progress.
If you perform batch processing of speaker verification requests, then you can manage the amount of CPU processing available by reducing the number of concurrent requests issued. In this situation, longer audio samples can be used.
If you have a user waiting for the speaker verification request to complete, then use small audio samples and perform verification only at the beginning of the call.

System resource

Once you know the maximum number of concurrent sessions required for speaker verification and the average amount of audio, you can determine how many WebSphere Voice Server machines are necessary. You can also determine the minimum specifications for the machines, which are also dependent on the operating system you select.
The actual number of concurrent speaker verification requests that will run on each machine is solution-dependent. A solution must be tested to verify that a system can handle a condition where all the speaker verification, ASR, and TTS engines are fully utilized.

Estimating resources for speaker verification

This section describes the resources required for speaker verification.
The critical resource is CPU. CPU is consumed when processing a speaker verification request. The amount of the audio determines how long CPU is consumed. Memory is consumed when WVS started with speaker verification installed and is minimal.
CPU utilization is determined by:

- The active duty cycle of the started speaker verification engines
- The length of the audio submitted for verification

Memory utilization is determined by:

- The number of active (allocated) engines at a given time
- The number of speaker models in the CCSM directory. By default, the speaker verification engine is installed with 396 models, which are loaded and remain in memory during system startup

NOTE: There are limits to the desirable CPU load. If the CPU utilization is too high, application response time can increase.

Chapter 5. Installing

Refer to the Getting Started Guide document for detailed instructions on how to install and uninstall the WebSphere® Voice Server component for speaker verification.

Chapter 6. Configuring

The table below lists the configuration properties used by this speaker verification component. These parameters can be changed using the methods described in the WebSphere® Voice Server information center and in Chapter 6. Configuring.

Name	Technical name and description	Default value	Valid values
Acceptance Threshold	**com.ibm.voice.server.verifier.adapter.engine.acceptThreshold** The score threshold used to determine whether a score is 'accepted' or 'rejected'. The value of this property is determined in the tuning process.	0.0	[-1,1]
Audio Codec	**com.ibm.voice.server.verifier.adapter.engine.codec** The audio codec used to process incoming audio.	G711ulaw	G711ulaw or G711alaw
Repository URI	**com.ibm.voice.server.siv.connector.repository-uri** Specifies the default voiceprint repository location to use if none is specified by the application.	File://${WVS_ROOT}/ siv/voiceprints	Any valid file or http URI

Configuring speaker verification Resource Adapter connector

You can configure the pool of connections to the speaker verification Resource Adapter through the WebSphere Application Server administration console. Each Resource Adapter connection represents an actual connection to the speaker verification engine. By default, the product is installed with the following connection pool settings.

Table 4: Connection pool settings

Name	Description	Default value
Min Connections	The minimum number of connections to the speaker verification engine	5
Max Connections	The maximum number of connections to the speaker verification engine	50

More information about how to configure the connection pool for a resource adapter can be found in WebSphere Application® Server Infocenter, in the **Configuring→Resources→Data Access-→Configuring Java 2 Connector connection factories→Connection pool settings** section.

NOTE: Do not change any other connection pool setting without direct instructions from IBM support.

Chapter 7. Administering

The administration of servers involves operational and monitoring functionality listed in the WebSphere Voice Server information center ‘Administering’ topic.

Chapter 8. Tuning

It is important to note that the WebSphere® Voice Server component for speaker verification is not intended to be used straight out of the box. For the speaker verification engine to perform properly, it needs to be tuned to accommodate the specific characteristics of the target deployment environment.

Tuning must be performed before application/solution deployment in the same (or similar) conditions of the target environment. Since this speaker verification component does not include features to help with the procedure, contact your IBM representative before proceeding with tuning. In this section, we give an outline of how the process looks.

Before you can start tuning, you need to understand how the application is expected to use the speaker verification technology. Questions you should be asking include: How long are the audio clips used for enrollment? How long are the audio clips used for scoring? What kind of audio clips are used for scoring (only digits, date of birth, etc.)? What is the audio codec to be used: u-law or a-law?

Once you have the information, you can proceed with tuning, which should include the following steps:

1. Collect multiple audio samples from multiple speakers.
2. Enroll speakers (off-line).
3. Augment speaker models.
4. Collect the scores for all audio clips (off-line).
5. Determine acceptance threshold for the solution.

Collect audio samples from multiple speakers on multiple devices

Collect audio samples from multiple speakers, making sure the samples resemble the ones the application will use. It is also important to make sure the input devices (landline, cell phones, VoIP, etc.) used to collect these samples are similar to those used by the application. Use as many speakers as possible in this phase and collect as many audio samples as possible from each one. As a rule of thumb, we recommend 50 speakers, with at least one audio clip for enrollment and five others for scoring. The outcome of this step should be a collection of audio clips (or files). Each speaker should be assigned a unique identification, which should be used to identify the audio clips for each speaker (for instance, you could save all the audio files from a given speaker in a separate directory with the speaker identification name).

