Avaya Media Processing Server Nuance VocalPassword 8.x Reference Guide

Release 4.1
Issue 1
August 2013

Notice

While reasonable efforts have been made to ensure that the information in this document is complete and accurate at the time of printing, Avaya assumes no liability for any errors. Avaya reserves the right to make changes and corrections to the information in this document without the obligation to notify any person or organization of such changes.

Documentation disclaimer

“Documentation” means information published by Avaya in varying mediums which may include product information, operating instructions and performance specifications that Avaya generally makes available to users of its products. Documentation does not include marketing materials. Avaya shall not be responsible for any modifications, additions, or deletions to the original published version of documentation unless such modifications, additions, or deletions were performed by Avaya. End User agrees to indemnify and hold harmless Avaya, Avaya's agents, servants and employees against all claims, lawsuits, demands and judgments arising out of, or in connection with, subsequent modifications, additions or deletions to this documentation, to the extent made by End User.

Link disclaimer

Avaya is not responsible for the contents or reliability of any linked websites referenced within this site or documentation provided by Avaya. Avaya is not responsible for the accuracy of any information, statement or content provided on these sites and does not necessarily endorse the products, services, or information described or offered within them. Avaya does not guarantee that these links will work all the time and has no control over the availability of the linked pages.

Warranty

Avaya provides a limited warranty on its hardware and Software (“Product(s)”). Refer to your sales agreement to establish the terms of the limited warranty. In addition, Avaya’s standard warranty language, as well as information regarding support for this Product while under warranty is available to Avaya customers and other parties through the Avaya Support website: http://support.avaya.com. Please note that if you acquired the Product(s) from an authorized Avaya Channel Partner outside of the United States and Canada, the warranty is provided to you by said Avaya Channel Partner and not by Avaya. “Software” means computer programs in object code, provided by Avaya or an Avaya Channel Partner, whether as stand-alone products or pre-installed on hardware products, and any upgrades, updates, bug fixes, or modified versions.

Licenses

THE SOFTWARE LICENSE TERMS AVAILABLE ON THE AVAYA WEBSITE, HTTP://SUPPORT.AVAYA.COM/LICENSEINFO ARE APPLICABLE TO ANYONE WHO DOWNLOADS, USES AND/OR INSTALLS AVAYA SOFTWARE, PURCHASED FROM AVAYA INC., ANY AVAYA AFFILIATE, OR AN AUTHORIZED AVAYA CHANNEL PARTNER (AS APPLICABLE) UNDER A COMMERCIAL AGREEMENT WITH AVAYA OR AN AUTHORIZED AVAYA CHANNEL PARTNER. UNLESS OTHERWISE AGREED TO BY AVAYA IN WRITING, AVAYA DOES NOT EXTEND THIS LICENSE IF THE SOFTWARE WAS OBTAINED FROM ANYONE OTHER THAN AVAYA, AN AVAYA AFFILIATE OR AN AVAYA AUTHORIZED AVAYA CHANNEL PARTNER; AVAYA RESERVES THE RIGHT TO TAKE LEGAL ACTION AGAINST YOU AND ANYONE ELSE USING OR SELLING THE SOFTWARE WITHOUT A LICENSE. BY INSTALLING, DOWNLOADING OR USING THE SOFTWARE, OR AUTHORIZING OTHERS TO DO SO, YOU, ON BEHALF OF YOURSELF AND THE ENTITY FOR WHOM YOU ARE INSTALLING, DOWNLOADING OR USING THE SOFTWARE (HEREINAFTER REFERRED TO INTERCHANGEABLY AS “YOU” AND “END USER”), AGREE TO THESE TERMS AND CONDITIONS AND CREATE A BINDING CONTRACT BETWEEN YOU AND AVAYA INC. OR THE APPLICABLE AVAYA AFFILIATE (“AVAYA”).

Copyright

Except where expressly stated otherwise, no use should be made of materials on this site, the Documentation, Software, or hardware provided by Avaya. All content on this site, the documentation and the Product provided by Avaya including the selection, arrangement and design of the content is owned either by Avaya or its licensors and is protected by copyright and other intellectual property laws including the sui generis rights relating to the protection of databases. You may not modify, copy, reproduce, republish, upload, post, transmit or distribute in any way any content, in whole or in part, including any code and software unless expressly authorized by Avaya. Unauthorized reproduction, transmission, dissemination, storage, and or use without the express written consent of Avaya can be a criminal, as well as a civil offense under the applicable law.

Third Party Components

“Third Party Components” mean certain software programs or portions thereof included in the Software that may contain software (including open source software) distributed under third party agreements (“Third Party Components”), which contain terms regarding the rights to use certain portions of the Software (“Third Party Terms”). Information regarding distributed Linux OS source code (for those Products that have distributed Linux OS source code) and identifying the copyright holders of the Third Party Components and the Third Party Terms that apply is available in the Documentation or on Avaya’s website at: http://support.avaya.com/Copyright. You agree to the Third Party Terms for any such Third Party Components.

Trademarks

The trademarks, logos and service marks (“Marks”) displayed in this site, the Documentation and Product(s) provided by Avaya are the registered or unregistered Marks of Avaya, its affiliates, or other third parties. Users are not permitted to use such Marks without prior written consent from Avaya or such third party which may own the Mark. Nothing contained in this site, the Documentation and Product(s) should be construed as granting, by implication, estoppel, or otherwise, any license or right in and to the Marks without the express written permission of Avaya or the applicable third party.

Avaya is a registered trademark of Avaya Inc.

All non-Avaya trademarks are the property of their respective owners. Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

Downloading Documentation

For the most current versions of Documentation, see the Avaya Support website: http://support.avaya.com.

Contact Avaya Support

See the Avaya Support website: http://support.avaya.com for product notices and articles, or to report a problem with your Avaya product. For a list of support telephone numbers and contact addresses, go to the Avaya Support website: http://support.avaya.com, scroll to the bottom of the page, and select Contact Avaya Support.

Contents

Chapter 1: Introduction 5
- Introduction 5
- Avaya support for VocalPassword 6
- Vendor reference documentation 6
- System architecture view 6
- Hardware and software requirements 7
- Sample application functional description 9
- Nuance VocalPassword API methods used by sample applications 11
- Verification types 13

Chapter 2: Installation and Configuration 15
- Installation and configuration overview 15
- Install and configure VocalPassword Server Node 15
- Overview 15
- Prerequisite to using VocalPassword 8.x 16
- Install and configure Database Node 16
- Installation roadmap for VocalPassword Node 17
- Install and configure MPS Application Processor Node 24
- Installation and configuration required for VocalPassword 8.x 24
- VocalPassword advanced configuration 28
  - Configuring Liveness Detection with Dictionary 29
  - Configuring for Text-Independent Verification 30
  - Configuring for Text-Prompted Verification 30
  - Configuring to catch inconclusive verification results 32
  - Configuring Playback Detection 32
  - Configuring the number of utterances required for enrollment 33
  - Configuring Unsupervised adaptation 34
  - Creating Background Models with the Calibration Wizard 36
  - Configuring Fraudster Detection 39

Chapter 3: MPS Developer Application Development 41
- Nuance VocalPassword MPS Developer Application programming 41
- Single user life cycle 42
- Request-response programming pattern 43
- Main module 44
- Begin call 46
- Read configuration parameters 47
- Get account number 50
- Caller chooses action 51
- StartSession 52
- Which action 53
- Enrollment 54
- Authenticate or verify a user 59
- Verify against members of a group 61
- Unenrollment 64
- Liveness verification 65

End the call ... 68
HTMLS resource operations ... 69
Delete user ... 72
Add enrollment utterance ... 73
Verify user ... 75
Query user ... 76
Clear user ... 76
Verify user from a group ... 76
Start session ... 77
End session ... 77
Create Adhoc group ... 77
Process error codes for HTMLS message ... 78
Free the HTMLS resource ... 79
Delete file ... 80
GSpeechKeyHost folder ... 81
Chapter 4: VoiceXML Application Development ... 83
Nuance VocalPassword VoiceXML application programming ... 83
Sample application functional description ... 83
Debugging tips for VoiceXML applications ... 84
VoiceXML application flow charts ... 85
VP-Main flow chart ... 86
Enrollment flow chart ... 86
Single User Verification flow chart ... 87
Group Verification flow chart ... 88
Obtain User Properties flow chart ... 89
Delete User flow chart ... 89
Liveness Verification flow chart ... 89
Chapter 5: Alarms ... 91
Alarm overview ... 91
Alarm log files ... 92
Alarm format ... 92
Nuance VocalPassword 8.x-specific alarms ... 93
MPS Developer alarm generation ... 94
VoiceXML alarm generation ... 94
Chapter 6: Troubleshooting and Known Issues ... 97
Introduction ... 97
Troubleshooting ... 97
Index ... 99

Chapter 1: Introduction

This chapter covers:

1. Introduction
2. Avaya support for VocalPassword
3. Vendor reference documentation
4. System architecture view
5. Hardware and software requirements
6. Sample application functional description
7. Nuance APIs used by applications
8 Verification types
9 VocalPassword advanced configuration

Introduction

The Nuance VocalPassword allows an Interactive Voice Response (IVR) application to verify caller's credentials, before allowing the caller to proceed with the transaction. This feature adds value to IVR applications for developer to build a security measure into the application.

Nuance VocalPassword 8.x performs the functions of enrollment and verification.

- **Enrollment**: The enrollment function enrolls users by prompting them for a series of utterances from which it can derive a voice model. The voice model stores the corresponding user's details in the database.
- **Verification**: The verification function allows users to verify themselves by speaking an utterance. The utterance is compared against the enrolled voice model.

Irrespective of the vendor speech resource that the application uses, any application can use VocalPassword.

Avaya support for VocalPassword

Avaya supports the following VocalPassword 8.x features, vendor parameters, utterance formats, and sample rate.

- **Features**: Avaya supports VocalPassword 8.x features, which include enrollment, verification (dependent and independent), and identification. Verification is also known as authentication.
- **Vendor Parameters**: Avaya supports all vendor parameters and their defaults, except the following: the database server IP, user name, and password.
- **Utterance Formats**: Avaya supports only the audio or x-wav format.
- Avaya supports the sample rate of only 8000 Hz.

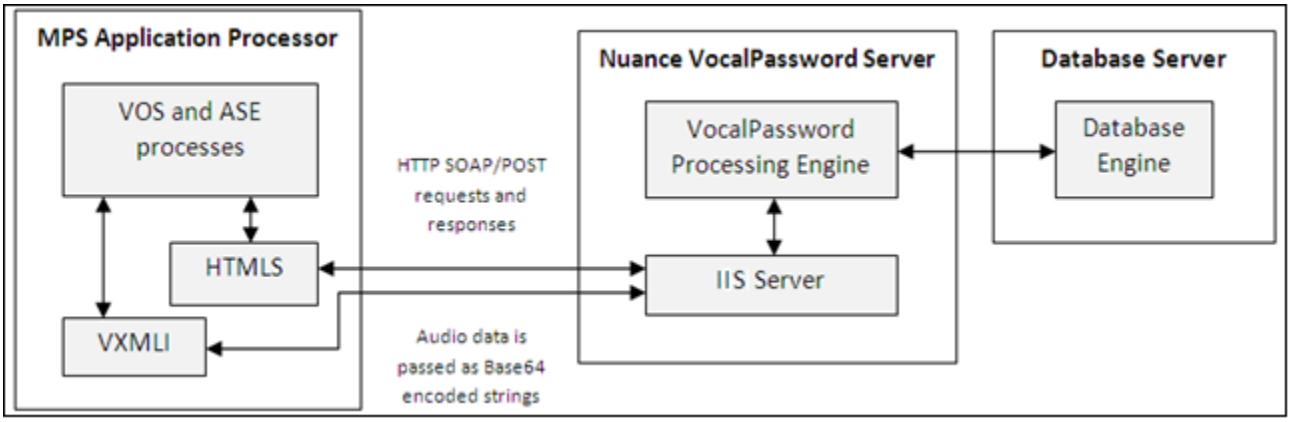
Vendor reference documentation

When working with VocalPassword, use the following documentation distributed by Nuance Communications, Inc.

- *VocalPassword 8.x Installation Guide*
- *VocalPassword Help Suite 8.x*

System architecture view

The following diagram shows a system view of Nuance VocalPassword within the MPS environment. MPS Manager resides on the MPS AP. It can also reside on another node. The VocalPassword server is not under control of the Avaya MPS service and resides on a separate node. The Database server is provided by the customer.



The following Web-service endpoints are available on the VocalPassword server:

The Native VocalPassword API:

```
http://<VocalPassword-Server-IP>/vocalpassword/
vocalpasswordserver.asmx
```

Hardware and software requirements

The following section identifies the hardware and software required for implementing the Nuance VocalPassword in the MPS environment.

MPS Application Processor Node

Hardware

The following guides describe all hardware for MPS platform. You must refer to these guides for all software that is valid in the MPS environment.

- *MPS 1000 Hardware Installation and Maintenance Manual*
- *MPS 500 Hardware Installation and Maintenance Manual*

Software

The environment, in which Nuance VocalPassword functions, requires the following software to be installed on the MPS Application Processor node. The following guides describe software for MPS platform. You must use the information in this guide when installing software for VocalPassword. See Install and configure MPS Application Processor Node on page 24.

- *Installing MPS Software on the Solaris Platform*
- *Installing MPS Software on the Windows Platform*

- Operating Systems
  - Solaris 10
  - Windows 2003 and 2008 R2
- Avaya_SelfService base software
- MPS Developer
- PERIxmlc
- PERIhtmls
- PeriPatch (if available)
- PERIvxml (required only for VoiceXML applications)

VocalPassword Server Node

The environment, in which Nuance VocalPassword functions, requires the following software to be installed on the Nuance VocalPassword Server node. Use the information in this guide to install and configure software on the VocalPassword node. See Install and configure VocalPassword Server Node on page 15 and Install and configure Database Node on page 16.

Hardware

The following are the minimum hardware requirements for Nuance VocalPassword server:

- Processor: 2 X Quad Core CPU
- Memory: 4 GB
- Storage: 20 GB

Software

The following are the minimum software requirements for Nuance VocalPassword node:

- Windows Server 2008 R2 x64 (Standard Edition)
- IIS with ASP and ASP.NET activated
- .Net Framework v4
- Nuance VocalPassword 8.3

While it is possible to install the database on the VocalPassword server machine, this configuration has a negative impact on performance as the load capacity on the VocalPassword Server is diminished. Therefore, Avaya supports only a configuration where the VocalPassword server and the database reside on different machines.

Database Server Node

Avaya supports Microsoft SQL Server 2008 Enterprise R2 and Oracle 11g for use with Nuance VocalPassword 8.x.

- If you have an existing Oracle 11g or Microsoft SQL Server 2008 R2 database server, you can create a new database on one of these servers to use for Nuance VocalPassword 78.x.
- If you do not have an existing database server, Avaya recommends using Microsoft SQL 2008 R2 due to ease of administration and supportability by Nuance Communications, Inc. For information about installing the MS SQL database, see Install and configure Database Node on page 16.

Avaya and Nuance do not provide guidance on how to install an Oracle 11g database server. It is assumed that if you are to use an Oracle database server, it has already been installed.

For assistance with creating an Oracle database, consult the Oracle 11g documentation at the address http://www.oracle.com.

For information about installing and configuring the VocalPassword and Database nodes, see Installation and configuration overview on page 15.

Hardware

The database node hardware is determined by the type of database you choose. Refer to the hardware requirements of the database vendor (Microsoft or Oracle) for these specifications.

Software

- Operating System: The type of operating system you use is determined by the type of database you choose. Refer to the operating system requirements of the database vendor (Microsoft or Oracle) for these specifications.
- Database software: Microsoft SQL Server 2008 R2 or Oracle 11g.

Sample application functional description

Both the MPS Developer and VoiceXML sample applications follow the same call flow. The application opens by greeting the caller and prompts for the caller's nine digit account number. The VocalPassword server does not require the account number to be nine digits, or even to be numeric. The request is simply a function of the sample application. The application then prompts the caller to select one of five actions, each of which demonstrate the base VocalPassword features of enrollment, single user verification, group verification, unenrollment, and liveness verification. A description of each feature follows. For information about the Nuance VocalPassword APIs used during these functions, see Nuance VocalPassword API methods used by sample applications on page 11.

Enrollment

The goal of the enrollment process is to obtain a voice model from the caller's speech and enroll the caller with the verification service. The application prompts the caller to speak the same phrase three times. In this case, the phrase is “one two three four five six seven eight nine”. The application may prompt the caller to repeat an utterance if it is too loud, too short, too soft, or otherwise unclear. If the caller speaks three acceptable utterances, the VocalPassword calculates a voice model and informs the caller that a voice model was created. The voice model is stored in the database.

The enrollment process uses the following Nuance VocalPassword API methods: `IsTrained`, `DeleteVoiceprint`, and `Enroll`.

Single user verification

The goal of the single user verification process is to verify the caller's identity by matching the caller's utterance against a previously created voice model existing in the database. The application prompts the caller to say the phrase that was used for enrollment just once. The VocalPassword matches the utterance against the voice model. Then, the application informs the caller about the resulting score (positive for pass, negative for fail) and is told if the verification passed or failed.

The single user verification process uses the following Nuance VocalPassword API methods: `Verify`.

Group verification (Identification)

The goal of the group verification process is to verify caller's identity from among a group of possible callers by matching the caller's utterance against voice models collected from individual members of the group.

You can use this feature for speaker identification during a conference call or in the case of a “family calling plan”, where multiple users exist for the same account and any one of the users can access the account.

The sample application prompts the caller to enter three account numbers to verify against. (Typically, a production level application already knows the account numbers to verify against.)

The application prompts the caller to say the phrase used for enrollment just once. Then, the application informs the caller about the resulting score against each user (positive for pass, negative for fail).

The group verification process uses the following Nuance VocalPassword API methods: `CreateAdhocGroup`, and `Identify`.

Unenrollment

The goal of this process is to delete the caller's database entry. This includes the speaker's voiceprints, voice templates, and other speaker related data.

The enrollment process uses the following Nuance VocalPassword API methods: `DeleteSpeaker`.

Liveness verification

Liveness verification adds a measure of security by verifying that the caller is live and not trying to trick the verification system with a pre-recorded prompt. When the speaker verification system performs a liveness test, the system prompts the caller to repeat a sequence of random digits. The verification system then recognizes the digits to see if they are the ones the caller was asked to speak and then authenticate the liveness utterance comparing the liveness utterance to the most recent verification utterance.

The liveness verification process uses the following Nuance VocalPassword API methods: `Verify`.

Nuance VocalPassword API methods used by sample applications

The following table identifies the Nuance VocalPassword API methods which are used in the MPS Developer and VoiceXML sample applications. For information about —

- Application development with MPS Developer, see Nuance VocalPassword MPS Developer Application programming on page 41.
- Application development with VoiceXML, see Nuance VocalPassword VoiceXML application programming on page 83.
- Complete Nuance VocalPassword API, refer to the documentation provided by Nuance Communications, Inc.

Nuance API Function	Function Description	MPS Developer	VoiceXML
`IsTrained`	The `IsTrained` method checks whether the speaker's voiceprint has a valid voice template. A speaker cannot be verified unless their voiceprint is trained.	HTTP SOAP message	HTTP POST message

Nuance API Function	Function Description	MPS Developer	VoiceXML
`DeleteVoiceprint`	The `DeleteVoiceprint` method deletes the voice template, the enroll segments collection, and additional data belonging to the voiceprint, the speaker itself and other voiceprints of that speaker are not deleted.	HTTP SOAP message	HTTP POST message
`Enroll`	The `Enroll` method adds an audio segment to the speaker's enroll segments collection. When enough audio has been collected, this method will create the voice template. If the voiceprint is already trained, this method adapts the voice template with the new added audio segment. A successful Enroll request always adds the given audio to the speaker's enroll segments collection, and may perform additional actions according to the enroll segment's status and configuration. When the enroll segments collection is ready (there is enough audio and enough segments), and depending on the AutoTrain parameter, the `Enroll` method automatically invokes the training process.	HTTP SOAP message	HTTP POST message
`Verify`	The `Verify` method checks if a given audio matches a speaker's voice template.	HTTP SOAP message	HTTP POST message
`CreateAdhocGroup`	The `CreateAdhocGroup` method creates a temporary group of speakers. This method is mainly used when the list of speakers to be used for identification or fraud detection changes constantly. The Adhoc group is not persistent and is kept in memory only during the session.	HTTP SOAP message	HTTP POST message
`Identify`	The `Identify` method checks if the given audio matches any of the voiceprints in a group.	HTTP SOAP message	HTTP POST message
`DeleteSpeaker`	The `DeleteSpeaker` method deletes a speaker from the system. This method deletes all the speaker's voiceprints, voice templates, and other speaker related data.	HTTP SOAP message	HTTP POST message

Verification types

VocalPassword supports the use of three verification engines: Text-Dependent Speaker Verification, Text-Independent Speaker Verification and Text-Prompted Speaker Verification.

- **Text-Dependent Speaker Verification** is performed using a pass phrase to identify the speaker.
  - Enrollment: Enrollment is normally performed using three consecutive repetitions of the selected pass phrase. The pass phrase can be spoken in any language or accent, and should contain at least 2 seconds of net audio.
  - Verification: VocalPassword verifies the speaker by comparing a single repetition of the enrolled pass phrase to the voice template stored in the system's voiceprints repository.
  - Identification: Speaker identification is carried out by comparing a single repetition of the speaker’s pass phrase to a list of voice templates.
- **Text-Independent Speaker Verification** is performed using the speaker’s voice alone regardless of the content spoken.
  - Enrollment: Enrollment is performed using samples of the speaker’s voice. Longer voice samples will allow for a more accurate voiceprint. The amount of speech needed for text-independent enrollment needs to be at least 45 seconds of net speech.
  - Verification: VocalPassword verifies the speaker by comparing the speaker’s voice sample to the voice template stored in the system’s voiceprint repository.
  - Identification: Speaker identification is carried out by comparing the speaker’s voice to a list of voice templates.
- **Text-Prompted Speaker Verification** is performed using a specific utterance that is a random subset of the phrases used for enrollment.
  - Enrollment: Enrollment is performed using a sample of the speaker’s voice repeating a predetermined phrase. This phrase includes all the elements that will be used for verification.
  - Verification: The calling application generates a prompt: a random phrase that is a subset of the phrase requested during enrollment. The speaker repeats the phrase and the system verifies the speaker by comparing the speaker’s voice sample to the voice template stored in the system’s voiceprint repository.
  - Identification: The calling application generates the prompt as in the verification process. The speaker repeats the phrase and the system verifies the speaker by

comparing the speaker's voice to a list of voiceprints belonging to the speaker or to multiple speakers.

The calling application asks the caller to say something unique for that specific call. For example: "Please repeat the following numbers: 0-3-7-2." Enrollment for Prompted Verification requires the user to say a predefined list of items. This might be the numbers 1-9 or it could be any list you choose (for example a list of cities). During verification, the user is required to say a random subset of this list.

- Prompted verification provides greater protection against playback attacks because the pass phrase is a single-use string created only for the session in progress.
- Prompted verification requires you to change your enrollment workflow to prompt your speakers to say the predefined list of items (for example, the numbers 1-9) and record these individually for verification.

VocalPassword supports three different methods to fight against recording attacks:

- Text prompted passphrase verification
- Playback detection
- Liveness detection

These methods can not be mixed (for example, the playback detection cannot be activated together with Text Prompted verification).

Chapter 2: Installation and Configuration

This chapter covers:

1. Installation and configuration overview
2. Install and configure VocalPassword Server and Database Nodes
3. Install and configure MPS Application Processor Node

Installation and configuration overview

This chapter describes the procedures for installing and configuring software the Nuance VocalPassword 8.x functionality in the MPS environment. Software must be installed and configured on three nodes: the VocalPassword Server node, the database node, and the MPS Application Processor node.

Install and configure VocalPassword Server Node

This section describes the software installed and configured on the Nuance VocalPassword node.

Overview

VocalPassword is installed on a separate node. See Installation roadmap for VocalPassword Node on page 17.

VocalPassword 8.x requires a valid license to be available before installation.

Prerequisite to using VocalPassword 8.x

After you install and configure software on both the VocalPassword and database nodes, ensure that the VocalPassword can connect with and access the database.

Install and configure Database Node

This section describes the software that is installed and configured on the Database node.

Procedure

1. Install Windows Server 2008 R2 x64 (Standard Edition).
   a. Run the Windows Server 2008 R2 x64 installer and follow the on-screen instruction.
   b. Choose **Windows Server 2008 R2 Standard (Full Installation) x64**.

      Note:

      Other variants are not tested.
2. Install Microsoft SQL Server 2008 Enterprise R2.
   a. Run the Microsoft SQL Server 2008 Enterprise R2 installer.
   b. Press **OK** to install .NET framework (if prompted).
   c. In SQL Server Installation Center choose **Installation** > **New installation or add features to an existing installation** and follow the instructions on the screen.
   d. When prompts, set all services (which are left empty by the installer) to start under the “Network Service” user account.
   e. Use the Windows Authentication mode.
   f. In **Specify SQL Server administrator**, set the system “Administrator” account.
   g. Choose **Install the native mode default configuration**.

Installation roadmap for VocalPassword Node

Use this roadmap for installing and configuring software on the VocalPassword node. For additional support for installation procedures, refer to documentation distributed by Nuance Communications, Inc. See Vendor reference documentation on page 6.

Before you begin

1. Install Windows Server 2008 R2 x64 (Standard Edition)
    a. Run the installer, follow the on-screen instructions.
    b. Choose **Windows Server 2008 R2 Standard (Full Installation) x64**.

    Note:
    Other variants are not tested
2. Set UAC (User Access Control) to minimum (or off) where relevant.
    a. Click **Start**, and then click **Control Panel**.
    b. In Control Panel, click **User Accounts**.
    c. In the User Accounts window, click **User Accounts**.
    d. In the User Accounts Tasks window, click **Change User Account Control Settings**.
    e. Change settings to **Never notify**.
    f. Restart the system to apply the change immediately.
3. Install IIS and ASP.NET

    Make sure that IIS is up and running with ASP and ASP.NET activated.

    In Windows Sever 2008: From servers roles, ensure the following IIS services are available:

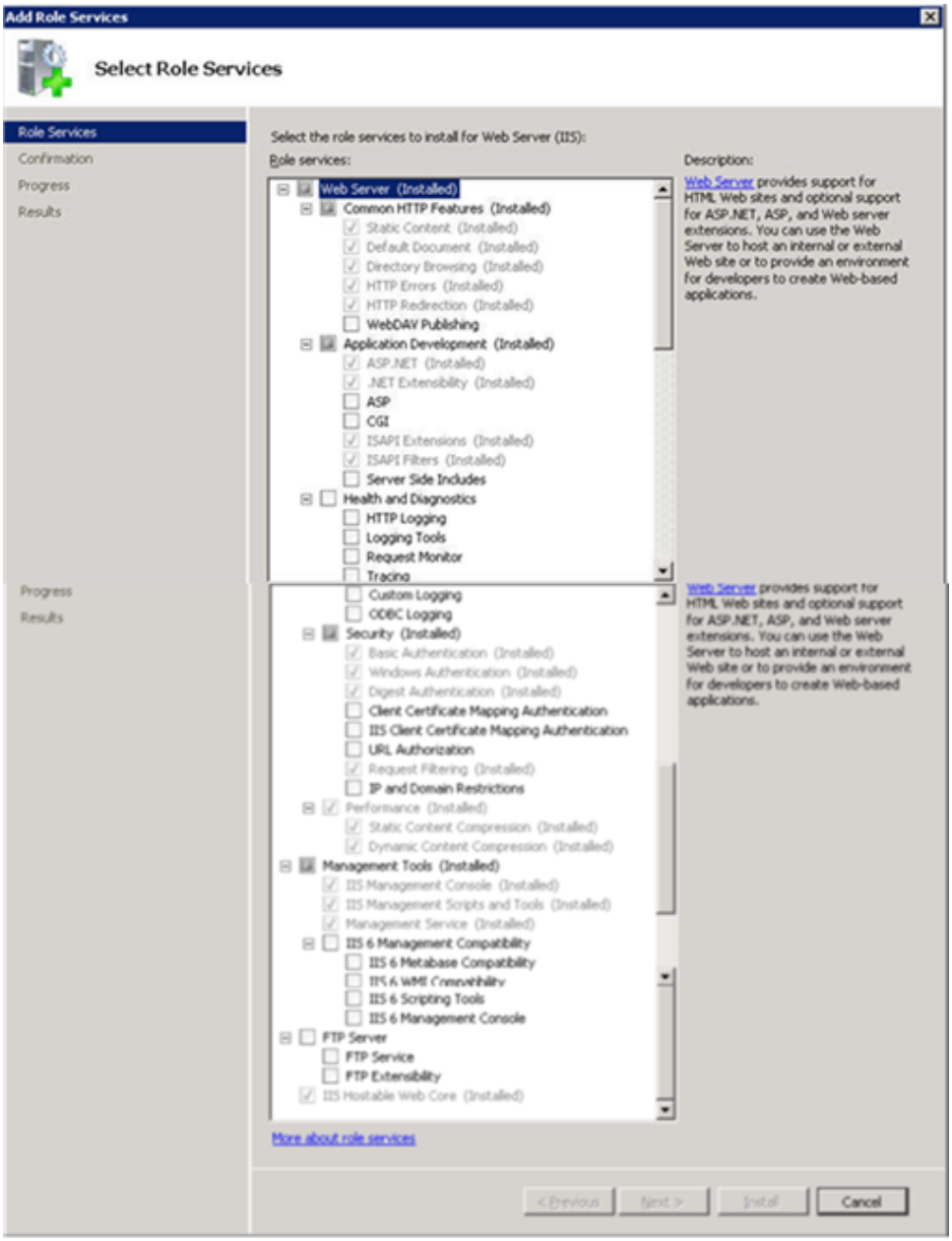


Figure 1: IIS installation - Server Roles

4. Install .Net Framework v4.

   .Net Framework v4 is installed as part of VocalPassword 8.3 installation.

5. Restart the computer.

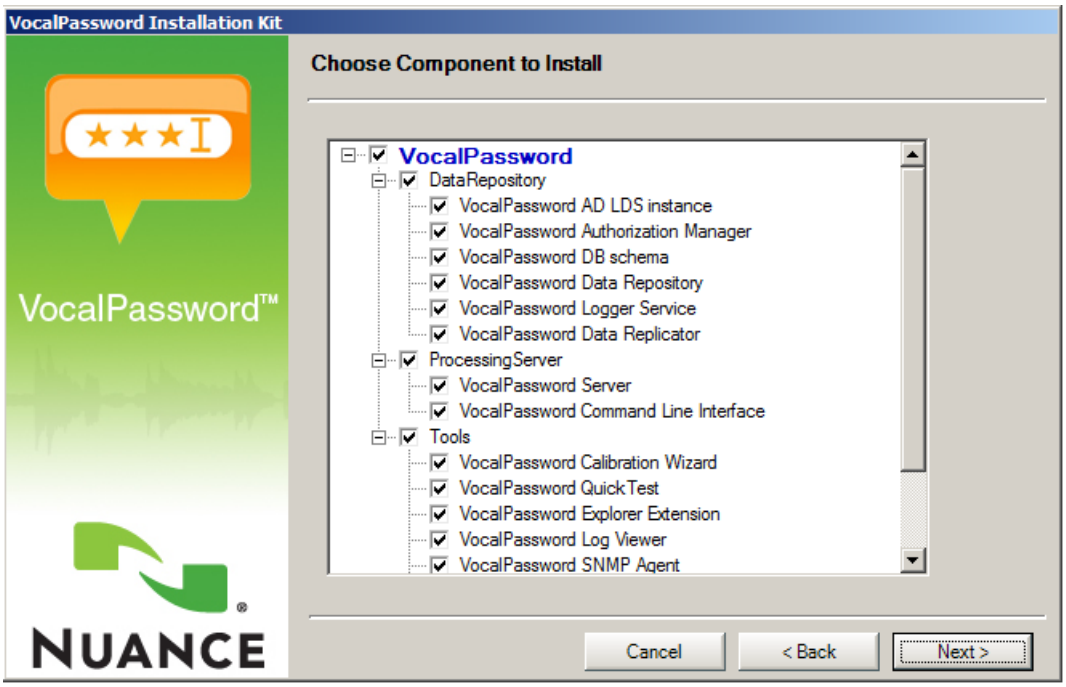
Procedure

1. To create a database in the SQL management studio, do the following:
   a. In the **DB Name** field, type `VP`

b. From the **DB File Location**drop-down list, choose `Not important`

2. Started the VocalPassword 8.3 installer (64bit).
   The system displays the VocalPassword Installation Kit wizard.
3. Select **Install** and click **Next**.
   The system displays with option to choose the components.

Figure 2: VocalPassword installer - component choice window



By default, the system install all components. The VocalPassword system comprises of two components, namely, the Processing Server and the Data Repository Server:

- Processing Server is the main processing unit of the VocalPassword system. Processing Server hosts the speaker verification software which performs algorithmic processing, controls client services and acquires audio through API calls. You use multiple servers optionally in a redundancy scheme for high availability purposes or in a load balancing scheme for scalability.
- Data Repository Server is responsible for the storage of system and voiceprint data. Data Repository Server runs the system's database, Active Directory/ ADAM, and file system (for storing audio recordings). Data Repository Server can also point to a remote database or Active Directory / ADAM. VocalPassword supports the use of two data repository servers for high availability.

You can install the VocalPassword components on all-in-one (both Data Repository and Processing Server on one machine) or in a distributed component architecture

on separate machines. When choosing to install the components on separate machines, ensure that you install the Data Repository before the Processing Server.

For an all-in-one installation, leave all components checked and click **Next**. To install the Processing server only, uncheck all components under Data Repository. To install the Data Repository only, uncheck all components under Processing Server and Tools.

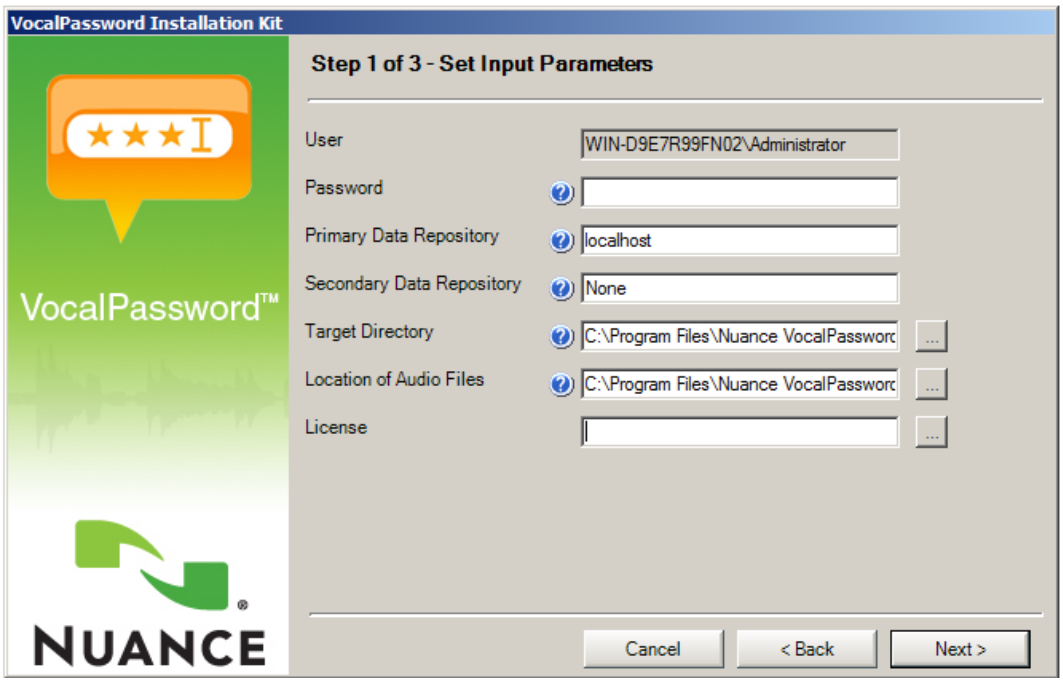
Note:

The Persistent Data Replication (PDR) component should only be installed in a high availability data replication environment.

4. Click **Next**.

   The system displays with option to choose the VocalPassword Database Input Parameters.

Figure 3: VocalPassword installer - parameters window



5. In the VocalPassword Input Parameters wizard, do the following:
   a. In the **User** field, enter the user name who is currently logged in to the system as `[Domain]\[User name]`.
   b. In the **Password** field, enter the user's password. This is the Windows Password.
   c. In the **Primary Data Repository** field, enter the host name.

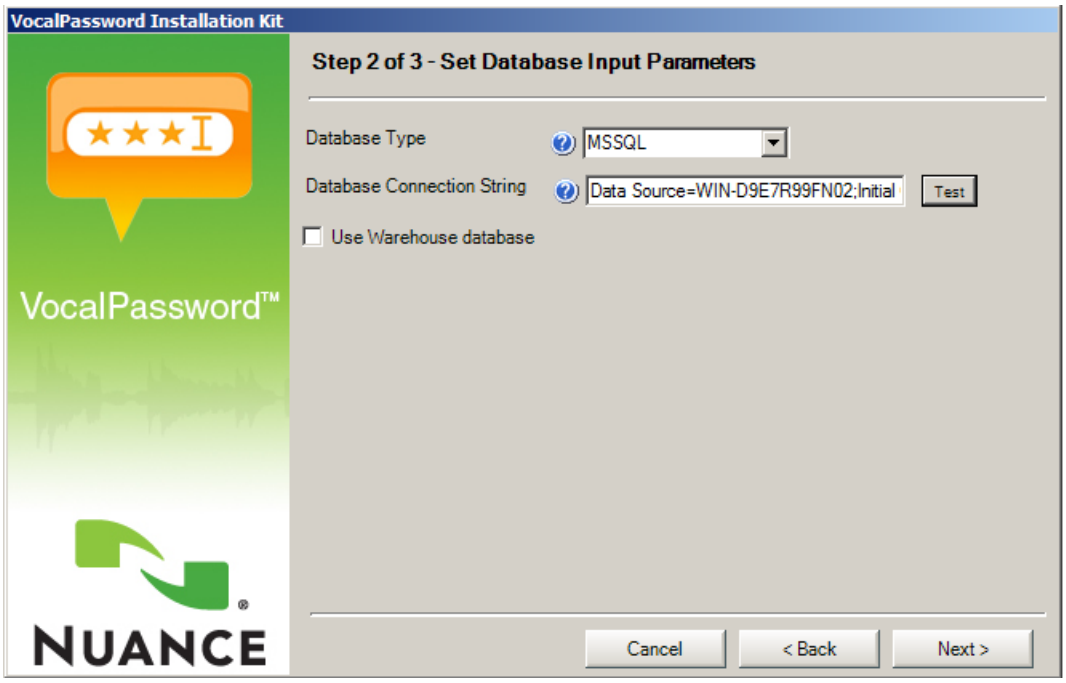
d. In the **Secondary Data Repository (optional)** field, if there is no secondary Data Repository, leave as `None`.
e. In the **Target Directory** field, enter the directory where you want to install the system.
f. In the **Location of Audio Files** field, enter the directory where you want to store the audio files.

   The audio file folder selected must have a sufficient available space, as the VocalPassword system stores audio in that folder.
g. In the **License** field, enter the license file provided by Nuance.

6. Click **Next**.

   The system displays with option to choose the VocalPassword Database Input Parameters.

Figure 4: VocalPassword installer – database parameters window

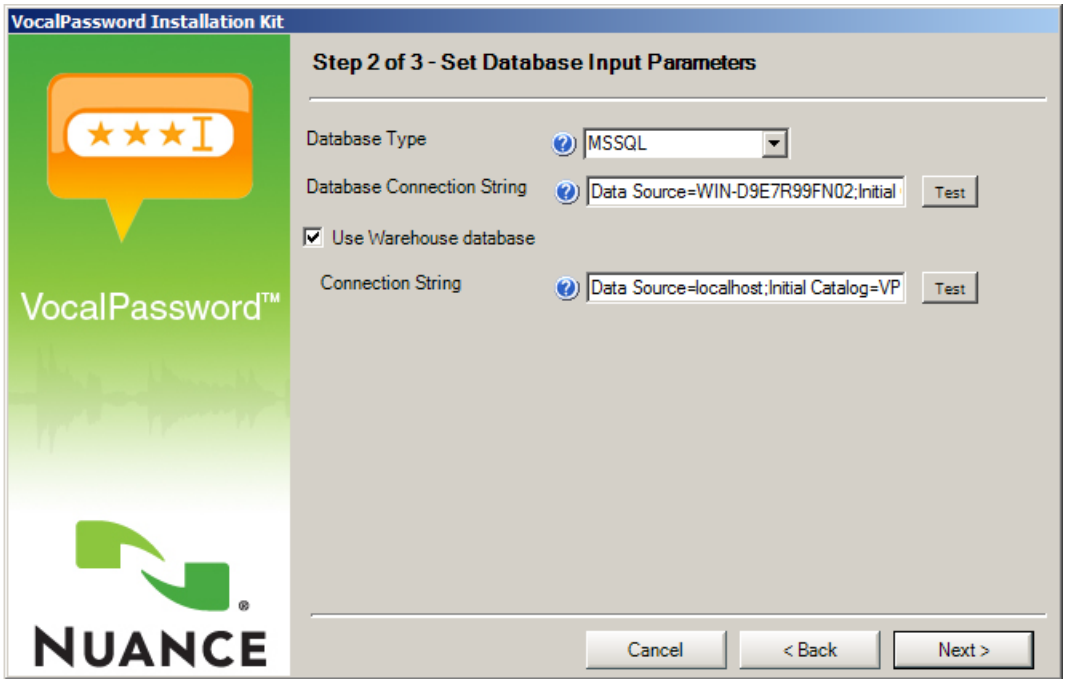


This screen appears only when installing a Data Repository. You must Select the database type, and insert a connection string. Click **Test** to test connectivity. When clicking **Next**, the system tests the connectivity. If the connectivity fails, installation will not proceed to the next step.

The VocalPassword system supports optional report and query performance optimization. This feature is intended for large scale deployments with substantial databases. The system supports two databases: the online database and the warehouse database. Reports and queries runs on the warehouse database, while the day to day processes runs on the online database. The feature requires creation of an additional database, as well as constant database update. These are not

automated by the system and are the responsibility of the Application Designer. To utilize this feature, check the **Use warehouse database** options, and insert a valid connection string.

Figure 5: VocalPassword installer – database parameters window / warehouse database



7. Click **Next**.

   The system displays with option to choose the VocalPassword Directory Server Input Parameters wizard.

Figure 6: VocalPassword installer – Data Repository

The system displays the above screen only when installing a Data Repository. You must specify the AD LDS data directory. This directory requires sufficient storage space. When working with a different LDAP server, check the **Use different directory server** options and refer to appendix 1 for more information.

8. Click **Next** to complete the installation.

   The system take about 10-15 minutes to complete the installation. It is normal for progress bar not to move in the first few minutes as it is running some 3rd party installers in silent mode.

9. Once installation is complete run the **Quick Test Tool** from the **Start** menu (in VocalPassword folder).

   - Quick Test Tool is a mini calling app that tests all aspects of installation including licenses.
   - Installation is complete if all tests return success.

   Windows Firewall configure rules to accept incoming HTTP traffic on port 80

10. Select **Server Manager** > **Roles** > **Web Server (IIS) > Internet Information Services Manager** and configure the authorization on the IIS server as follows:.

    a. Open **localhost** > **Sites** > **Default Web Site**.
    b. Click the **Authentication** icon.
    c. Enable **Basic Authentication**.
    d. Enable **Windows Authentication**.
    e. Disable all other authentications except Basic and Windows authentication.
    f. Open **localhost** > **Sites** > **Default Web Site** > **VocalPassword.**.

g. Click the **Authentication** icon.
h. Enable **Basic Authentication**.
i. Enable **Windows Authentication**.
j. Disable all other authentications except Basic and Windows authentication.

Install and configure MPS Application Processor Node

This section describes the installation of software required for VocalPassword 8.x on the MPS Application Processor node.

Assumptions

Before you begin installing the software on the MPS Application Processor node, it is assumed that —

- the MPS AP is installed with the Avaya Self Service base software and with MPS Developer
- MPS Manager is installed either on the MPS AP or on another node, and
- the MPS AP is fully configured for operation.

Information about installing Avaya software on the MPS AP node is available in the *Installing MPS Software on the Solaris Platform Installing MPS Software on the Windows Platform* document.

Installation and configuration required for VocalPassword 8.x

Install the required software and configure the MPS AP node for applications running on the node can use the Nuance VocalPassword API. This includes installing the software packages namely, PERIxmlc, PERIhtmls, PERIvxml, copying applications, and configuration and MMF files to the directories where they are required to reside to add alarms to the database.