© Copyright IBM Corp. 2007,, 2008

Enroll speakers

Once the audio samples are collected and saved to a safe location, enroll each speaker by submitting his or her enrollment audio clip to the speaker verification engine.
This step is normally performed off-line, through the Web connector API (for example, using Perl scripts). The outcome of this step is a number of newly created voiceprints, one for each speaker. We recommend the usage of the default file repository to facilitate the procedure (you should see the new voiceprint files in the default repository directory). Also, consider saving these voiceprints in a separate location after enrollment is complete.

Augment speaker models

The speaker verification engine relies on statistical speaker models to operate. These models are generated off line and deployed during product installation. During system startup, the speaker verification component loads these models into memory.

The idea behind augmenting the models used for speaker verification is to provide the engine with more information about the deployment environment. This can be achieved by simply adding (copying) the voiceprints created in the previous step to the location for loading the default models. These are the steps:

1. Stop the WAS.
2. Make a backup copy of the relevant directory:
   Linux *WVS_ROOT*/siv/engine/ccsm
   Windows *WVS_ROOT*\siv\engine\ccsm
3. Copy all new voiceprints to the directory.
4. Restart the WAS.

Collect scores for all audio clips

Collect scores for all audio clips using the same mechanism used to enroll each speaker (e.g., Perl scripts). This time, however, each audio clip should be submitted for scoring, first against its own voiceprint (target score), and then against each of the remaining voiceprints (non-target scores) created during the enrollment step. **IMPORTANT**: Do not use any of the enrollment audio clips during this phase.

The outcome of this step should be two sets of scores. The first one contains all the target scores; the second contains all the non-target scores.

Determine acceptance threshold

A speaker verification tuner application is available for download from the WebSphere® Voice Server support site. The tuner analyzes the provided target and non-target scores, determining the acceptance threshold in terms of False Acceptance (FA) and False Rejection (FR) rates. Note that the speaker verification tuner is not part of this component. The tool can also calculate the Equal Error Rate (EER), defined as the point of operation where FA and FR are the same.

The outcome of this step is the target acceptance threshold for the application. Use this value to configure the Acceptance Threshold property (see Configuring for more details).

Chapter 9. Troubleshooting

Most of the tasks involved in troubleshooting speaker verification are common to the WebSphere® Voice Server product. Refer to the troubleshooting section of the WebSphere® Voice Server Infocenter for further details on how to diagnose problems.

Monitoring log messages

Similarly to all WebSphere® Voice Server components, speaker verification log messages are written to the WebSphere Application Server log file, located in the relevant directory:

- Linux *WAS_ROOT*/profiles/<profile>/logs/*server1*/SystemOut.log
- Windows *WAS_ROOT**profiles*\<*profile*>\logs\*server1*\SystemOut.log

All log messages used by the speaker verification components follow the conventions used by other WebSphere® Voice Server components, adding the following message prefix:
CWVSVXXXXS

Where

CWV:	IBM-registered three-character prefix identifying WebSphere Voice Server.
***SV*:**	Identifies speaker verification components.
***NNNN*:**	Numeric message ID (4-digits).
***T*:**	Severity type (**I**nformation, **W**arning, or **E**rror).

Speaker verification resource adapter - CWVSV0000X-CWVSV0099X

CWVSV0001I: Starting Verifier...
Explanation: Informational message indicating that the Verifier has started.
User Response: None.

CWVSV0010E: Invalid state transition. Not allowed to switch from state {0} to {1}.
Explanation: SIV subsystem cannot handle the received request in its current state.
User Response: None.

CWVSV0011E: Error creating adapter connection. Reason: {0}:{1}.
Explanation: The SIV subsystem failed to create a new adapter connection.
User Response: Check the reported reason for more details.

CWVSV0012E: Error detected while processing audio.
Explanation: The SIV subsystem found an I/O error while reading the WAV input [is this right?] file.
User Response: Enable the trace for more details.

CWVSV0020E: Cannot start verification for {0}. Failed to open input wave file: {0}. Reason: {1}
Explanation: The SIV subsystem failed to read the WAV input file.
User Response: Make sure the input directory exists and has read/write permissions. Make sure the file system is not full. Enable the trace for more details.

CWVSV0021E: Cannot start verification for {0}. Failed to fetch voiceprint. Reason: {1}
Explanation: The SIV subsystem failed to fetch the specified voiceprint from the repository.
User Response: Make sure the voiceprint repository is accessible. Enable the trace for more details.

CWVSV0022E: Cannot start verification for {0}. The specified voiceprint does not exist.
Explanation: The voiceprint could not be located in the voiceprint repository.
User Response: Make sure the specified voiceprint was previously enrolled.

CWVSV0023E: Cannot start verification for {0} due to internal error. Reason: {1}
Explanation: None.
User Response: Check the reported reason for more details.