MPS Developer application

Procedure

1. Install the PERIhtmls and PERIxmlc packages on the MPS Application Server.

2. Change directory to `$MPSHOME/PERIppro/sample/vocalpassword` and verify the contents include the following:
   - VocalPassword.ppr: The MPS Developer Application.
   - VocalPassword.vex: The executable code generated from the MPS Developer Application.
   - VocalPassword.acfg: The application configuration file needed to run the application in tappman.
   - vocalpassword.mmi: Vocabulary index file containing prompts spoken in VocalPassword.ppr.
   - vocalpassword.mmd: Vocabulary data file containing prompts spoken in VocalPassword.ppr.
   - VocalPassword.alarm: A file containing alarm definitions necessary for the application.
   - VocalPasswordConfig.xml: A configuration file for the VocalPassword application.
3. Copy the vocabulary files vocalpassword.mmi and vocalpassword.mmd to `/mmf/peri` (`C:\mmf\peri`).
4. Enter **`vsh`** and type the following. (You may exit **`vsh`** afterwards):
   - Solaris:

     `vmm mmfload /mmf/peri/vocalpassword`
   - Windows:

     `vmm mmfload C:\mmf\peri\vocalpassword`
5. To make the vocabulary load every time SRP starts, add the following line to `$MPSHOME/mpsN/etc/vmm-mmf.cfg` (where N is the MPS component number):
   - Solaris:

     `mmfload /mmf/peri/vocalpassword`
   - Windows:

     `mmfload C:\mmf\peri\vocalpassword`
6. Install the alarms into the database. Type the following:
   `alm fuser VocalPassword.alarm`
7. Configure the `VocalPasswordConfig.xml` file.
8. Copy the executable (.vex) and configuration (.acfg) files to the `$MPSHOME/mpsN/apps` directory (where N is the MPS component number).
   - Solaris:

```
cp VocalPassword.vex VocalPassword.acfg $MPSHOME/mpsN/
apps
```

- Windows:

```
cp VocalPassword.vex VocalPassword.acfg %MPSHOME%\mpsN
\apps
```

9. Add `$MPSHOME/PERIppro/sample/verifier` to the PPROPATH:
   - Solaris:

```
setenv PPROPATH ${PPROPATH}:/${MPSHOME}/PERIppro/sample/
vocalpassword
```

   - Windows:

```
set PPROPATH=%PPROPATH%:\PERIppro\sample\vocalpassword
```

10. Assign and start the application.
    - Solaris:

```
tappman -c mps.N -l<line range> -app $MPSHOME/mpsN/apps/
VocalPassword assign start
```

    - Windows:

```
tappman -c mps.N -l<line range> -app %MPSHOME%\mpsN\apps
\VocalPassword assign start
```

VoiceXML application

Procedure

1. Install the PERIvxml package on the MPS Application Server.
2. Create an empty MMF named **/mmf/peri/cmrdata** and add the following line to the`$MPSHOME/mpsN/etc/vmm-mmf.cfg` file:

   mmfrecord /mmf/peri/cmrdata
3. To effect the change immediately, type the following command from a VSH prompt:

   vmm mmfrecord /mmf/peri/cmrdata

   This command creates a temporary CMR recording file.
4. Configure oki_recorder resources in the `tms.cfg` file.
5. Change directory to `$MPSHOME/PERIvxml/samples/vocalpassword` and verify the contents include the following files:
   - `VP-verifyUser.vxml`

- VP-verifyLiveness.vxml
- VP-verifyGroup.vxml
- VP-functions.js
- VP-queryUserExt.vxml
- VP-main.vxml
- VP-enrollment.vxml
- VP-deleteUser.vxml
- VocalPassword.alarm
- vocalpassword.mmd
- vocalpassword.mmi

6. Put the provided VXML pages to the $MPSHOME/mpsN/apps directory path.
7. Edit the settings in VP-main.vxml file as described below:
   - **serviceURL** - The base URL to VocalPassword Web-service API. You must have to modify the IP address and set it to the IP address of the VocalPassword Server node.
   - **AvayaHTTPUsername** - User name for HTTP basic authorization. By the default, the user name is the Windows Administrator's user name.
   - **AvayaHTTPPassword** - Password for HTTP basic authorization. By the default the password is the Windows Administrator's password.
   - **configSetName** - The Configuration Set name. This must not be changed.
   - **EnableTextPromptedMode** - To enable or disable the Text-Prompted mode. Set "true" to enable or "false" to disable.
   - **EnableLivenessTestWithDictionary** - To use a user-defined XML dictionary file for the Liveness Detection. Set "true" to enable or "false" to disable.
8. Copy the vocabulary files vocalpassword.mmi and vocalpassword.mmd to /mmf/peri.
9. Enter vsh and type the following:
   vmm mmfload /mmf/peri/vocalpassword
10. To make the vocabulary load every time SRP starts, add the following line to the $MPSHOME/mpsN/etc/vmm-mmf.cfg file
    mmfload /mmf/peri/vocalpassword
11. Install the alarms into the database. Type the following:
    alm fuser VocalPassword.alarm

12. Assign and start the `VP-main.vxml` application using MPS Manager.

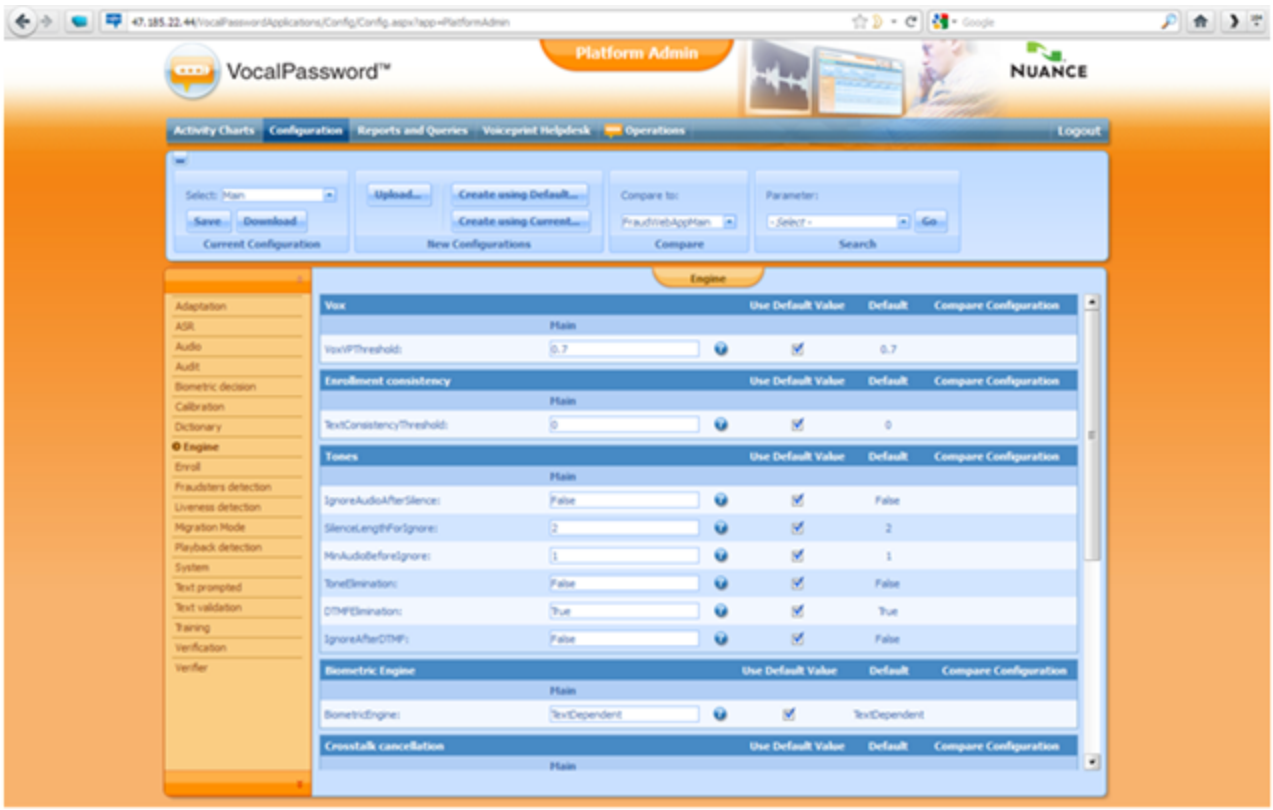
VocalPassword advanced configuration

VocalPassword Platform Admin is a web-based Application that provides a variety of tools for properly setting up the system and its biometric functionality as well as for managing speakers, voiceprints and groups. Use this application to configure VocalPassword, perform queries and reports, and monitor the system usage.

For more information refer to Vendor reference documentation on page 6.

To edit VocalPassword configuration set using Platform Admin application, open the following link:

`http://<VocalPassword-Server-IP>/VocalPasswordApplications/Config/Config.aspx?app=PlatformAdmin`



Configuring Liveness Detection with Dictionary

About this task

Liveness detection is a unique and patented method which significantly reduces recording threats. This method uses text-independent voice biometrics technology to compare the voice sample captured during text-dependent verification with an additional sample which is the speaker's repetition of a random or semi-random sentence. By combining the obtained biometrics score and a score which is extracted by validating the content of the repeated sentence a liveness detection score is extracted.

The following steps are required to configure Liveness Detection with Dictionary:

Procedure

1. Liveness Detection feature requires a Dictionary with random prompts to be created and uploaded to the system, following is a sample of a Liveness Detection Dictionary:

```
<?xml version="1.0" encoding="utf-8" ?>
<Dictionary>
  <Verify>
    <Prompt>
      <Text>Better a good plow mule than a lame horse</Text>
      <IVRInfo>please say Better a good plow mule than a lame horse</
IVRInfo>
      <LivenessBgModel>TIEagle</LivenessBgModel>
    </Prompt>
    <Prompt>
      <Text>Don't worry about the mule just load the wagon</Text>
     <IVRInfo>please say Don't worry about the mule just load the wagon< /
IVRInfo>
      <LivenessBgModel>TISunRises< /LivenessBgModel>
    </Prompt>
  </Verify>
</Dictionary>
```

2. Create appropriate background models (see <LivenessBgModel> tag in dictionary) for each phrase in the dictionary.
3. Set LivenessDetectionDictionary to liveness dictionary name which contains random prompts.
4. Set LivenessDetectionBgModelForTDSegment to the name of Text-Dependent background model.
5. Set BiometricEngine to "TextDependent".

Note:

- If you are using Avaya sample MPS Developer application, you must also set EnableLivenessTestWithDictionary paramater to true in.

- If you are using Avaya sample VoiceXML application, you must also set EnableLivenessTestWithDictionary paramater to true in VP-main.vxml application.

Configuring for Text-Independent Verification

About this task

In order to work with Text-Independent audio in VocalPassword, parameter “BiometricEngine” should be set to “Text-Dependent” but a background model should be created using GMM algorithmic engine. You then enroll against the created background model.

The following steps are required to configure Text-Independent verification:

Procedure

1. Record about 100 prompts with a different texts and voices; store them to `.wav` format into temporary folder.
2. Create Background model using Calibration Wizard tool (use GMM algorithm).
3. Set BiometricEngine to **TextDependent**.
4. Set BackgroundModel parameter in Enroll section in Default Config Set to setting to background model created in step 2.
5. Enroll with long utterances (The amount of speech needed for text-independent enrollment however needs to be at least 45 seconds of net speech).

Configuring for Text-Prompted Verification

About this task

The following steps are required to configure Text-Prompted verification:

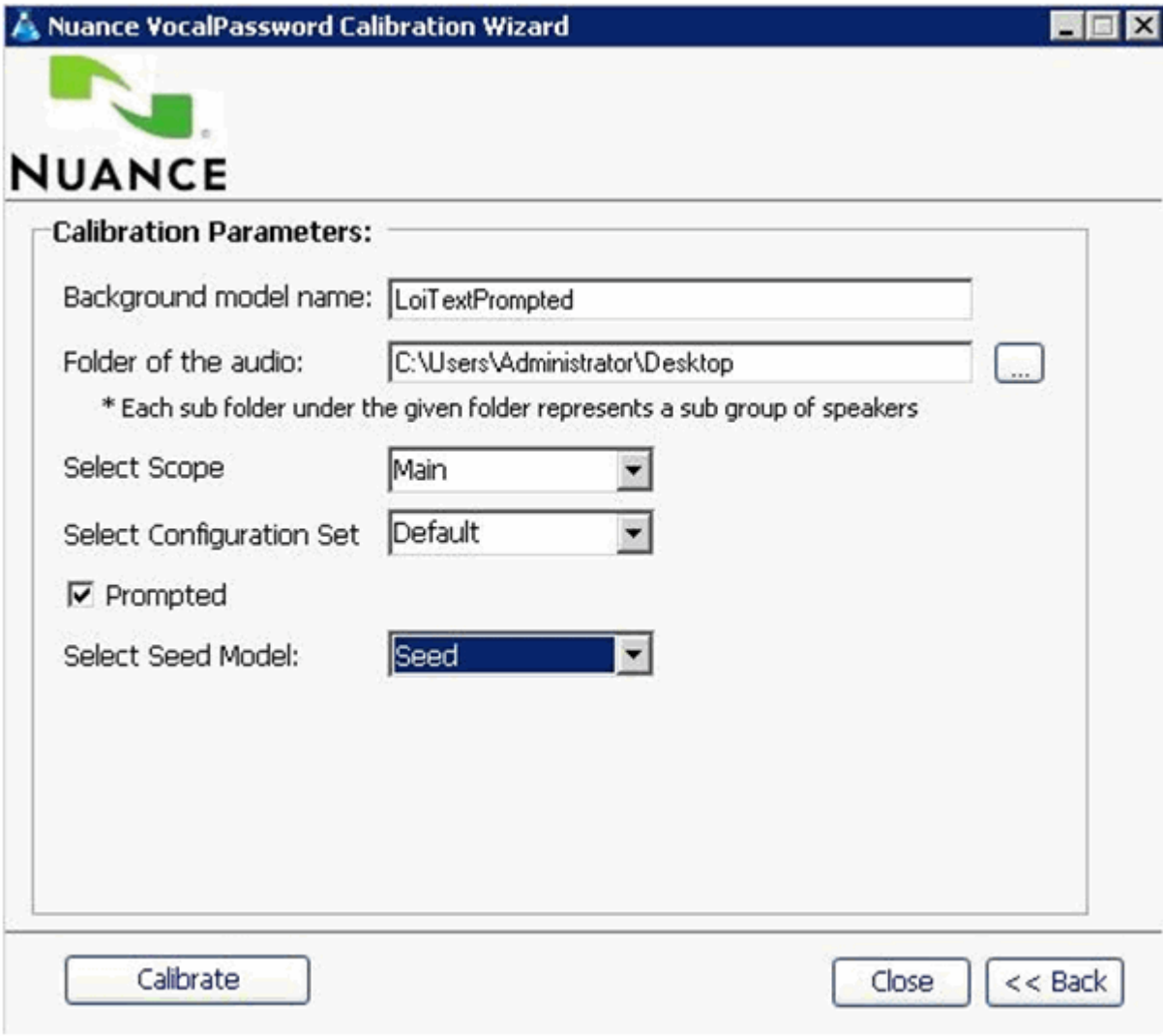
Procedure

1. Record about 100 prompts with a different texts and voices; store them to `.wav` format into temporary folder.
2. For every single audio file you must create a corresponding text file with the content of the audio in it:

   For example - if the phrase is 0123456789 the text file will contain 1 line with the text 0 1 2 3 4 5 6 7 8 9.

   If the audio file name is `123.wav`, the text file should be `123.txt`.

3. Create a new Background model and select the **Prompted** checkbox .
4. Choose **Seed** from the list.



Creating a Text Prompted Background Model requires the use of a Seed.

A Seed is a model that describes how each Atom (digit, phoneme or word) we plan to use in the Text Prompted verification is uttered. It is the foundation of the background model used for biometric verification; therefore, the process of creating a Text Prompted Background Model requires the use of a Seed. The Seed is Language dependent and Lexicon dependent.

For example, when using digits as the prompt phrase, we will create a Seed that models each digit's (Atom) pronunciation. VocalPassword does not support the process of creating Seeds; in order to create a Seed, contact Nuance.

5. Set BiometricEngine to **TextPrompted**.

6. Set BackgroundModel parameter in Enroll section in Default Config Set to setting to background model created in step 3.

Configuring to catch inconclusive verification results

About this task

The verification (authentication) decision is determined according to the thresholds. A match decision is returned by the system when the score is above the upper threshold. A mismatch decision is returned by the system when the score is below the lower threshold. An inconclusive decision occurs when the score is between the upper and lower thresholds.

For example to always catch inconclusive result, set the following parameters:

Procedure

1. Click on **Biometric decision** and set:
   - LowerThreshold to 100.
   - UpperThreshold to 100.
2. Click on the **Save** button to save the change.

Configuring Playback Detection

Playback detection feature enables the system to identify verification segments which are possible playbacks of recorded utterances. The playback detection mechanism runs as part of the verification process, recognizing identical audio segments.

Note:

Playback detection cannot be enabled in the Text-Prompted mode.

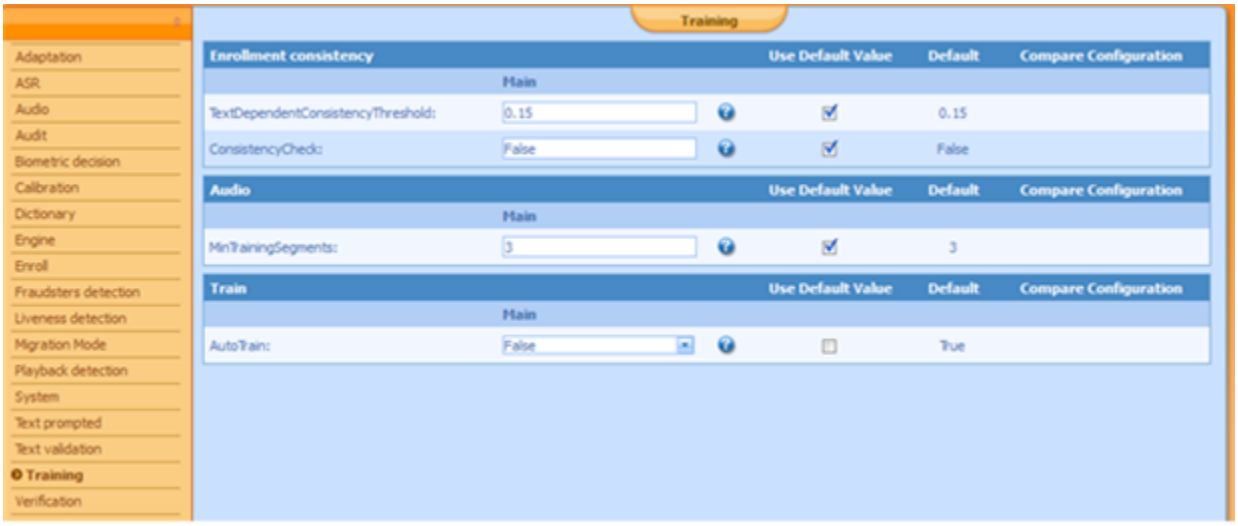
How playback detection works:

1. When audio is recorded into VocalPassword, the system simultaneously creates and stores a “footprint”. This is a miniature representation of the audio file.
2. The next time audio is received for verification, the footprint is compared with all previous footprints.
3. A result is returned for each comparison between the newest footprint and all stored footprints. The configuration parameter PlaybackDetectionThreshold defines the threshold used to decide whether the newest audio footprint represents an audio file that is “too good” and is therefore a recording of a previous pass phrase utterance.

To enable or disable playback detection feature use **PerformPlaybackDetection** parameter in Platform Admin.

Configuring the number of utterances required for enrollment

VocalPassword requires at least three pass phrase utterances for enrollment. You can change the number of utterances required for enrollment using the MinTrainingSegments configuration parameter. It is recommended to set the minimum to three. Possible values are 1-30.



Configuring Unsupervised adaptation

Adaptation is the process of using new audio to update an existing voice template, enabling voiceprints to adapt to changes. It allows each speaker to maintain an accurate voice template according to their changing regular background noises and voice tones that shift with age. There are two types of adaptation:

- Supervised adaptation – Explicitly perform adaptation by calling the Enroll API method with new audio.
- Unsupervised adaptation – Perform adaptation implicitly as a result of successful verification when the following criteria are met:
  - A certain verification score has been reached. If a lower score has been returned from the verification process, audio is not used to update the existing voice template.
  - A certain configurable interval of time has passed since the last time the voice template was updated.

Considerations:

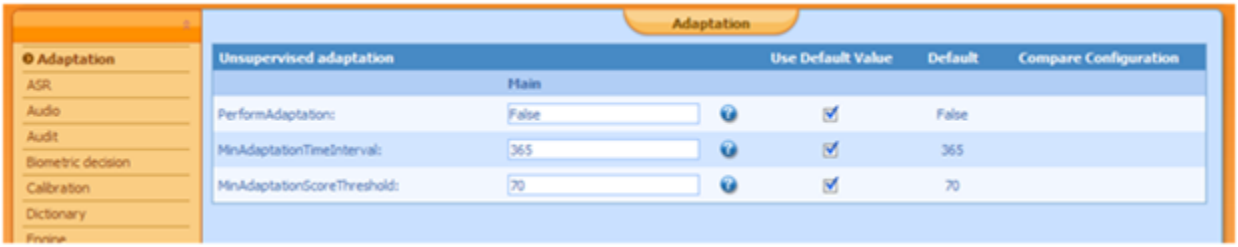
Using unsupervised adaptation has a few drawbacks:

- Bad quality voice templates will not be adapted because the verification score will be lower than the adaptation threshold.
- There is a risk that due to false acceptance errors: An imposter's voice will be used for the adaptation process and will corrupt the voice template.

Using Supervised adaptation eliminates these drawbacks but requires an external method of authentication as in the Enrollment process. This external method of authentication is not always available.

There are three configuration parameters that control the behavior of adaptation:

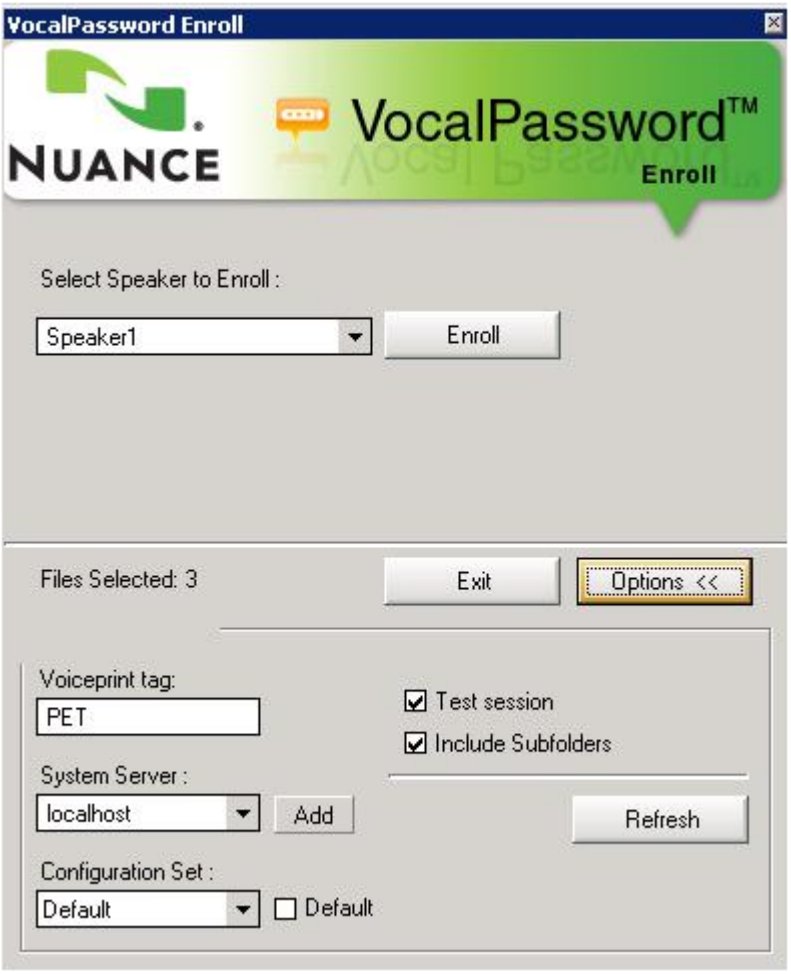
- **PerformAdaptation** - Specifies whether to implicitly perform voiceprint adaptation when adaptation criteria are met. Set this parameter to **True** to enable unsupervised adaptation.
- **MinAdaptationTimeInterval** - The minimum time interval (in days) since the last voice template's update. Setting this value to a very short interval may cause an excessive adaptation rate which has no added value. Setting this value to a longer interval may cause a voice template to be outdated.
- **MinAdaptationScoreThreshold** - The minimum verification score that triggers unsupervised adaptation. Using a threshold that is too high will in practice prevent adaptation of voice templates that are not best-performing. However, using a threshold that is too low increases the risk of performing adaptation with an imposter's voice.



Supervised adaptation

To perform a supervised adaptation use the following steps:

1. Choose **Windows Start Menu** > **VocalPassword** > **Operations** > **Enroll**. Choose 3 WAV files.
2. Then, input the speaker ID, leave the voiceprint as “PET” and configuration set “Default” as default.



3. Click **Enroll**.

   The Enroll pane closes and an enroll icon appears in the system tray. A notification balloon indicates Enrollment success or failure. The Enroll icon disappears after 5 seconds.

Creating Background Models with the Calibration Wizard

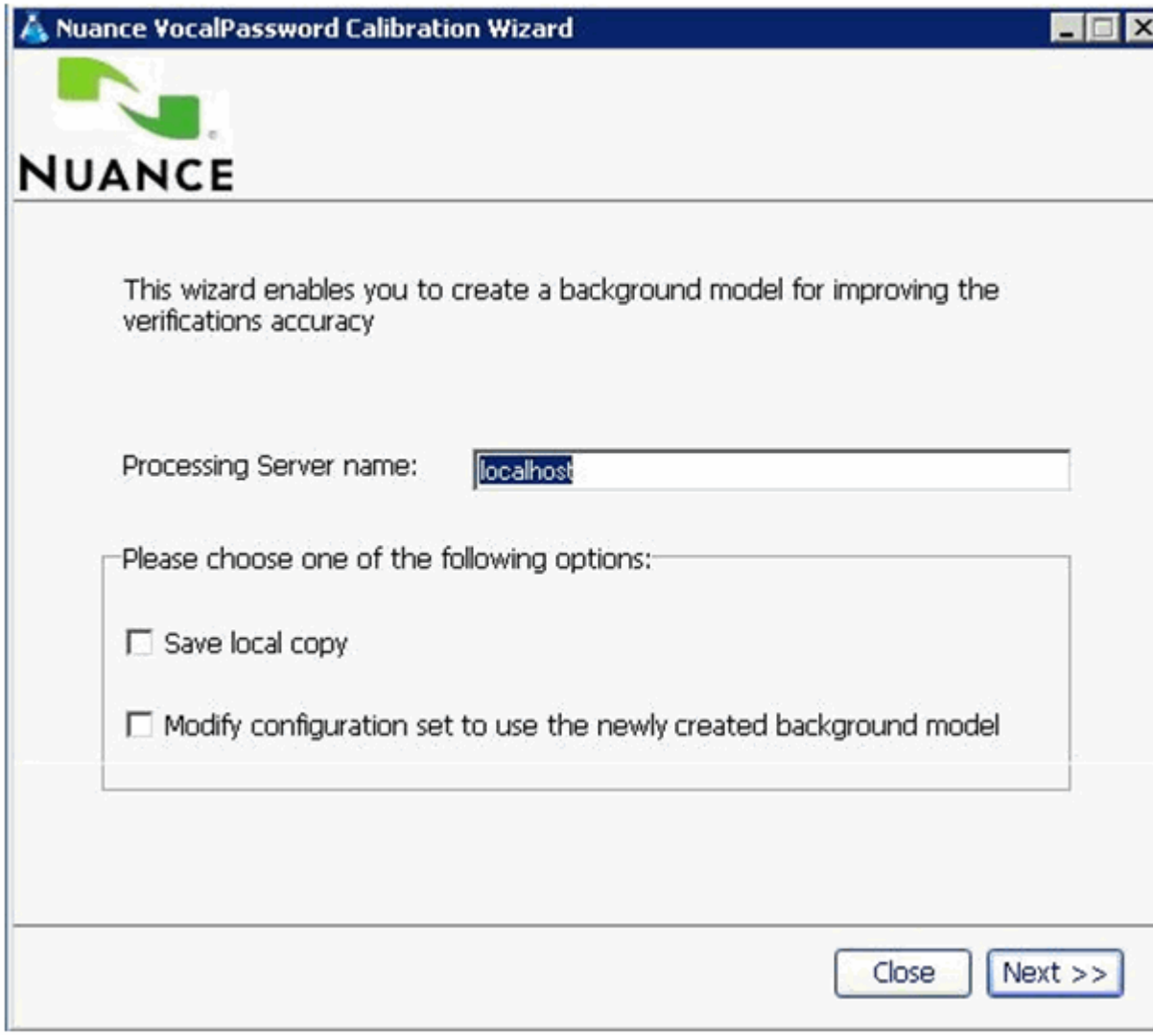
About this task

After collecting the audio, you can create a Background Model using the Calibration Wizard.

To access the Calibration Wizard:

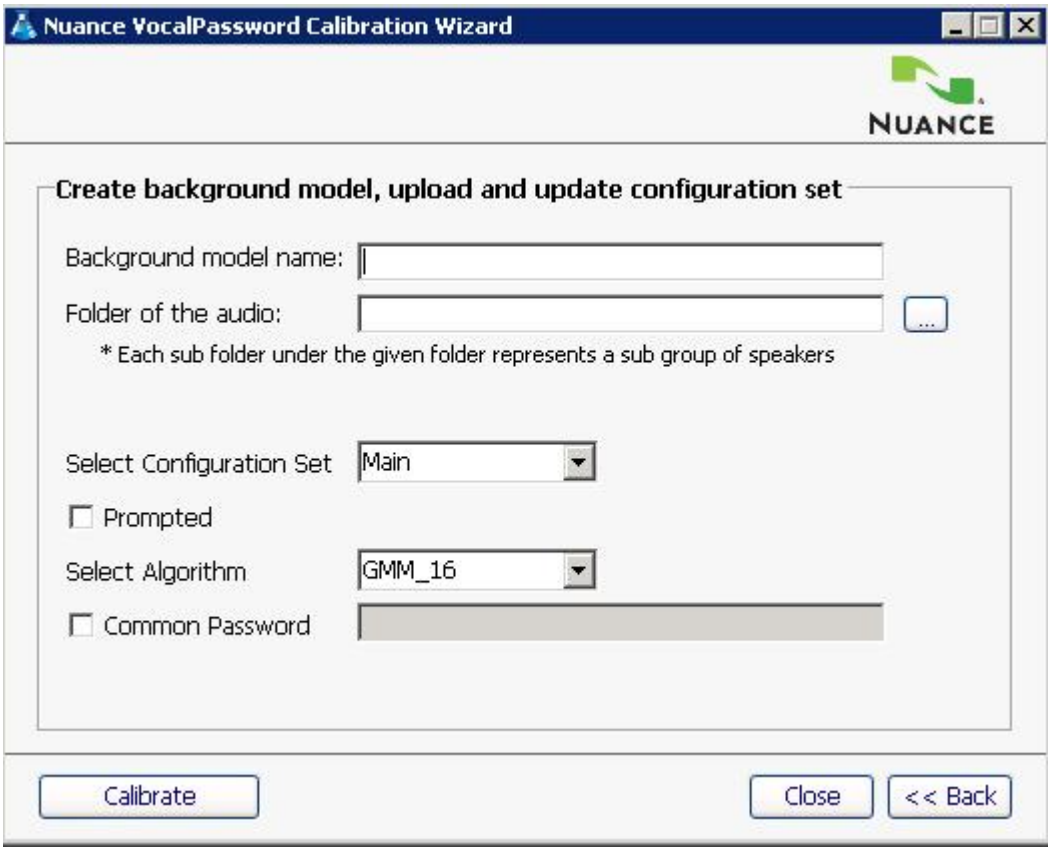
Procedure

1. From the **Start** menu, open **VocalPassword** and select **Calibration Wizard**. The Background Model wizard appears.



The Processing Server name field indicates the VocalPassword processing server.

2. Decide which of the management options is appropriate:
   - **Create Background Model and save to File** - This enables you to save a Background Model to file.
   - **Background Model to file** - This enables you to create a Background Model and load it to the system for use by system users. The loaded model appears in the list of models available for selection when training the user.
   - **Create Background Model, load to the system and modify Configuration Set to use it** - This enables you to create a Background Model, load it to the system, and modify a specific configuration set to use it.
3. Select the relevant option and click **Next**. Options appear for the audio folder and the target file name and path.



4. Enter the path of the folder containing the audio files.
5. Enter the target path and name of the new Background Model.
6. Select the relevant scope (when more than one scope exists) and algorithm from the drop down lists.

   The VXML & PPRO VocalPassword sample applications use the “Default” configuration set, so we should choose **Default**.
7. If you had selected **Create Background Model, load to the system and modify configuration set to use it**, select the relevant configuration set from the drop down list.
8. Click **Calibrate**.

   The progress bar indicates the processing activity of the sample files. The result upon completion is dependent upon the option chosen above:

   - **Create Background Model and save to File** - Background Model is saved to the specified file.
   - **Create Background Model and upload to system** - Background Model is loaded to the system, and from now on appears in the list of Background Models available for speaker training in the Monitor application.
   - **Create Background Model, load to the system and modify configuration set to use it** - Background Model is loaded to the system, and appears from

now on in the list of Background Models available for speaker training in the Monitor application.

Configuring Fraudster Detection

With VocalPassword's Fraudster Detection capability, you can keep track of known fraudsters, when a common pass phrase (for example, 1 2 3 4 5 6 7 8 9) is used. This functionality is used to analyze enrollment/verification audio in real-time and alert the application whenever a known fraudster is accessing the system. The Fraudsters administration application supplies all the tools needed in order to successfully perform fraudster detection. The administrator can manage fraudster and watchlist entities, analyze audio segments marked as suspicious by contact center agents, and compare them to known fraudster voiceprints. An extensive reporting mechanism is available for audit purposes.

To enable or disable fraudsters detection during verification, use the PerformFraudDetectionOnVerify parameter in Platform Admin. In addition, you must set the WatchListName parameter to the name of the watch list with known fraudsters.

About this task

In addition, you must set the WatchListName parameter to the name of the watch list with known fraudsters. Use the following procedure to create a watch list:

Procedure

1. Go to the Fraudsters Web application: `http://<VocalPassword-Server-IP>/VocalPasswordApplications/Apps/GroupSpeakers/GroupSpeakers.aspx?app=Fraudsters`.
2. Open the **Manage Watchlists** tab.
3. Enter the watch list name and click **Create Watchlist**.
4. Add a speaker which are known fraudster.
5. In the list of the speakers, find the speaker, select **Actions** and choose **Clear Test Flag**.

   When this speaker attempts to verify against account of another speaker, the speaker will be indicated as fraudster.

Chapter 3: MPS Developer Application Development

This chapter covers:

1. Nuance VocalPassword MPS Developer Application programming
2. Application functional description
3. Single user life cycle
4. Request-response programming pattern
5. Main module

Nuance VocalPassword MPS Developer Application programming

This chapter describes how to program an MPS Developer application to use Nuance VocalPassword 8.x and implement its features.

The primary focus of the sample application is to illustrate how to implement Nuance VocalPassword features. For information about how to use the MPS Developer tool, refer to the *MPS Developer User Guide*. The sample application and the files that support the application are installed with the Avaya software. See Install and configure MPS Application Processor Node on page 24.

Application functional description

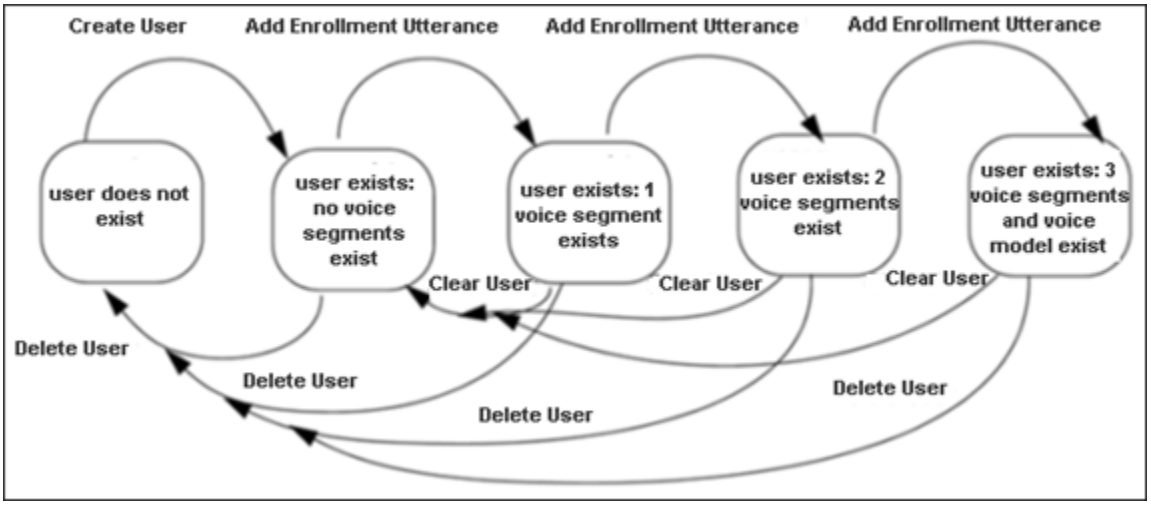
Both the MPS Developer and VoiceXML sample applications follow the same basic design and call flow. The application opens by greeting the caller and prompts for caller's nine digit account number. The VocalPassword server does not require the account number to be nine digits or even to be numeric. The request is simply a function of this sample application. The application then prompts the user to select one of six actions that demonstrate the VocalPassword features of enrollment, single user verification, group verification, unenrollment, and liveness verification. For a description of these features, see Sample application functional description on page 9.

Single user life cycle

The following diagram illustrates the life cycle of a single user's interaction with the database. When the user does not exist, the application prompts the user to enter an account number and create a database entry for that user. Once the user exists, the application prompts the user to speak an utterance, from which the VocalPassword creates a voice segment to use when calculating the voice model. After successfully collecting three voice segments, the VocalPassword creates a voice model.

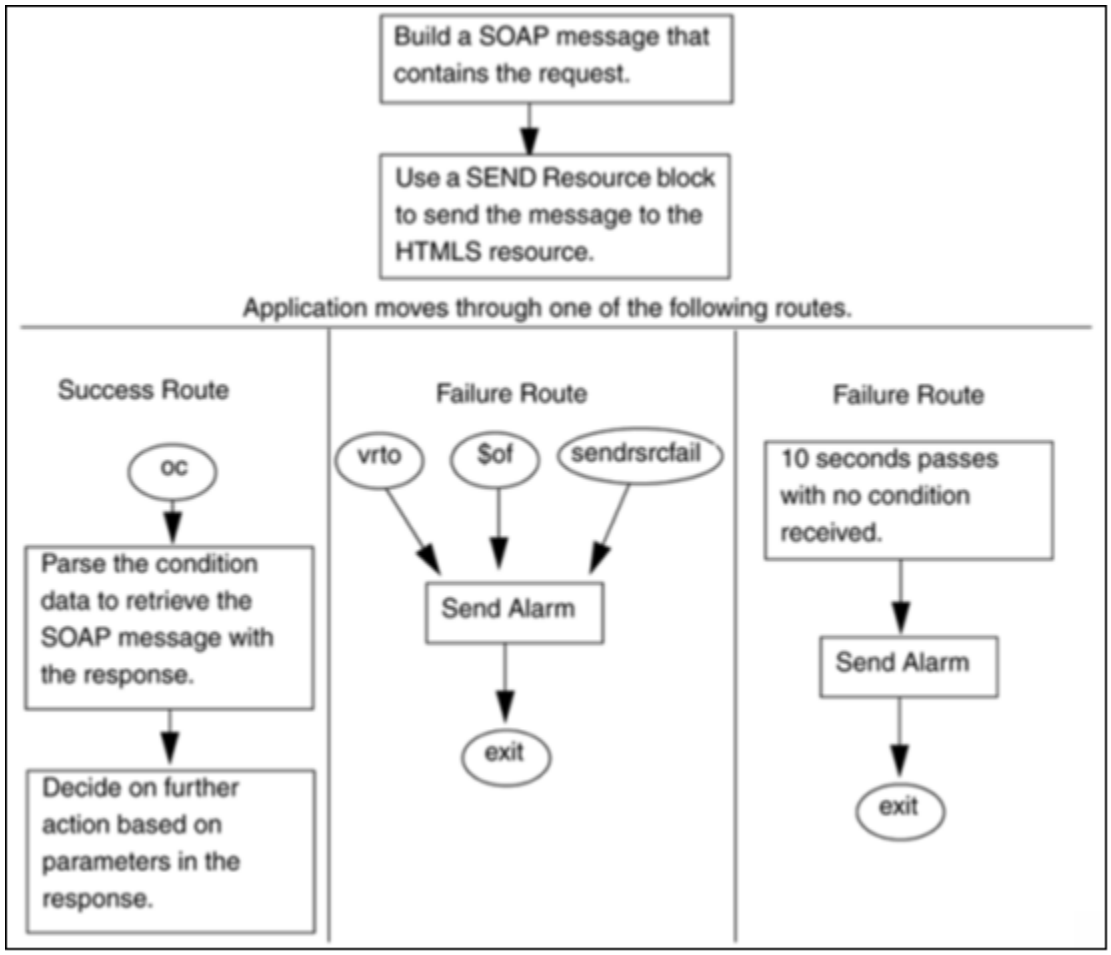
- Each time a clear user operation occurs, the voice template, the enroll segments collection, and the additional data belonging to the voiceprint are deleted. The speaker itself is not deleted.
- Each time a delete user operation occurs, all the speaker's voiceprints, voice templates, and other speaker related data are removed from the system.

In the voice model creation cycle, the caller speaks an utterance, which is saved as a `.wav` file and stored on the MPS Application Processor node. The application encodes the `.wav` file to base64 encoding and sends encoded string to VocalPassword as an API method parameter (i.e. Enroll(), Verify() methods). The VocalPassword gets the encoded `.wav` file, captures statistics, and creates a voice model for the speaker. At this point, the `.wav` file is no longer necessary and the application deletes the `.wav` file.



Request-response programming pattern

The following description explains the pattern of requests sent to the HTMLS resource as a SOAP message and the responses returned back to the application from the VocalPassword in the form of a SOAP message.



Messages are sent to the VocalPassword Server through the HTMLS resource. A SOAP XML request message is constructed and enclosed in a SOAP envelope, which is sent by means of a SEND Resource block to the VocalPassword. One of the following three operations can occur when a message is sent:

- The application receives an oc condition containing the SOAP XML response message as condition data. Further processing is required to parse the SOAP XML response and

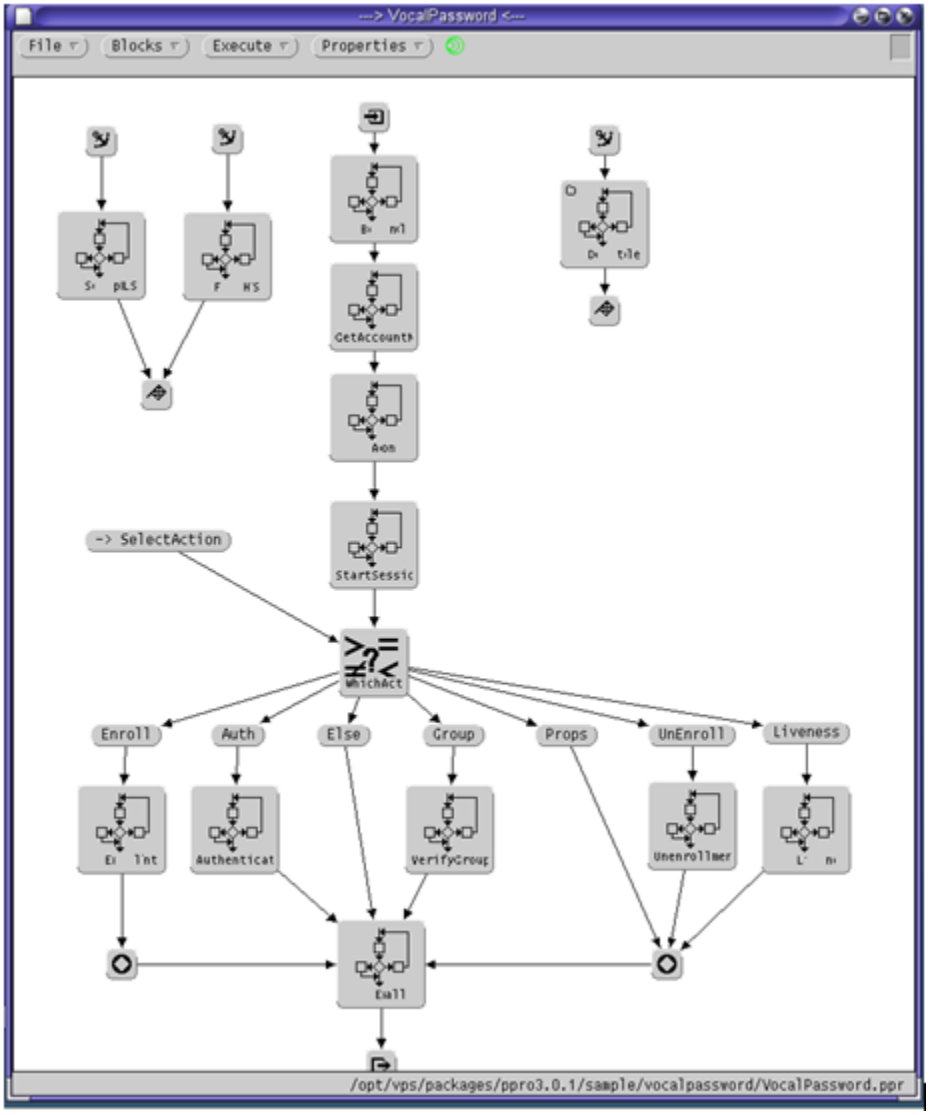
decide on the next course of action, based on parameters received in the response message. See HTMLS parses which SOAP message on page 70.

- The application receives a `vrto`, `$of`, or `sndrsrcfail` condition. Then, the application generates an alarm and terminates the application. See Handle HTMLS conditions on page 70.
- Ten seconds pass with no condition returned to the application. Then, the application generates an alarm and terminates the application.

Main module

The application opens by greeting the caller and asks for the caller's nine digit account number. The application then asks the user to select one of six actions that demonstrate the VocalPassword features of enrollment, single user verification, group verification, unenrollment, and liveness verification. For a description of these features, see Sample application functional description on page 9.

When the main container opens, the construction area window appears with the following building blocks.