(WAS error messages)

J2CA0020E: The Connection Pool Manager could not allocate a Managed Connection: com.ibm.websphere.ce.j2c.ConnectionWaitTimeoutException: Connection not available.
Explanation: A request to the speaker verification resource adapter failed because the maximum number of connections was exceeded.

User Response: Change the resource adapter configuration to increase the number of maximum connections.

Speaker verification engine: CWVSV0100X-CWVSV0199X

CWVSV0101E: Invalid value for configuration property {0}.
Explanation: The specified configuration has an invalid value.
User Response: Change the property value.

Speaker verification web connector: CWVSV1000X-CWVSV1199X

CWVSV1001I: WVS SIV Connector started. Build date: {0}
Explanation: Informational message indicating that the SIV Connector has started.
User Response: None.

CWVSV1002E: Configuration problem detected while starting up SIV Connector.
Explanation: The SIV Connector found a missing or invalid configuration parameter.
User Response: Check the logs for more details. Fix the specified configuration setting and restart the SIV Connector.

CWVSV1011E: Exception caught while decoding HTTP message boundary.
Explanation: The message boundary received by the HTTP connector appears to be invalid.
User Response: Check the logs for more details.

CWVSV1012E: IO Exception caught while building HTTP response.
Explanation: The SIV Connector found an exception while building an HTTP response.
User Response: Check the logs for more details.

CWVSV1013E: Exception caught while handling HTTP request targeting {0}, Reason: {1}
Explanation: The SIV Connector reported an error when handling a request to the specified target.
User Response: Check the reported reason and the logs for more details.

CWVSV1020I: SIV installation verification test has completed successfully.
Explanation: Informational message indicating that the SIV IVT servlet was invoked and ran successfully.
User Response: None.

CWVSV1021E: SIV installation verification test has failed.
Explanation: Message indicating that the SIV IVT servlet was invoked but failed to complete.
User Response: Check logs for more details.

CWVSV1100W: Speaker verification session has expired. Session id={0}
Explanation: An application established a verification session but did not send any requests for a certain period of time. The session timed out and was automatically cleaned up.
User Response: The application should either send more requests or end the verification session within the timeout period.

Enabling and monitoring trace for speaker verification

Use the WebSphere® Application Server administrative console to enable traces. The trace files are located in one of the following paths, based on your operating system:

- Linux *WAS_ROOT*/profiles/<profile>/logs/*server1*/trace.log
- Windows *WAS_ROOT*\profiles\<profile>\logs\server1\trace.log

The trace components in the following table are available for speaker verification.

Table 5 - SIV components

Acronym	**Trace Group** **(WAS Admin Console)**	**Component**
VERIFIER	IBM.WVS.SIV	Verifier component
SIVRA	IBM.WVS.SIV	SV Resource Adapter
SIVENG	IBM.WVS.SIV	SIV Engine
SIVHTTPCONNECTOR	IBM.WVS.SIV	SIV Web Module
SIVMRCP	IBM.WVS.SIV	RTSP/MRCP messages

Collecting information for IBM support

The WebSphere® Voice Server collector includes support for speaker verification. Refer to the WebSphere® Voice Server product documentation for information on how to run the WVS collector.

Notices

This information was developed for products and services offered in the United States.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM
Corporation North Castle Drive
Armonk, NY 10504-1785
USA

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department T01B
3039 Cornwallis Road
Research Triangle Park, NC 27709-2195
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

The products offered by IBM in this publication are platforms or middleware that can provide the capability for you to add, design, or create applications to take advantage of the products. Notwithstanding the terms of any other agreements You have with International Business Machines Corporation, or any of its related or affiliated companies, IBM shall not be liable to You for any and all claims of patent infringement, including inducement or contributory infringement, or any claims for indemnification for such claims brought against the products or based on the combination, use or operation of the products with software or hardware. You also should be aware that it may be necessary for You to obtain a patent license from one or more third parties, including, but not limited to, Ronald A. Katz or Ronald A. Katz Technology Licensing, L.P. (commonly referred to as RAKTL), before using certain applications designed and built to run on the IBM products listed in this publication.

Copyright license

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

AIX®, IBM and WebSphere are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Microsoft® and Windows® are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Dialogic and Intel® are registered trademarks of Intel Corporation in the United States, other countries, or both.

Cisco is a registered trademark of Cisco Systems, Inc., in the United States, other countries, or both.

Sun, Java and Java-based marks are registered trademarks of Sun Microsystems, Inc., in the United States, other countries, or both.

For more information on CallPath products please contact Genesys Telecommunications Laboratories, Inc. On the World Wide Web, go to the CallPath Framework part of the Genesys Web site (http://www.genesyslabs.com).

Other company, product and service names may be trademarks or service marks of others.

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. These are the major accessibility features in WebSphere Voice Server:

- You can use screen-reader software and a digital speech synthesizer to hear the HTML version of this guide.
- You can operate many features using the keyboard instead of the mouse.