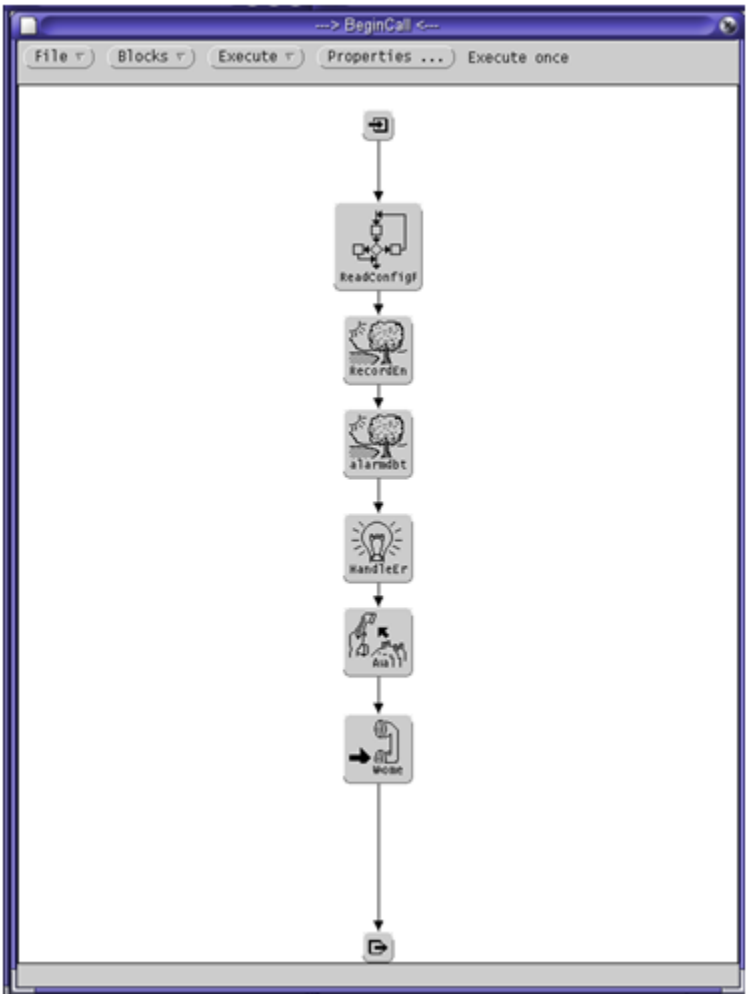


- The **`BeginCall`** Container block reads the `VocalPasswordConfig.xml` file, sets the `ALARMDBTASK` environment variable to VocalPassword, handles error conditions, answers the call and greets the caller. See Begin call on page 46.
- The **`GetAccountNumber`** Container block prompts the caller to enter an account number (that serves as the user ID) and speaks the number back to the caller for confirmation. An alarm is generated in the case of a problem with obtaining an account number and the application moves to an exit path. See Get account number on page 50.
- The **`Action`** Container block prompts the caller to choose one of six paths: enroll a user, verify against a single user, verify against a group of users, delete (unenroll) a user, and perform liveness testing. See Caller chooses action on page 51.
- The **`WhichAction`** Container block switches the application to the path selected by the caller in the **`Action`** Container block. See Which action on page 53.

- The **StartSession** Container block creates a new session with VocalPassword. See StartSession on page 52.
- The **Enrollment** Container block first determines if the caller is enrolled in the database. If the caller is not enrolled, it collects utterances, creates a voice model, and enrolls the caller. The container handles conditions and sets alarms associated with these functions. See Enrollment on page 54.
- The **Authentication** Container block first prompts the caller to enter an account number and determines if the caller is enrolled in the database. If the caller is enrolled, it checks the caller's utterance against the voice model and tells the caller if verification passed or failed. The container handles conditions and sets alarms associated with these functions. See Authenticate or verify a user on page 59.
- The **VerifyGroup** Container block prompts the caller to enter three account numbers to verify against and speaks the results for each account number to the caller. The container handles conditions and sets alarms associated with these functions. See Verify against members of a group on page 61.
- The **Unenrollment** Container block deletes the caller's database entry from the database. The container handles conditions and sets alarms associated with these functions. See Unenrollment on page 64.
- The **Liveness** Container block performs liveness verification. See Liveness verification on page 65.
- The **EndCall** Container block ends the call by closing VocalPassword session and freeing the HTMLS resource and speaking a goodbye message. See End the call on page 68.
- The **SetupHTMLS** Container block acquires HTMLS resource, handles conditions, and sets alarms associated with these functions. See HTMLS resource operations on page 69.
- The **FreeHTMLS** Container block determines if the HTMLS resource is allocated, and if allocated, it frees the resource. See Free the HTMLS resource on page 79.
- The **DeleteFile** Container block removes the user's database entry (record) from the database and generates an alarm if it cannot remove the file. See Delete file on page 80.

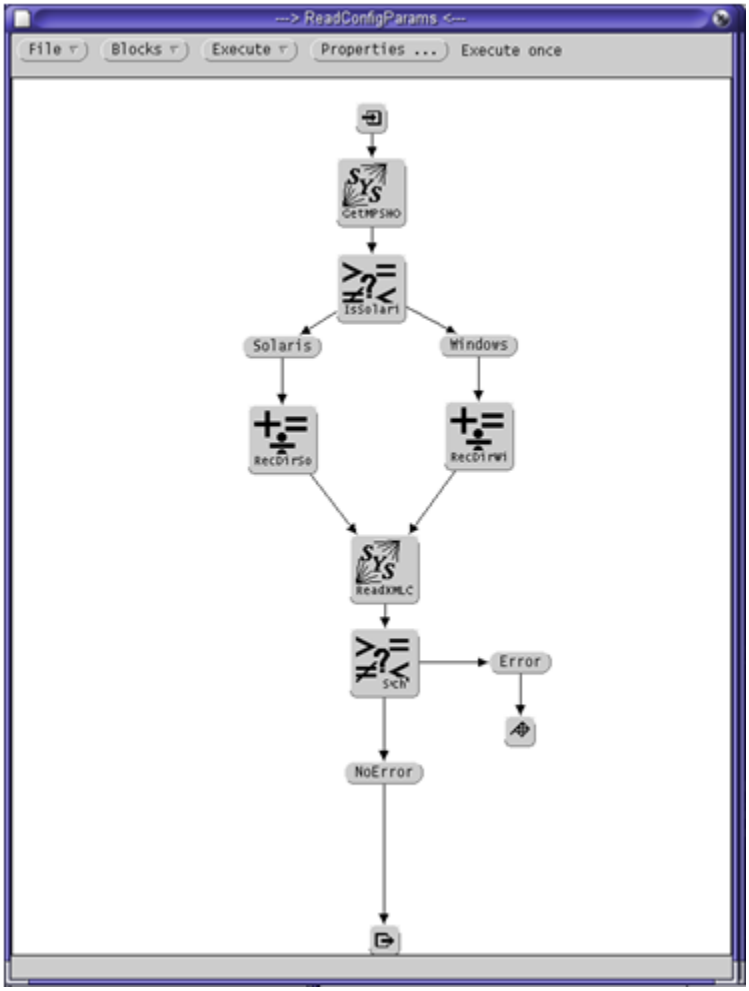
Begin call

The **BeginCall** Container block reads the `VocalPasswordConfig.xml` file, sets the `ALARMDBTASK` environment variable to VocalPassword, handles error conditions, answers the call.



Read configuration parameters

The **`ReadConfigParams`** Container block, determines if the MPS AP is Solaris or Windows and reads configuration file `VocalPasswordConfig.xml` to obtain base information. This file resides in the directory path `$MPSHOME/PERIppro/sample/vocalpassword`. It was configured during the installation process on the MPS Application Processor node. See Installation and configuration required for VocalPassword 8.x on page 24.



Solaris or Windows AP

The **GetMPSHOME** System block obtains the information about whether the MPS AP is Solaris or Windows. The IsSolaris? Switch block determines the path to the configuration files based on the operating system.

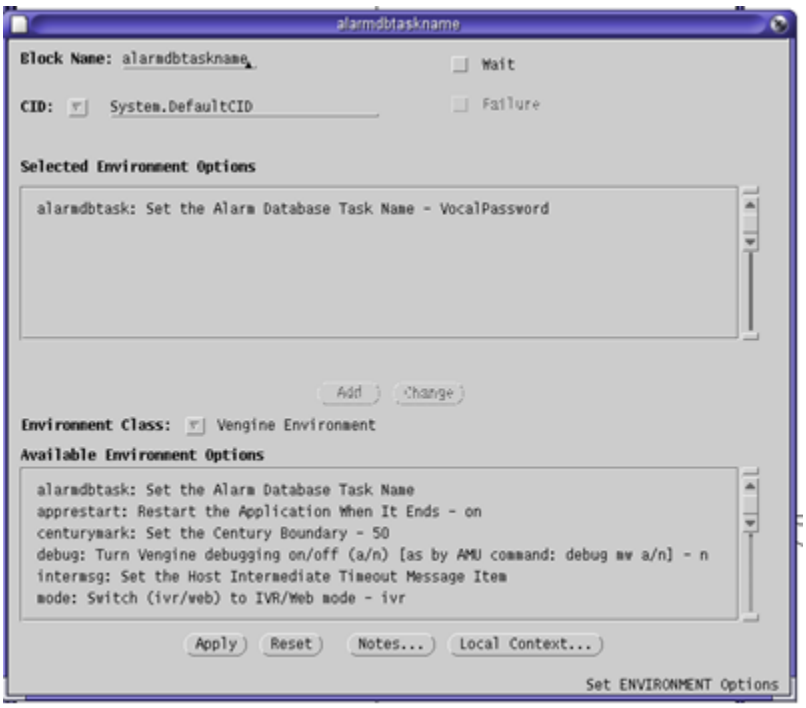
Get VocalPassword parameters

The **ReadXMLConfig** block obtains configuration information from the `VocalPasswordConfig.xml` file. This container block is used for either Solaris or Windows, depending on the operating system.

Set environment variables

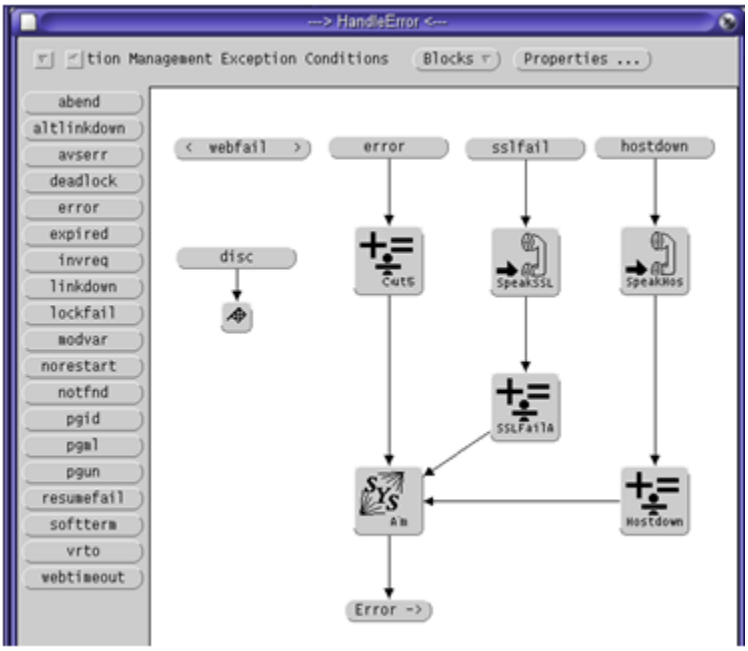
Environment variables that are set affect the application globally. The alarmdbtaskname Environment block sets the ALARMDBTASK environment parameter to VocalPassword. In addition, Avaya recommends setting the following environment parameters.

The RecFirstSil parameter is the duration of front-end silence allowed before the MPS automatically terminates the recording. The RecInterSil parameter is the duration of backend silence allowed before the MPS automatically terminates the recording.



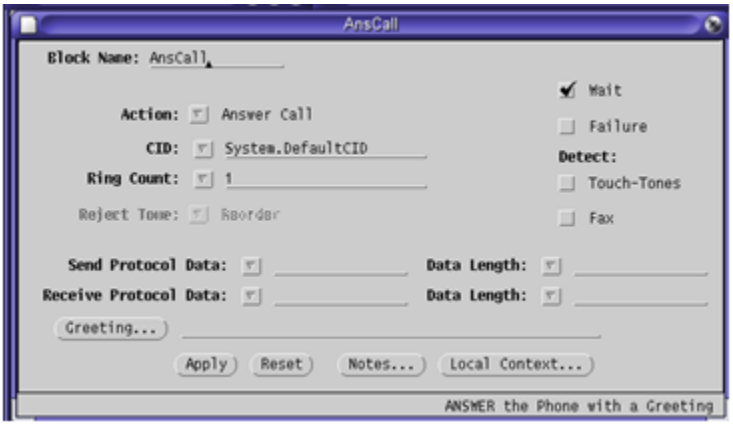
Handle conditions

The **`HandleError`** Condition block handles the conditions that may occur during application execution. These include: the catch-all **`error`** condition, **`disc`**, **`hostdown`** and **`sslfail`**. The application generates an alarm if either a **`hostdown`** **`error`** or **`sslfail`** condition is generated.



Answer the call and greet the caller

The **AnsCall Answer** block answers the call. The **Welcome Speak** block speaks a greeting to the caller.



Get account number

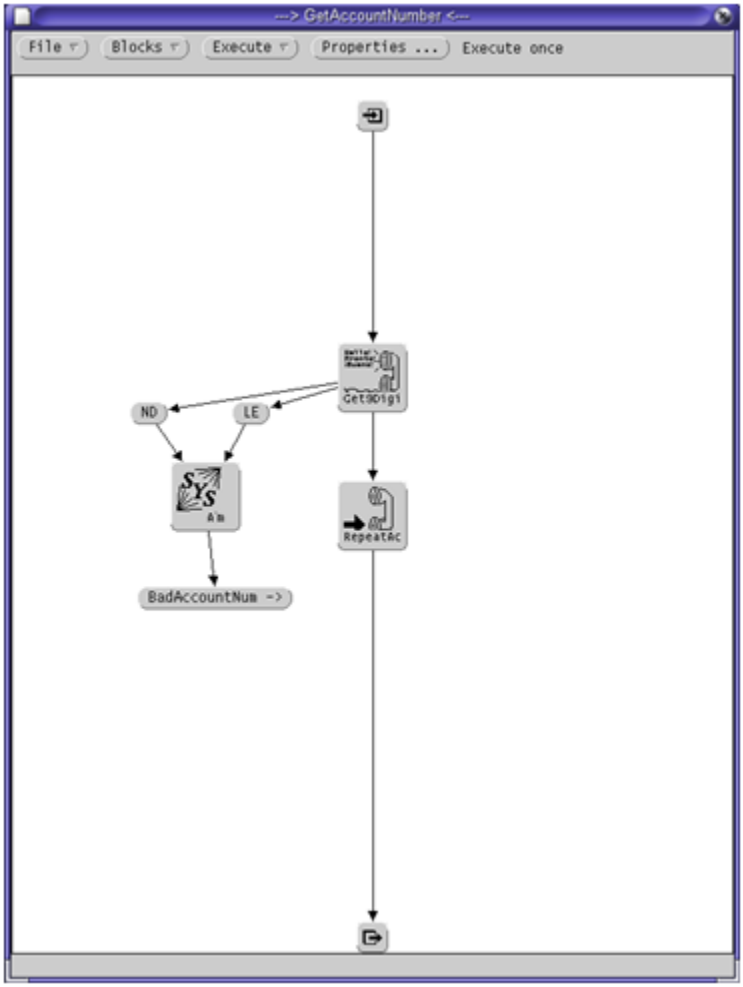
The **GetAccountNumber** Container block prompts the caller to enter a nine digit account number that serves as the user ID and speaks the number back to the caller for confirmation. An alarm is generated in the case of a problem with obtaining an account number and the application moves to an exit path.

The **Get9Digits Read** block expects exactly nine digits and reads the digits into the datacard **AppFolder.AccountNumber**. The Nuance VocalPassword does not require nine digits, or that the value it is numeric. This account number is expected in the case of the sample application only.

The **RepeatAccountNum Speak** block speaks the account number to the caller for confirmation.

The **Alarm System** block generates an application alarm if either no data was detected before the first character timeout or less than nine digits were received before an inter-character timeout. See Set environment variables on page 48.

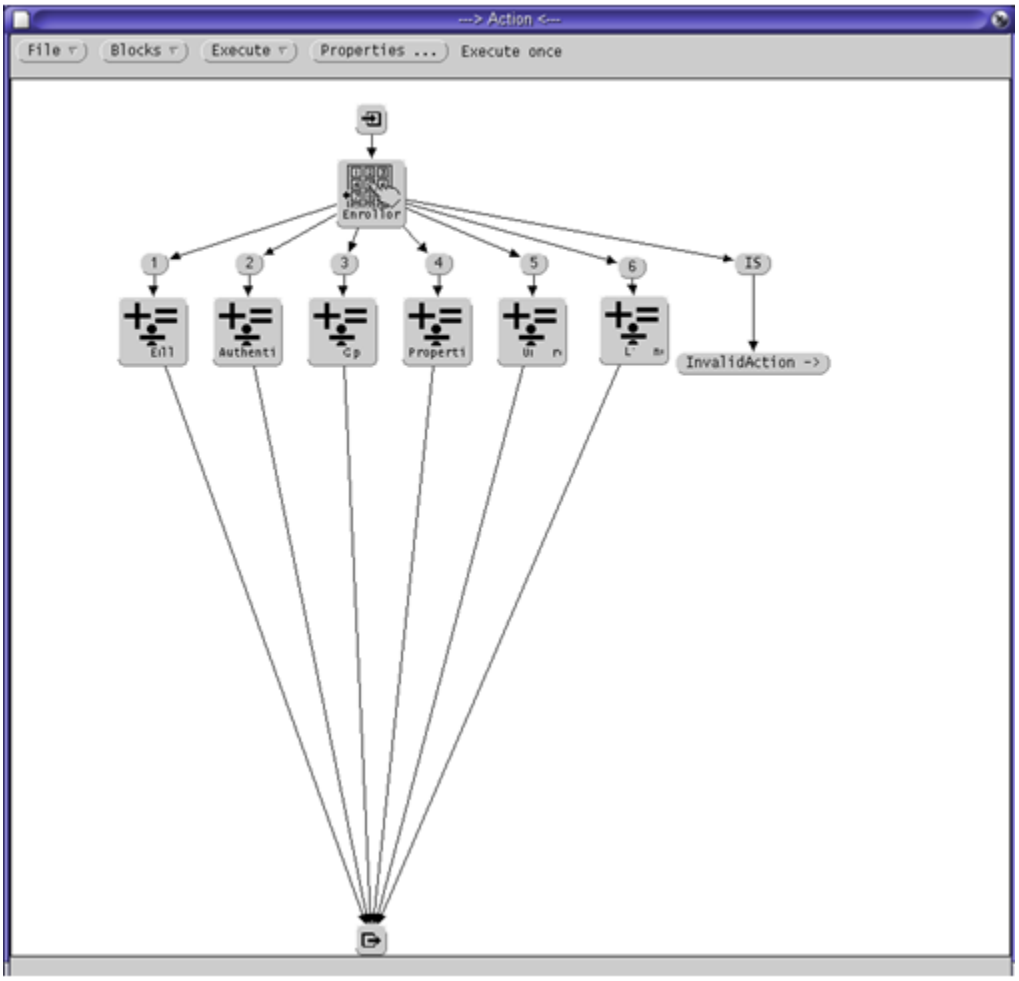
The **BadAccountNum Connector** block moves the application to the clean-up path.



Caller chooses action

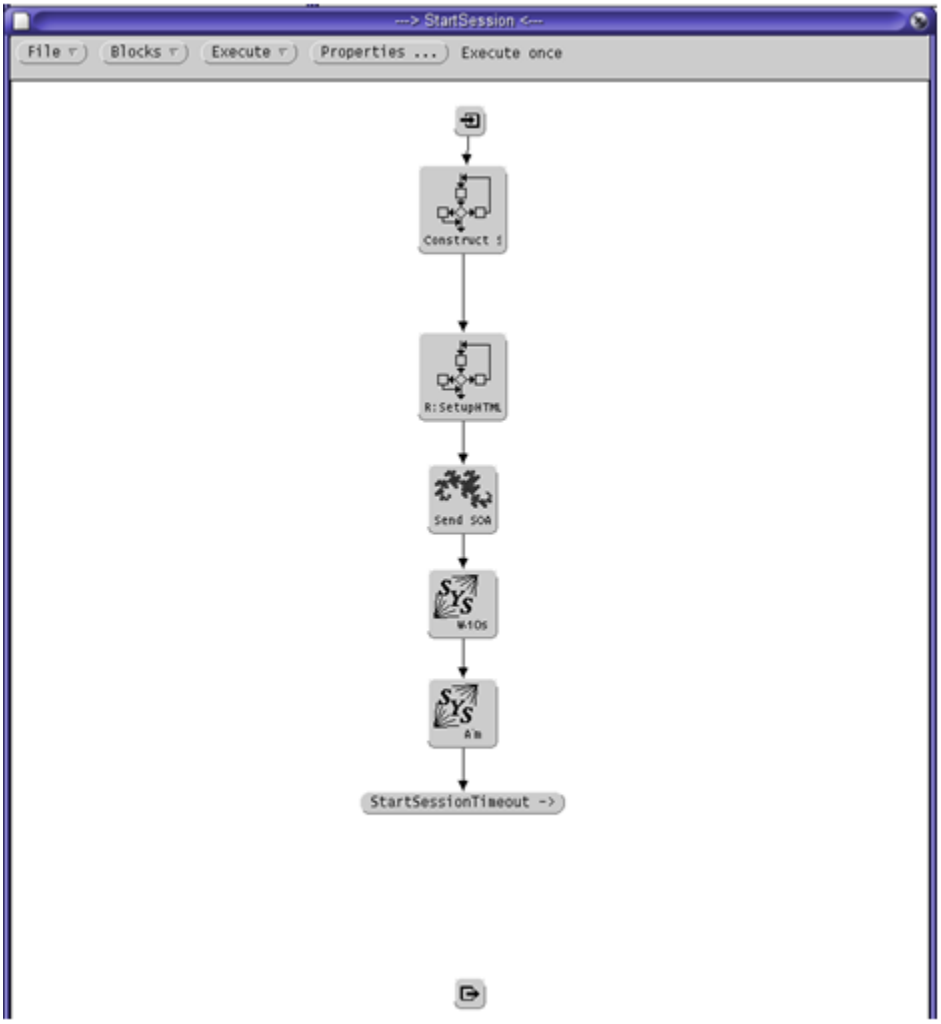
The **`Action`** Container block prompts the caller to choose one of five paths by entering the appropriate touchtone: enroll a user (1), verify against a single user (2), verify against a group of users (3), obtain user properties (4), delete or unenroll a user (5), liveness verification (6).

The **`EnrollorUnenroll`** Select block collects the digit. The **`Enroll`** Compute block assigns the digit to the **`Appfolder.Action datacard`**.



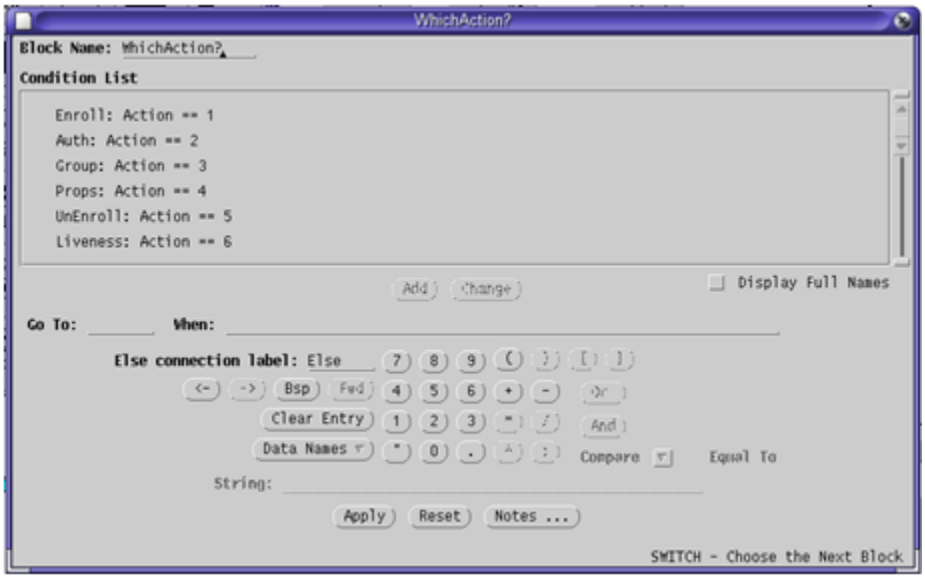
StartSession

The **`StartSession`** Container block creates a new session with VocalPassword. First it constructs SOAP request, then opens HTMLS connection and sends SOAP message to the VocalPassword Server. If HTMLS failed to send the request within 10 seconds, an alarm is generated.



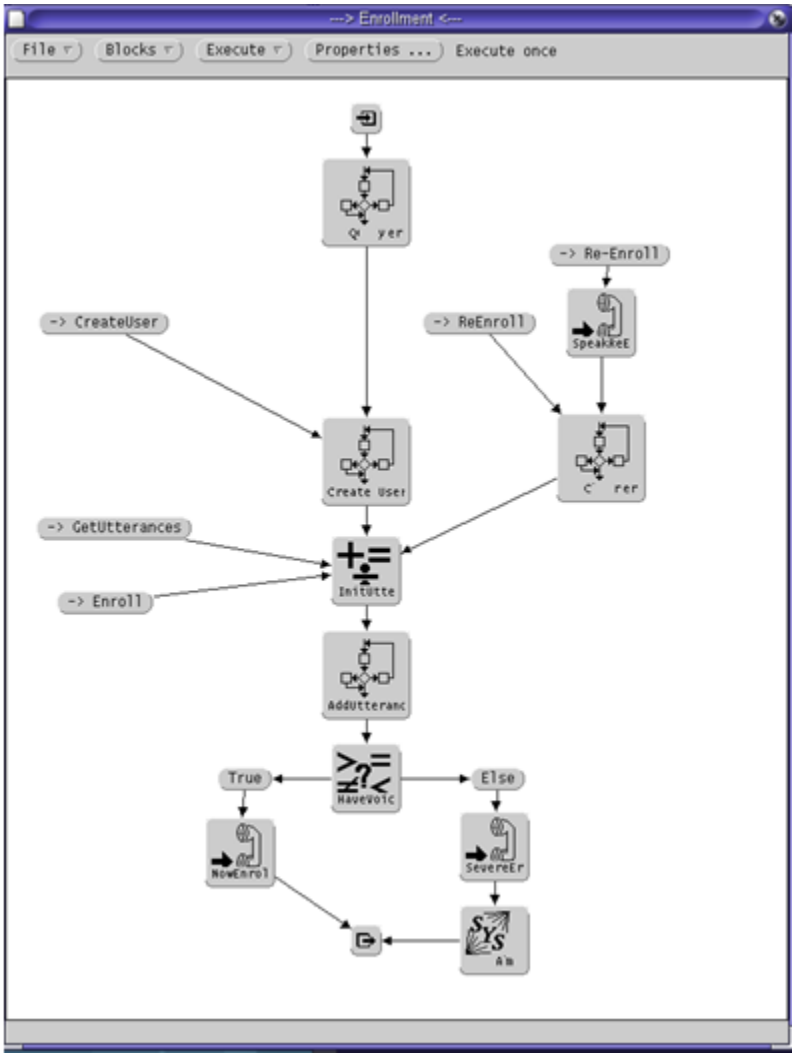
Which action

The WhichAction Switch block switches the application down the path selected by the caller in the Action Container block. See Caller chooses action on page 51.



Enrollment

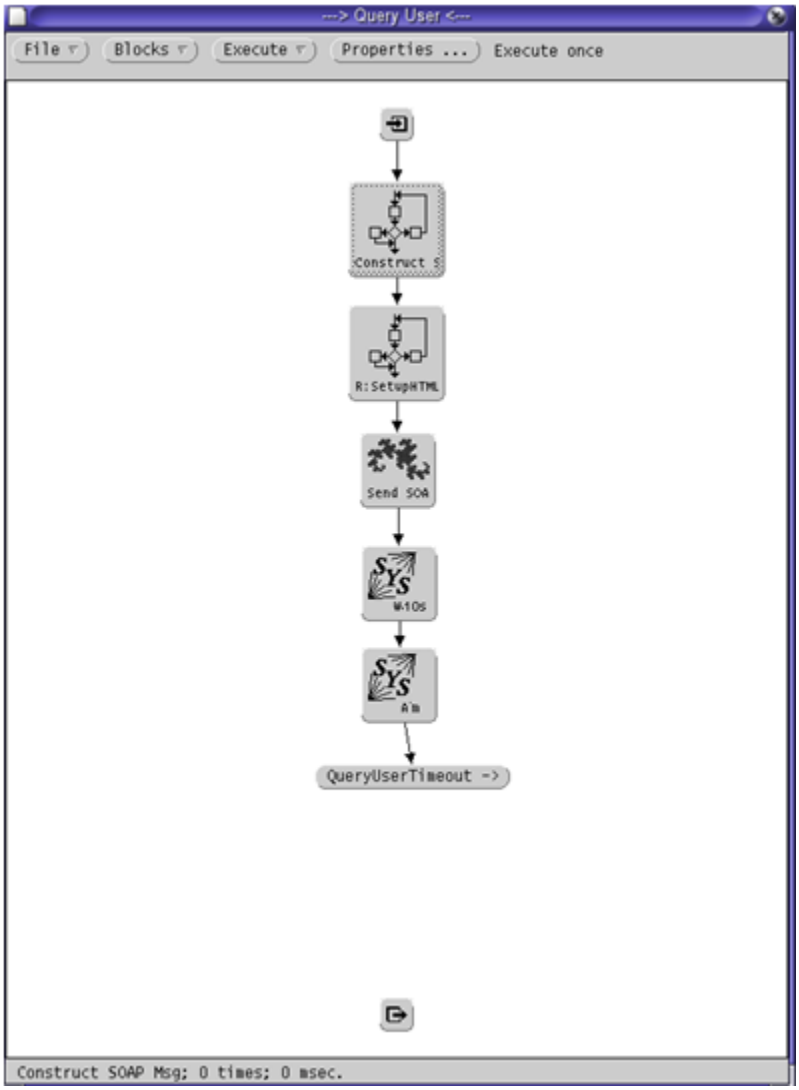
The **`Enrollment`** Container block first determines if the caller is enrolled in the database. If the caller is not enrolled, it collects utterances, creates a voice model, and enrolls the caller. The container handles conditions and sets alarms associated with these functions.



- The application takes the **`CreateUser`** container path if the caller did not previously exist in the database.
- The application takes the **`Re-Enroll`** container path if the caller does exist in the database but a new voice model must be created for the user. For example, it may be that after some period of time, a voice model is considered stale because voices can change. In this case, the application would prompt the user to re-enroll in the database and create a new voice model for this caller.
- The application takes the other **`Re-Enroll`** container path if the user enrolled in standard mode with inconsistent utterances. In this case, the user is asked to restart the enrollment process. In this case, the application would prompt the user to re-enroll in the database and create a new voice model for this caller.
- The application takes the **`GetUtterance`** container path when it is ready to begin adding utterances after having created the user's database entry. The VocalPassword requires one three utterances to create the voice model.

Query the user

The **QueryUser** Container block determines if the caller is enrolled by sending a SOAP message (IsTrained API method) to the VocalPassword. The application uses the **QueryUserTimeout** to link to the **EndCall** container.



The set of blocks that construct and send the SOAP message is used repeatedly in the application. Only the parameters contained in the message are different. This initial set of blocks illustrates the programming pattern that describes how to create and send any SOAP message that goes to the VocalPassword.

- For information about where the application receives responses to these SOAP messages, see HTMLS parses which SOAP message on page 70.
- For information about where the application receives the Query user response to the SOAP message, see Query user on page 76.

The **Alarm** system block is an example of defensive programming. The application only reached this block if the follow-through conditions to one of the other exit paths have not been handled properly.

Construct the SOAP message

The **ConstructSOAPMsg** Container block constructs the SOAP message that is sent to the VocalPassword. This block is used repeatedly in the application to construct a SOAP message.

However, the message that is constructed depends on the context in which the block is used. In this case, the block is designed to construct the Query User message. For information about where the application receives responses to these SOAP messages, see HTMLS parses which SOAP message on page 70.

Set the URL and SOAP message body parameters:

The **SetURL** Compute block sets the VocalPassword Service URL. The **SetParams** Compute block sets the parameters that form the message.

These blocks are repeated in each **ConstructSOAPMsg** Container block, however the actual parameters that are set depends on the context in which block is used. In this case, the block is designed to create the Query User message.

Construct SOAP message header:

The **Header** Compute block creates the message header. This block is repeated in each **ConstructSOAPMsg** Container block, however the actual parameters that are set depend on the context in which block is used. In this case, the block is designed to be the header for the Query User message.

Build SOAP message:

The **Data** Compute block puts the parameters in the SOAP message. This builds the SOAP body from the input parameters.

The **Close** Compute block builds the SOAP message as an XML string and puts it in the **AppFolder.SoapMessage** datacard and closes the SOAP message.

These blocks are repeated in each **ConstructSOAPMsg** Container block, however the actual parameters that are set depends on the context in which blocks are used. In this case, the blocks are designed to create the Query User message.

Send SOAP message

This set of three blocks are used to send the SOAP message and generate an alarm if the message is not sent. This set of blocks is used repeatedly in the application after constructing each SOAP message. See Query the user on page 56.

For information about where the application receives responses to these SOAP messages, see HTMLS parses which SOAP message on page 70.

The **SendSOAPMsg** Resource block sends the SOAP message to the HTMLS resource. If the message was sent, the application receives an oc condition. The application must handle the oc condition. See HTMLS resource operations on page 69.

The **Wait10s** System block waits for 10 seconds to allow the application to receive the oc condition. If the application did not receive an oc condition within the 10 seconds, the **Alarm** System block generates an alarm. See HTMLS resource operations on page 69.

For information about:

- where the oc condition is handled, see Handle HTMLS conditions on page 70.
- where the oc condition is received and the application parses the SOAP message, see HTMLS parses which SOAP message on page 70.
- where the return code is examined to determine the next action, see Process error codes for HTMLS message on page 78.

Clear user

The **ClearUser** Container block is used to delete the voice template, the enroll segments collection, and additional data belonging to the voiceprint from the database entry by sending a SOAP message (**DeleteVoiceprint** API method) to the VocalPassword. The user's account number is not deleted from the database. This occurs when a new voice model is required for an existing user and the caller is asked to re-enroll.

For information about where the application receives the Clear user response to the SOAP message, see Clear user on page 76.

Initialize utterance values

The **InitUtteranceLoop** Compute block sets initial values in the **AppFolder.UtteranceCount** datacard to 0 and in the **AppFolder.HaveVoiceModel** datacard to "false".

The VocalPassword requires three utterances to create the voice model. Thus, this counter is increased until the three utterances are captured and the **AppFolder.HaveVoiceModel** datacard is set to true.

GetEnrollPromptAndSpeak

The **GetEnrollPromptAndSpeak** block contains functionality to ask caller to speak code phrase required for Enroll operation. There are two types of the phrase:

- **Default number sequence 1-9**: In this case MMF item is used to ask user to speak phrase.
- **Text-Prompted mode**: The application receives code phrase from VocalPassword and asks user to speak it using TTS prompt.

Add enrollment utterance

The **AddUtterances** Container block performs the functions associated with recording the utterances required for computing a voice model. Once the voice model is computed, the application tells the caller he is enrolled in the database.

The **AddEnrollUtterance** Container block sends a SOAP message (**Enroll** API method) to the VocalPassword to add the caller's utterance, which will be used to calculate a voice model.

For information about where the application receives the Add Enrollment Utterance response to the SOAP message, see Add enrollment utterance on page 73.

Record utterance--set recording file name

The **RecordUtterance** Container block performs the function of recording and collecting the utterances required to compute a voice model. The **GetRecFileName** Compute block sets the filename of the recording to the format `rec<user id>-<utterance number>.wav`. The variables for the file name are defined in datacards in the `AppFolder`.

Record utterance--request and record utterance

The Record1 Record block speaks a prompt instructing the user to speak the digits `1` through `9` in order. Then, it records caller input into an 8KHz, ULaw `.wav` file. The application sets the conditions under which the recording terminates. These conditions include: the maximum of 10 seconds wait, the silence periods to tolerate set with RecFirstSil and RecInterSil parameters, and if the caller presses a DTMF key. See Set environment variables on page 48

The **Alarm** System block generates an alarm if there is an error. The `SevereError` Speak block speaks a message indicating there is a severe error.

Switch and next utterance

The Switch1 Switch block determines the application flow depending on whether a voice model has been computed.

- If the voice model exists, the application exits the **AddUtterances** Container block.
- If a voice model does not exist, the **NextUtterance** Compute block increments the utterance count by 1. In this case, the application proceeds to record another utterance

See Record utterance--set recording file name on page 59.

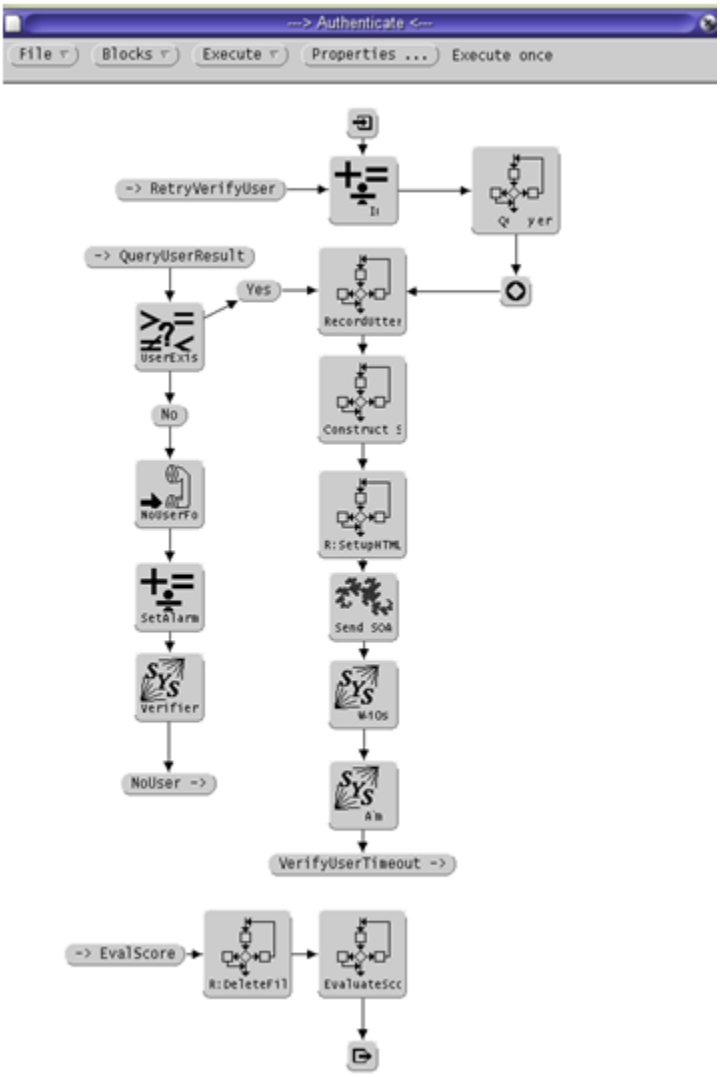
Voice model

Once a `.wav` file is used to create the voice segment, it is no longer needed and the application deletes the `.wav` file. See Delete file on page 80.

The **VoiceModel?** Compute block checks to see if the **AddEnrollUtterance** response indicates there is a voice model.

Authenticate or verify a user

The **Authentication** Container block first prompts the caller to enter an account number and determines if the caller is enrolled in the database. When the caller is enrolled, it checks the caller's utterance against the voice model by sending a SOAP message (`Verify` API method) and tells the caller if verification passed or failed.



In the sample application, the user is prompted to speak an utterance which is converted to a `.wav` file and saved on the MPS Application Processor node. Then, the application encodes the `.wav` file to base64 string and sends a SOAP message containing the base64 encoded `.wav` file to the VocalPassword. The VocalPassword creates a voice model from three utterances. Thus, the utterance collection process occurs three times. Each time an utterance sent to VocalPassword, the application deletes the corresponding `.wav` file from the MPS AP node.

During the authentication process, the caller's utterance is compared to the voice model and the caller is told if they pass or fail. If no voice model exists, the application generates an error.

For information about where the application receives the Authentication response to the SOAP message, see Verify user on page 75

Once a `.wav` file is used to create the voice segment, it is no longer needed and the application deletes the `.wav` file. See Delete file on page 80.

The application evaluates the score and speaks a pass or fail message to the caller. The container handles conditions and sets alarms associated with these functions. The application uses the **`VerifyUserTimeout`** to link to the **`EndCall`** container. See End the call on page 68.

The application moves down the **`RetryVerifyUser`** path to verify the user. The application moves down the **`EvalScore`** path to obtain the score if the user is in enrolled in the database and has a voice model.

Record utterance

The **`RecordUtterance`** Container block performs the function of recording and collecting the utterances required to compute a voice model. This Container block is reused to record utterances at other places during application execution. In this case, it is recording an utterance for the purpose of verification against a single user.

Set record file name and record utterance

The **`GetRecFileName`** Compute block sets the filename of the recording to the format `rec<user id>-<utterance number>.wav`. The variables for the file name are defined in datacards in the `AppFolder`.

The **`Record1`** Record block speaks a prompt instructing the caller to speak the digits `1` through `9` in ascending order. Then, it records caller input into an 8KHz, U-Law `.wav` file. The **`Alarm`** System block generates an alarm if there is an error. The `SevereError` Speak block speaks a message indicating there is a severe error.

Evaluate verification score

The **`EvaluateScore`** Container block checks the caller's verification score.

Verification score and check sign

The **`StrToInt`** Compute block converts the character string score to an integer. The **`Switch1`** Switch block checks the sign of character score. The **`StrToInt`** Compute block converts the character verification score and verification threshold to the integer values. The **`Switch1`** Switch block checks whether the verification score is greater than the verification threshold.

Speak pass or fail

The **`SpeakPass`** Speak block informs the caller that he has passed verification and provides a positive score. The **`SpeakFail`** Speak block informs the caller that he failed verification and provides a negative score.

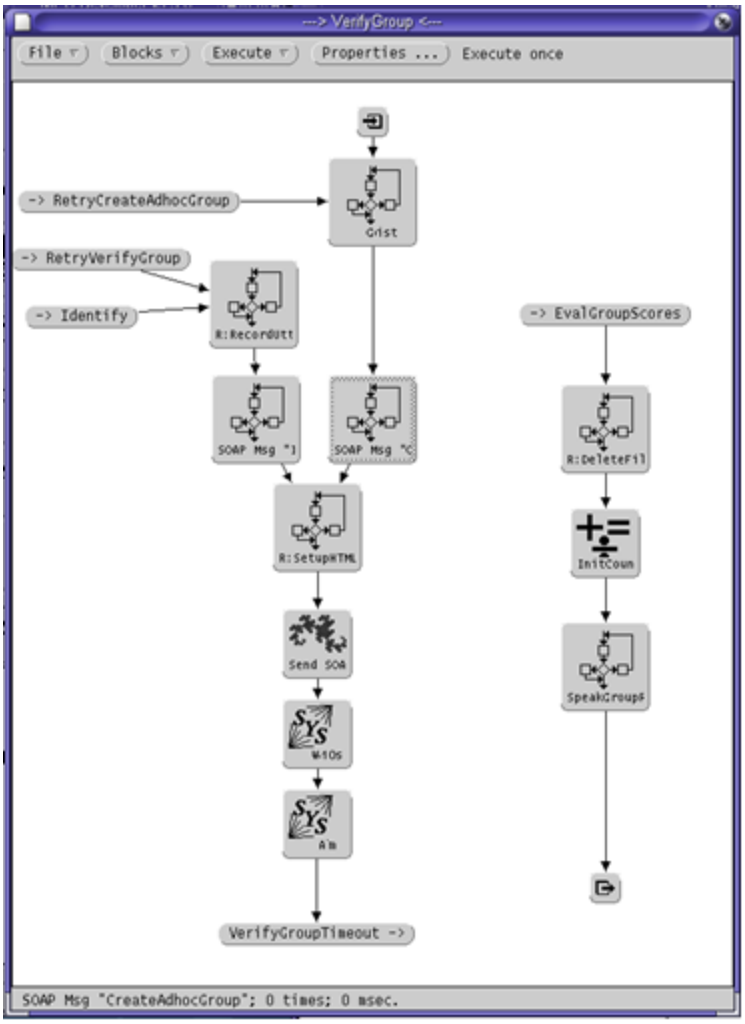
Verify against members of a group

The **`VerifyGroup`** Container block prompts the caller to enter three account numbers to verify against. The application constructs and sends a SOAP message (`CreateAdhocGroup` API

method, `Identify` API method) to the VocalPassword. The application speaks the verification score for each account number to the caller. The application handles conditions and sets alarms associated with these functions. The application moves through the **`EvalGroupScores`** path if all three users exist and have voice models. The application uses the **`VerifyGroupTimeout`** to link to the **`EndCall`** container. See End the call on page 68.

For information about where the application receives the Verify group response to the SOAP message, see Verify user from a group on page 76.

The **`VerifyGroup`** container block first creates an Adhoc group provided user accounts and then Identifies the utterance against the Adhoc group.



Get and record account numbers

The **`GetList`** Container block prompts the caller to enter three nine-digit account numbers to verify against. See Get account numbers on page 63.

The **Alarm** System block generates an application alarm if either no data was detected before the first character timeout or less than nine digits were received before an inter-character timeout.

This **RecordUtterance** Container block used to record the caller's utterances. It is reused to record utterances at other places during application execution. In this case, it is recording an utterance for the purpose of verification against a group of users. The **Alarm** System block generates an alarm if there is an error with the recording. The **SevereError** Speak block speaks a message indicating there is a severe error. See Record utterance on page 61

Get account numbers

The **Acct1**, **Acct2**, **Acct13** and **Read** blocks, prompt the caller to speak the first, second and third nine-digit account numbers against which to verify. This data is stored in the three **AppFolder.AccountList** datacards.

Evaluate caller against group

The caller's utterance is verified against each member of the group. Once a `.wav` file is no longer needed and the application deletes the `.wav` file from the MPS AP node. See Delete file on page 80.

The **InitCount** Compute block initializes the **AppFolder.count** datacard to `1` for looping inside the **SpeakGroupResults** Speak block. See Speak results of group verification on page 63.

Speak results of group verification

The **SpeakGroupResults** container block executes three times. Each time it speaks the caller's score against one of the three account numbers. The caller gets three scores spoken back to them.

Speak group verification message

This set of blocks executes three times. Each time it speaks the caller's score against one of the three account numbers.

The **BeginMsg** Speak block speaks "`your score against user id # <... .> is ...`". The **IntScore** Compute block converts the character string score to an integer. The **Negative** Switch block checks sign of character score and the application follows the positive or negative path depending on the result. The **IntScore** Compute block converts the character verification score and verification threshold to the integer values. The **Pass?** Switch block checks whether the verification score is greater than the verification threshold.

Speak negative group verification message

When the **Negative** Switch block determines that the score is negative, the **Negative** Speak block speaks that negative score to the caller. The **Wait1s** System block pauses the application for one second. The **IncCount** Compute block increment **AppFolder.count** by 1. Ultimately, the scores for all three account numbers are spoken to the caller.

Speak positive group verification message

When the **Negative** Switch block determines that the score is positive, the **Positive** Speak block speaks that positive score to the caller. See Speak group verification message on page 63.

The **Wait1s** System block pauses the application for one second. The **IncCount** Compute block increment **AppFolder.count** by 1. Ultimately, the scores for all three account numbers are spoken to the caller. See Speak negative group verification message on page 63.

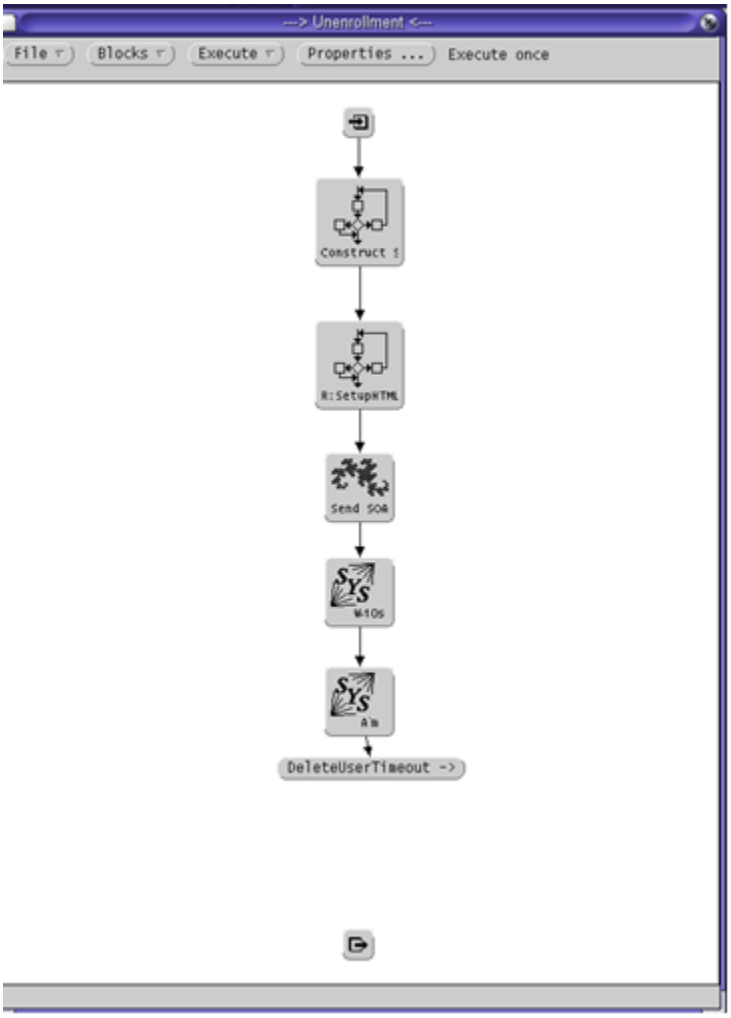
Unenrollment

The **Unenrollment** Container block deletes the caller's record from the database by sending a SOAP message to the VocalPassword that set the Delete User parameters. The container handles conditions and sets alarms associated with these functions. The unenrollment function deletes all the speaker's voiceprints, voice templates, and other speaker related data and the database entry with the account number.

This **ConstructSOAPMsg** Container block is used repeatedly in the application to construct a SOAP message (`DeleteSpeaker` API method).

For information about where the application receives the un-enrollment response to the SOAP message, see Delete user on page 72.

The set of three blocks following the **ConstructSOAPMsg** Container block are used to send the SOAP message and generate an alarm if the message is not sent. This set of blocks is used repeatedly in the application after constructing each SOAP message. See Send SOAP message on page 57. The application uses the **DeleteUserTimeout** Destination Connector to link to the **EndCall** container. See End the call on page 68.



Liveness verification

The **Liveness** Container block performs liveness verification (using `Verify` API method), it first verifies the caller as described in Authenticate or verify a user on page 59, storing the verification utterance in "**AppFolder.Utterances(1)**" datacard. Then, the application generates four random digits and asks the caller to repeat them..

The **Record** Container block performs the function of simultaneous recording and recognition of caller's utterance.

The **GetRecFileName** Compute block sets the filename of the liveness recording to the format `rec<user id>-liveness.wav`. The variable for the filename is defined in "**AppFolder.Utterances(2)**" datacard.

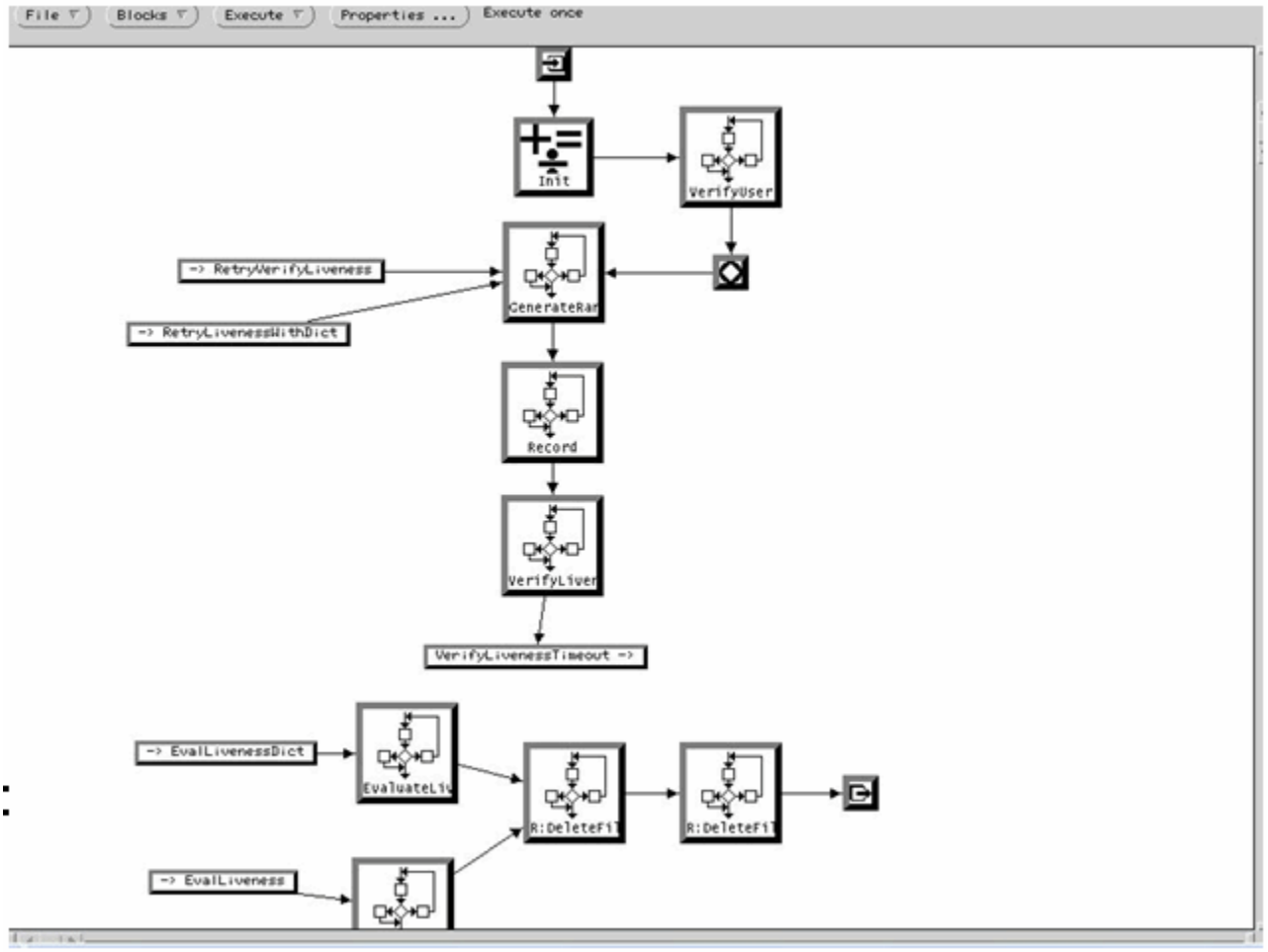
The **RecordStart** block starts the async recording.

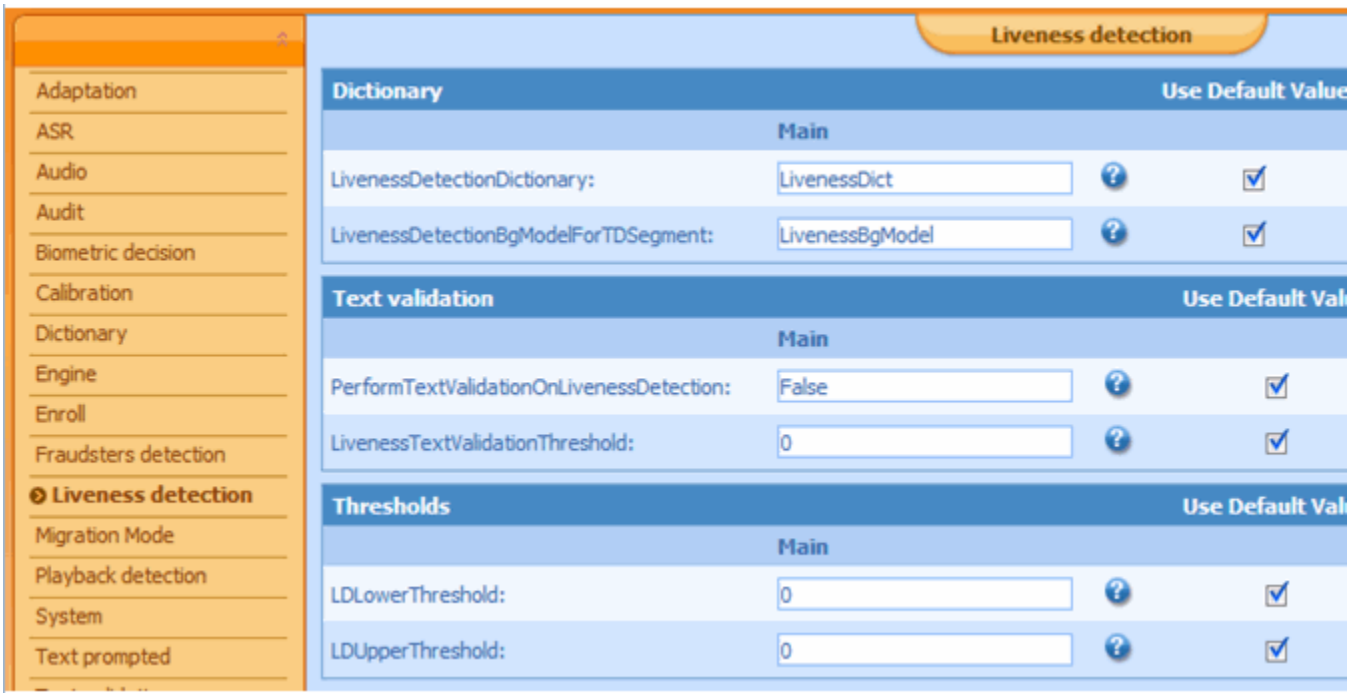
The **Recognize** Container block performs caller's speech recognition. In the sample application Nuance 9 ASR is used. There must be a "digits_001" grammar label (specified in "Resources.lvr-s-label-name" datacard). The result type must be AVS.

If a problem occurs during speech recognition, the application prompts the caller about that, frees the recognition resource and exits. If recognition was successful, the application proceeds to **RecordEnd** block that stops the async recording.

As soon as caller's utterance recognition is performed, the **VerifyLiveness** Container block is executed. It sends the Verify SOAP request to the VocalPassword Server.

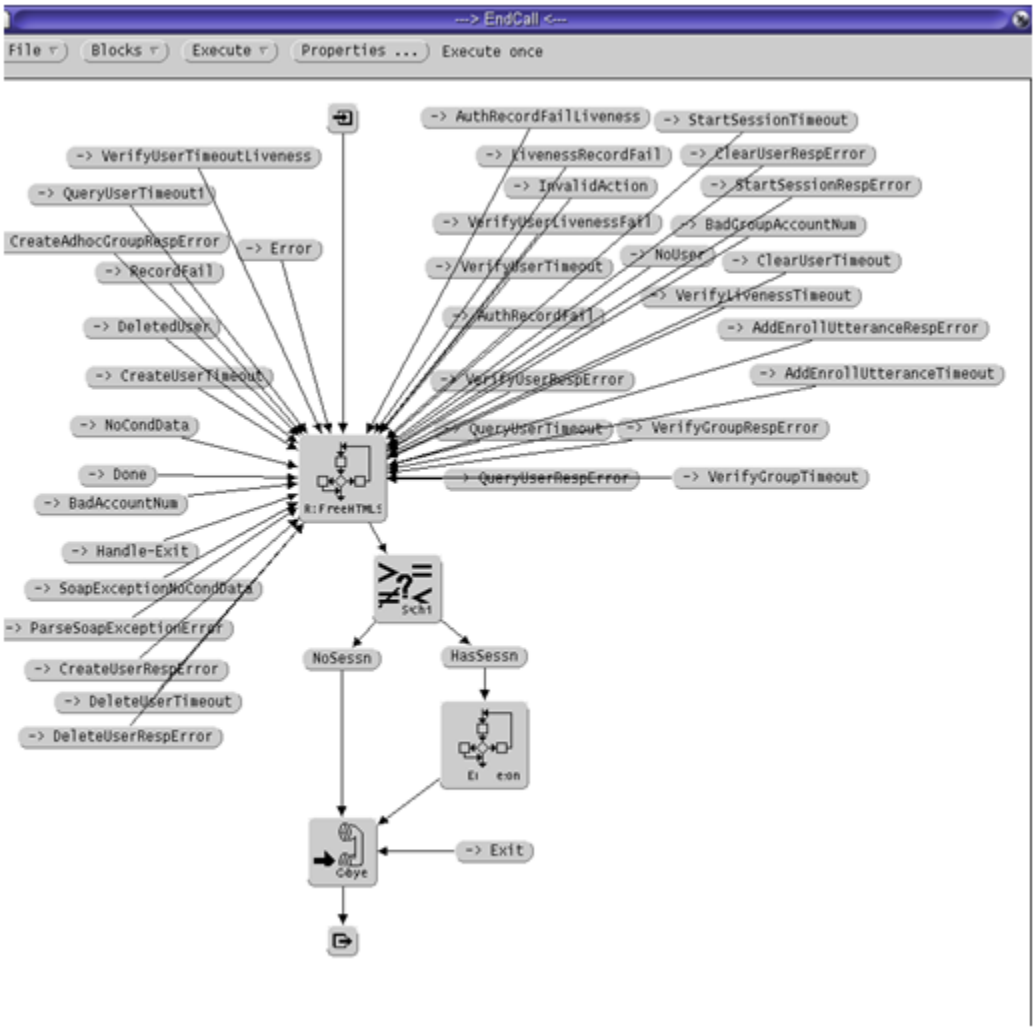
The **EvaluateLivenessScore** Container block evaluates whether the score returned by the VocalPassword server is a positive result and the digits spoken are the ones the caller was asked to speak. The liveness verification result is pass if both of these conditions are met. If any of these is not true, an error occurs and application moves to an exit path.


File
Blocks
Execute
Properties ...
Execute once
Init
VerifyUser
-> RetryVerifyLiveness
-> RetryLivenessWithDict
Record
-> EvalLivenessDict
-> EvalLiveness
R:DeleteFi
R:DeleteFi


Liveness detection
Adaptation
ASR
Audio
Audit
Biometric decision
Calibration
Dictionary
Engine
Enroll
Fraudsters detection
Liveness detection
Migration Mode
Playback detection
System
Text prompted
Dictionary
Use Default Value
Main
LivenessDetectionDictionary:
LivenessDict
LivenessDetectionBgModelForTDSegment:
LivenessBgModel
Text validation
Use Default Val
Main
PerformTextValidationOnLivenessDetection:
False
LivenessTextValidationThreshold:
0
Thresholds
Use Default Val
Main
LDLowerThreshold:
0
LDUpperThreshold:
0

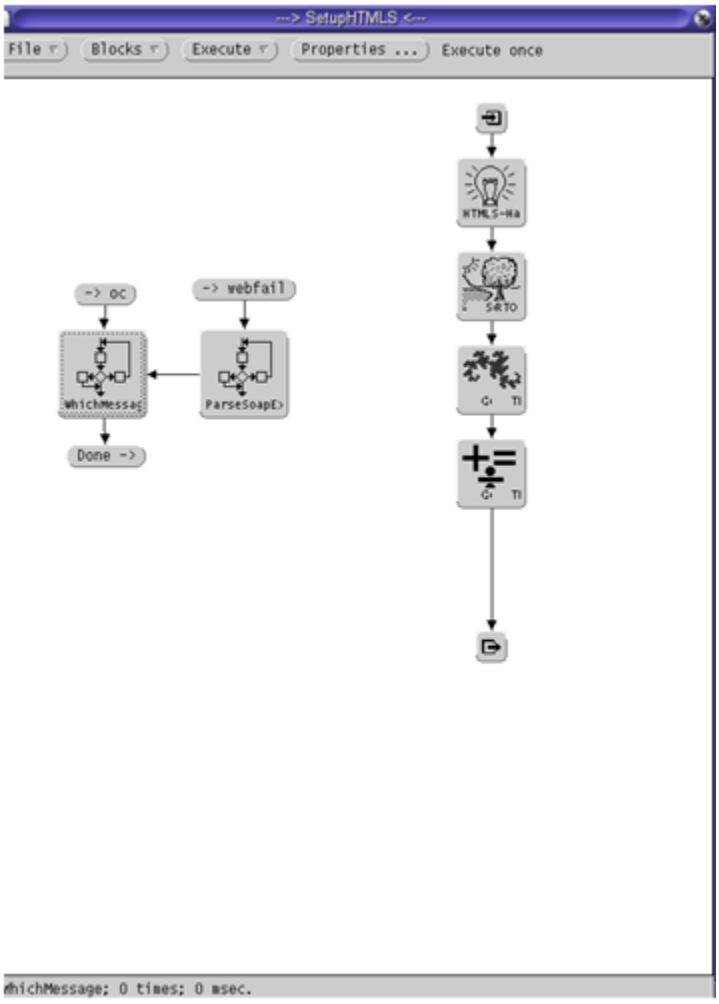
End the call

The **`EndCall`** Container block ends the call by sending EndSession SOAP request to VocalPassword (it session is still open), freeing the HTMLS resource and speaking a goodbye message. The **`Goodbye`** Speak block speaks the message to the caller. See Free the HTMLS resource on page 79.



HTMLS resource operations

The **SetupHTMLS** Container block performs the functions associated with following through with the SOAP requests that are sent to the HTMLS resource. This container block receives and interprets the SOAP messages, then directs the application down the appropriate path.



The actions can be as follows:

- create the user (enter the user's account number and begin recording an utterance)
- determine if caller has an existing voice model
- enroll the user if there is no existing voice model (create a voice model)
- record the next utterance when developing the voice model
- verify the user against a voice model

- verify the user against an each user in a group
- delete the user (delete the user's account number)
- verify the user against a liveness utterance

The container handles conditions and sets alarms associated with these functions.

Handle HTMLS conditions

The **HTMLS-Handler** Handle Condition block handles the conditions associated with satisfying the SOAP requests.

When the application sends a SOAP request, it expects to receive an oc (output complete) condition, which indicates that the HTMLS resource received the message successfully. The oc Source Connector handles completed SOAP requests and the application moves down the path set with the oc Destination Connector to parse the SOAP message. See HTMLS parses which SOAP message on page 70.

The **webfail** Source Connector handles SOAP exceptions.

The **getrsrcfail** Source Connector handles the condition when HTMLS resource allocation fails. The **Alarm-getrsrcfail** System block generates an application alarm indicating that the HTMLS resource is not allocated to the application. See Set HTMLS condition parameters on page 70.

The **$of** Source Connector handles the condition when there is a failure sending the SOAP message to the HTMLS resource. The **Alarm-of** System block generates an application alarm indicating that the SOAP message was not sent.

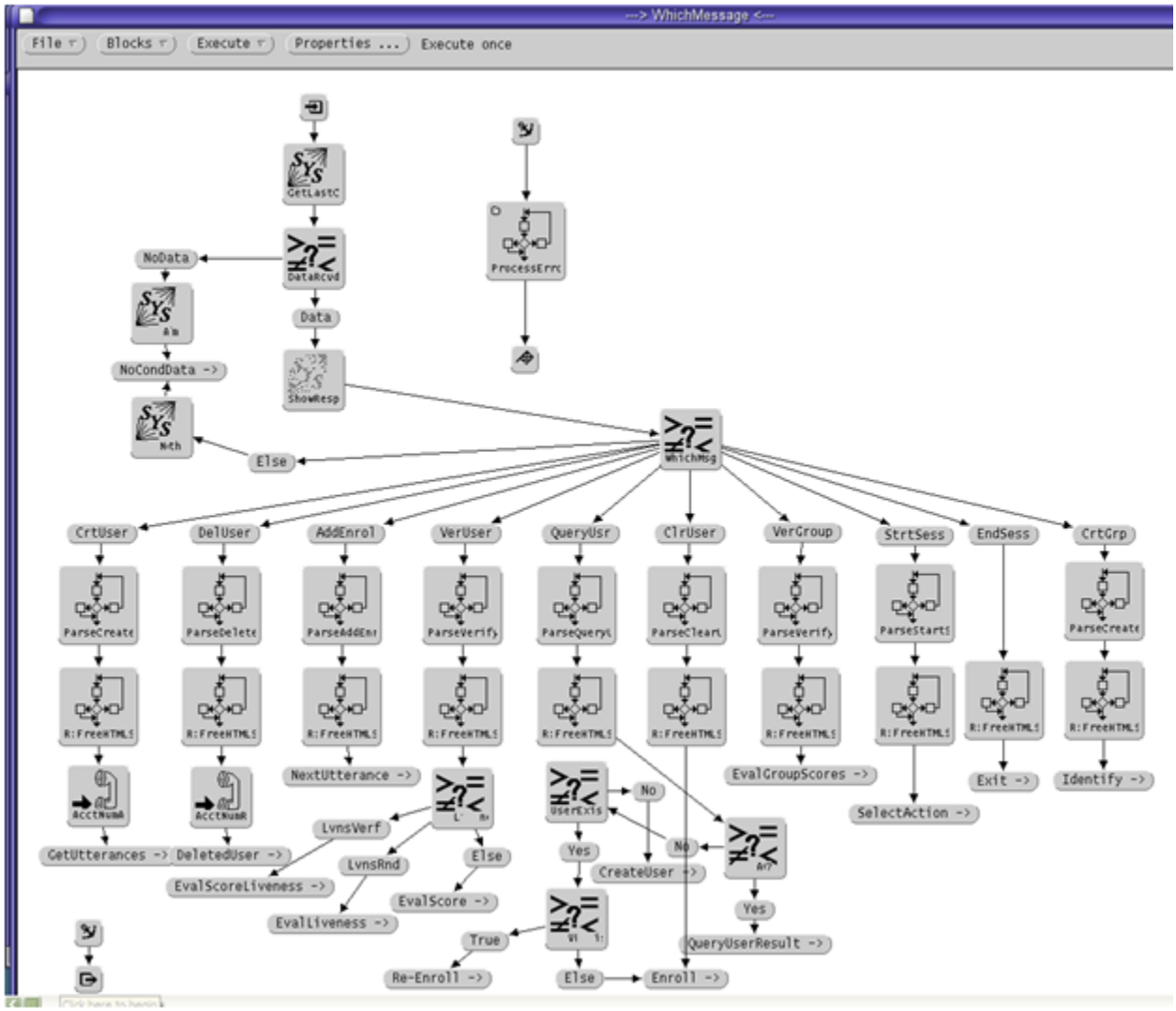
The **vrto** Source Connector handles the condition that occurs when the application fails to get the HTMLS resource within time allowed by environment parameter vpsrcvtime (default 10 seconds). The **Alarm-vrto** System block generates an application alarm indicating the failure to get a resource in configured time period. See Handle HTMLS conditions on page 70. The application uses the **Handle-Exit** Destination Connector to link to the **EndCall** container. See End the call on page 68.

Set HTMLS condition parameters

The **SetVRTO** Environment block set the environment property vpsrcvtime to 90 seconds. If this parameter is not set, the value defaults to 10 seconds. The **GetHTMLS** Resource block, attempts to allocate the HTMLS resource. The **GotHTMLS** Compute block determines if allocation was successful (`true` or `false`). The application handles conditions when there is a failure associated with these blocks. See Handle HTMLS conditions on page 70.

HTMLS parses which SOAP message

The **WhichMessage** Container block determines which of the SOAP messages to parse. The application follows down the path of that message. The application only goes through the **WhichMessage** Container if it received an oc condition after sending a SOAP message. The oc condition indicates that HTMLS resource received the SOAP message. See Handle HTMLS conditions on page 70.



Get last condition

The **GetLastCond** System block uses the call function **last-condition** to retrieve the last condition received by the application and the condition data associated with it. The **DataRcvd?** Switch block determines if condition data exists.

- If there is no data, the application generates and alarm and ends the call. See No data about which SOAP message on page 71.
- If there is data, the application moves down the data path to determine which SOAP message to parse. See Data available about which SOAP message on page 72.

No data about which SOAP message

When the **DataRcvd?** Switch detects that there is no condition data about which SOAP message to parse, the **Alarm** System block generates an application alarm and the application moves down the **NoConditionData** path.

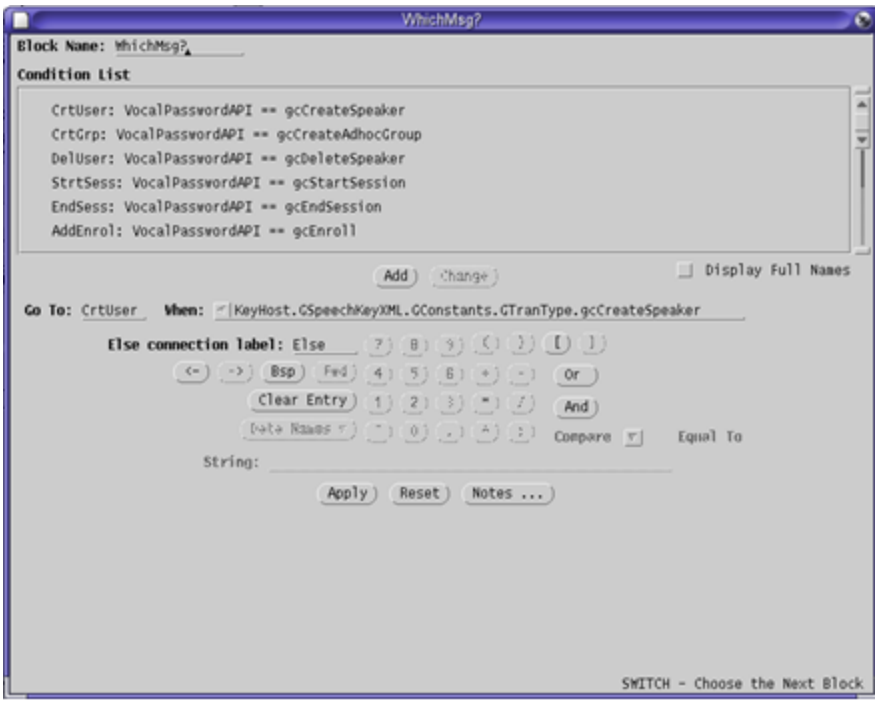
The application takes the route of the **NoPath** System block, if no SOAP message was received by the HTMLS resource. However, this case should never occur as the only way for

the application to get to this set of blocks is if the application received an `oc` condition after sending the SOAP message to the HTMLS resource.

Data available about which SOAP message

When the **`DataRcvd?`** Switch detects that condition data specifies which SOAP message to parse, the application moves down that path. The **`WhichMsg:`** Switch block switches to the path determined by the SOAP message.

In this case the application moves down one of the following paths: create a user, delete a user, add enrollment utterances, verify a user, query user parameters, clear a user, verify the user against a group, start VocalPassword session, end VocalPassword session, and liveness verification. See HTMLS parses which SOAP message on page 70.



Delete user

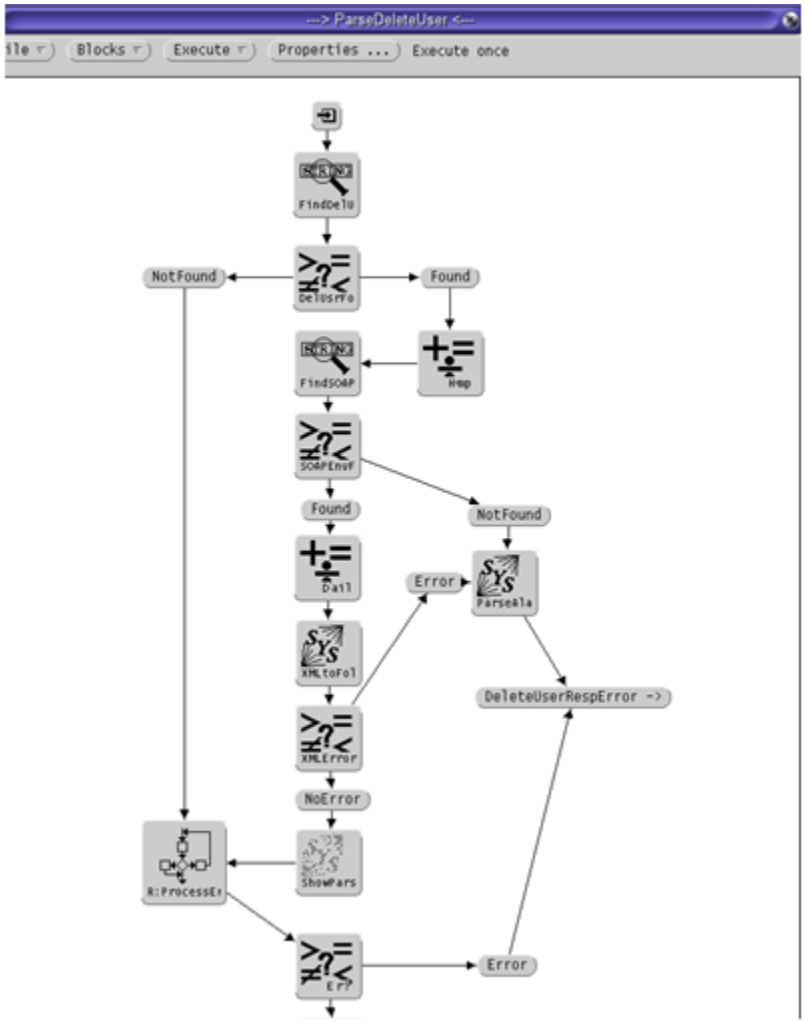
The application moves down the delete user path based on the condition data returned to the resource about which SOAP message to parse.

For information about where the application sends the SOAP delete user message, see Unenrollment on page 11.

The **`ParseDeleteUser`** Container block performs the functions associated with parsing the SOAP message to delete a user. Once the user is deleted, the application links to the

`FreeHTMLS` Condition block to free the HTMLS resource and end the call. See Free the HTMLS resource on page 79.

The **`AcctNumRemoved`** Speak block informs caller that his account number has removed from database. The **`DeleteUser`** Destination Container block links to end the call. See Enrollment on page 54.



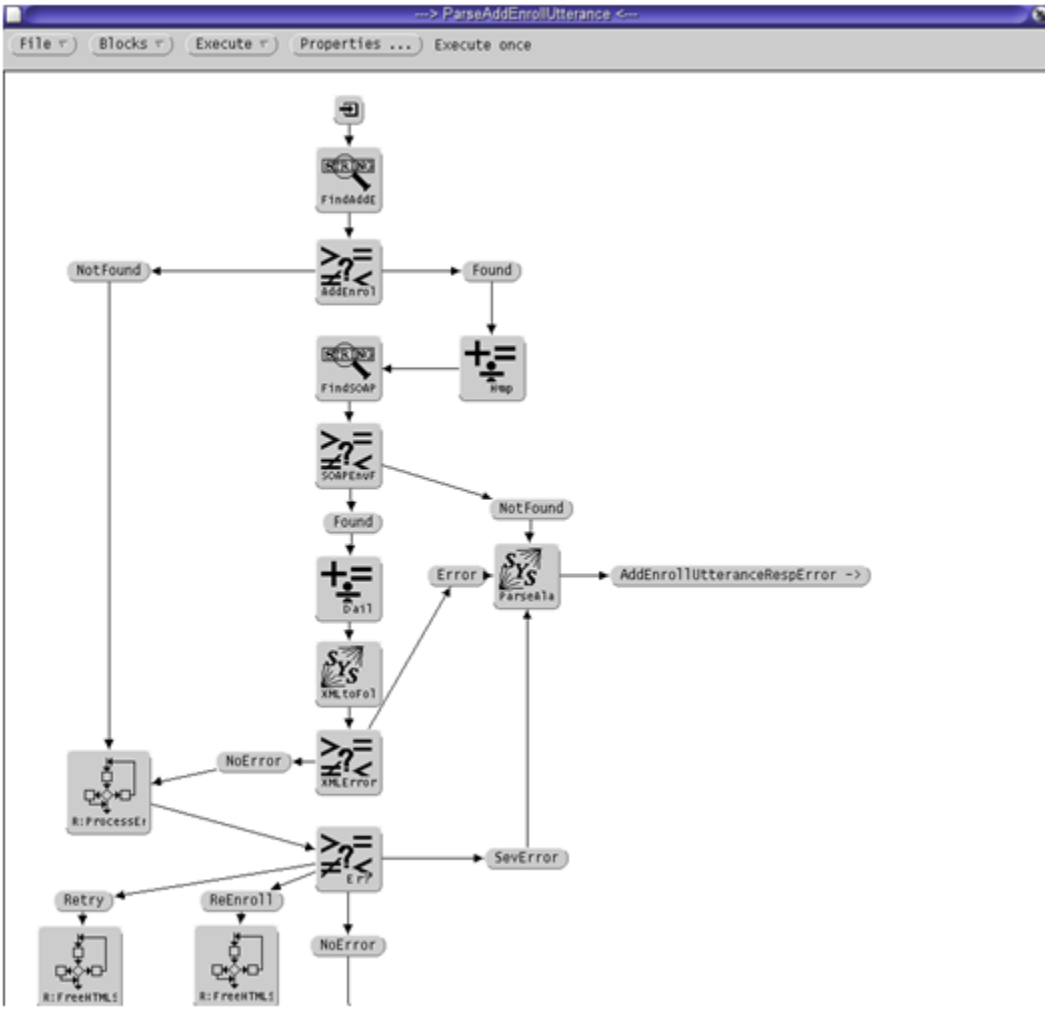
Add enrollment utterance

The application moves down the add enrollment utterance path based on the condition data returned to the resource about which SOAP message to parse. The SOAP message passed the base64 encoded `.wav` file that resides on the MPS to the VocalPassword.

For information about where the application sends the SOAP Enrollment message, see Add enrollment utterance on page 58.

The **ParseAddEnrollUtterance** Container block performs the functions associated with parsing the SOAP message to record and collect user utterances. Once the utterances required for the voice model are collected, the application links to the **FreeHTMLS** Condition block to free the HTMLS resource and end the call. See Free the HTMLS resource on page 79.

The **NextUtterance** Destination Container block links to end the call. The **ReEnroll** Destination Container returns to the Enrollment container to clear the user voice model and re-enroll the user. See Enrollment on page 54.

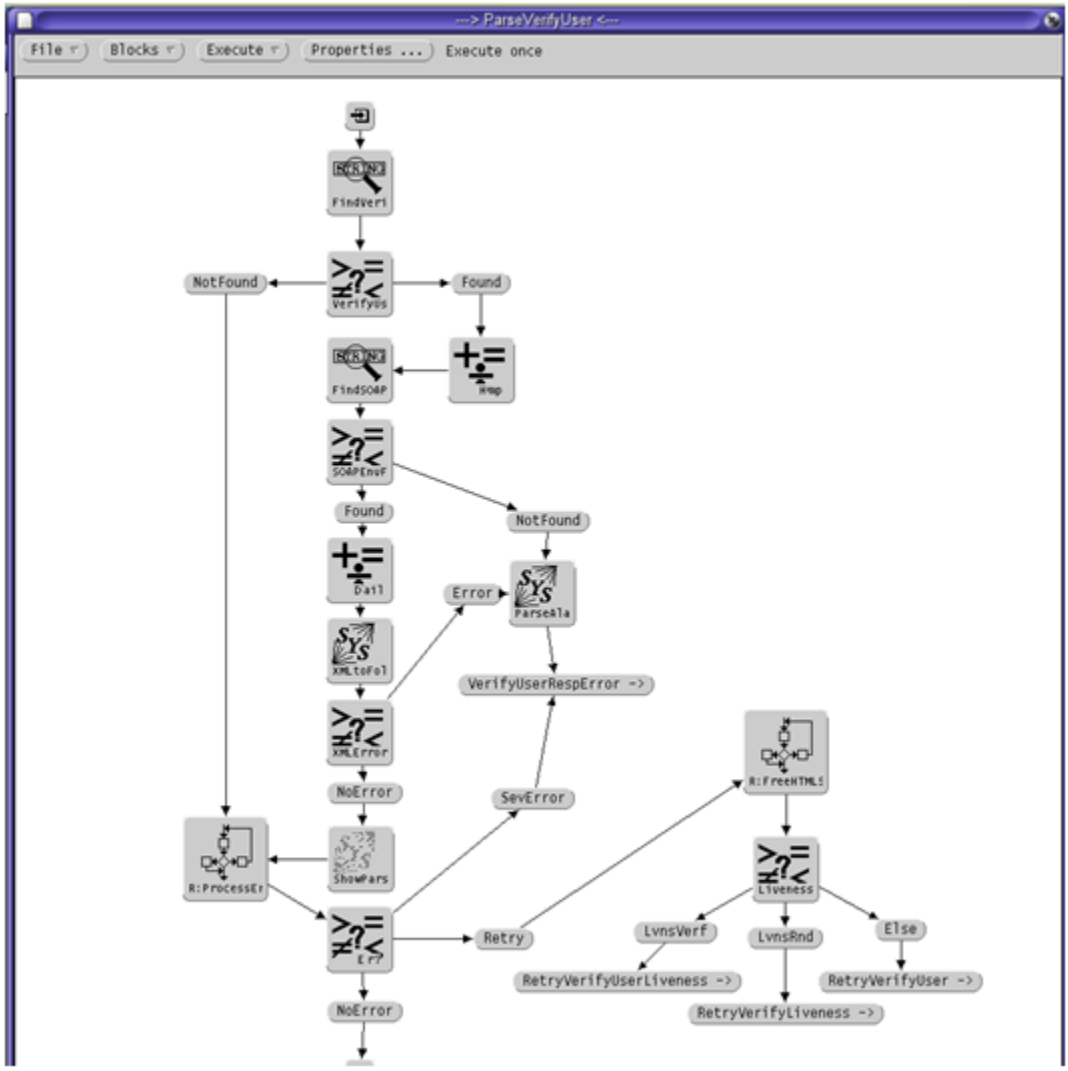


Verify user

The application moves down the verify user path based on the condition data returned to the resource about which SOAP message to parse.

Once the result of verification is obtained, the application links to the **`FreeHTMLS`** Condition block to free the HTMLS resource and end the call. See Free the HTMLS resource on page 79.

The **`EvalScore`** Destination Container block links to the **`Authentication`** Container to speak the verification result. The **`RetryVerifyUser`** Destination Container block returns to the **`Authentication`** Container to retry the verification utterance process. See Authenticate or verify a user on page 59.



Query user

The application moves down the query user path based on the condition data returned to the resource about which SOAP message to parse.

The **ParseQueryUser** Container block performs the functions associated with parsing the SOAP message to check if the user is trained (has a voice model). Once the properties are obtained, the application links to the **FreeHTMLS** Condition block to free the HTMLS resource and end the call. See Free the HTMLS resource on page 79.

The **UserExists** Switch block checks if user exists. The **CreateUser** container moves to the **CreateUser** container inside the **Enrollment** container if the user does not exist.

The **VMExists** Switch block checks if a voice model exists. The **Re-Enroll** container, if the user does have a voice model, moves to the **ClearUser** container inside the **Enrollment** container. The application moves to the **Enrollment** container to proceed with enrollment if the user does not have a voice model. See Enrollment on page 54.

Clear user

The application moves down the clear user path based on the condition data returned to the resource about which SOAP message to parse.

The **ParseClearUser** Container block performs the functions associated with parsing the SOAP message to remove the voice template, the enroll segments collection, and additional data belonging to the voiceprint from the database. Once the voice segments and voice model are removed, the application links to the **FreeHTMLS** Condition block to free the HTMLS resource and end the call. See Free the HTMLS resource on page 79.

The application moves to the **Enrollment** container if the user does not have a voice model, to proceed with enrollment. See Enrollment on page 54.

Verify user from a group

The application moves down the verify user from a group path based on the condition data returned to the resource about which SOAP message to parse.

The **ParseVerifyGroup** Container block performs the functions associated with parsing the SOAP message (`Identify` API method result) to verify the user from a group of users. Once

the user is verified, the application links to the **FreeHTMLS** Condition block to free the HTMLS resource and end the call. See Free the HTMLS resource on page 79.

The application moves to the **EvalGroupScores** container returns to the **Authentication** Container to speak back the verification result against each user in the group.

Start session

The application moves down the start session path based on the condition data returned to the resource about which SOAP message to parse.

The **ParseStartSession** Container block performs the functions associated with parsing the SOAP message to start VocalPassword session. Once the session is started, the application links to the **FreeHTMLS** Condition block to free the HTMLS resource and end the call. See Free the HTMLS resource on page 79.

End session

The application moves down the end session path based on the condition data returned to the resource about which SOAP message to parse.

The application links to the **FreeHTMLS** Condition block to free the HTMLS resource and end the call. See Free the HTMLS resource on page 79.

Create Adhoc group

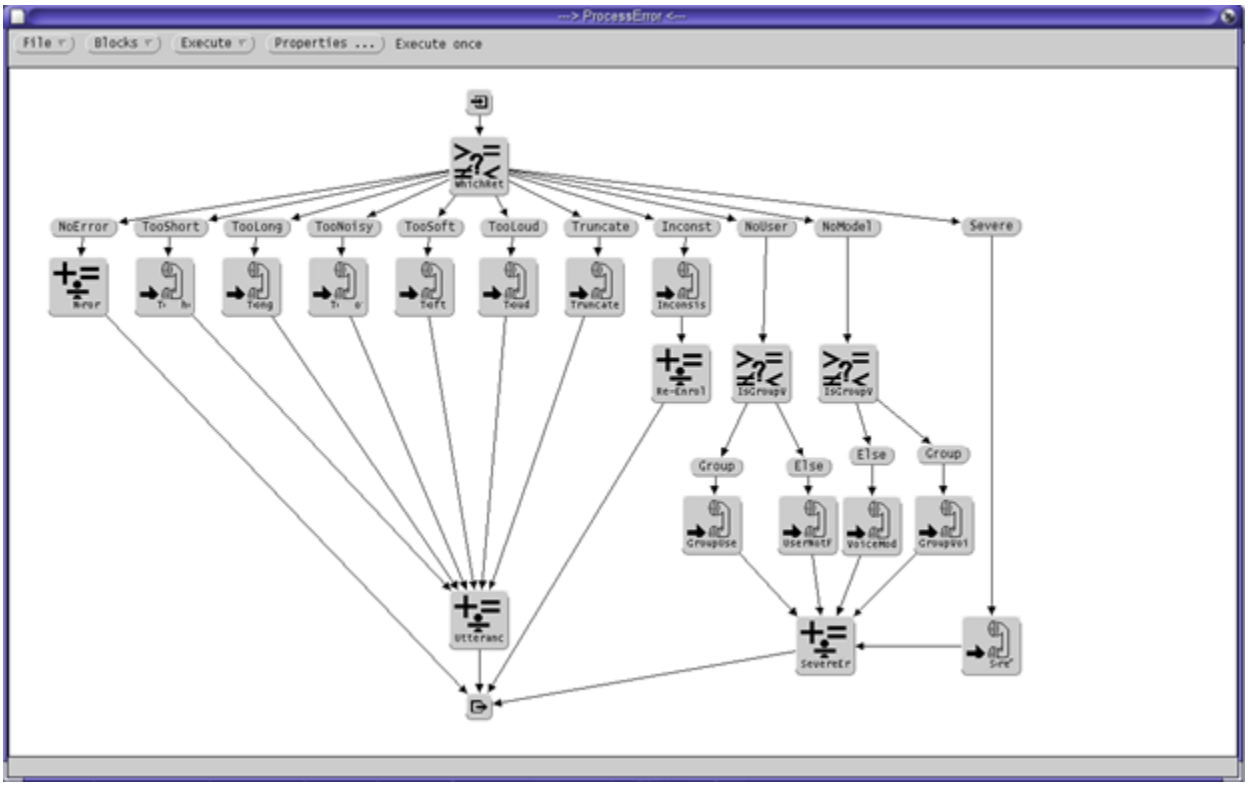
The application moves down the create ad-hoc group path based on the condition data returned to the resource about which SOAP message to parse.

The **ParseCreateAdhocGroup** Container block performs the functions associated with parsing the SOAP message (`CreateAdhocGroup` API method result) to create a temporary group of users which will be used to identify utterance against the group. Once the group is created, the application links to the **FreeHTMLS** Condition block to free the HTMLS resource and end the call. See Free the HTMLS resource on page 79.

The application moves to the **Identify** connector which returns to the **VerifyGroup** Container to verify the speaker against each user in the group.

Process error codes for HTMLS message

The **`ProcessError`** Container illustrates how to handle error return codes associated with an HTMLS message when an error occurred. The sample application shows only a subset of the error codes (the ones that are most likely to be returned) as an example of how to actually handle the error, regardless of which return code the application receives. A complete description of return codes are described in the documentation delivered by Nuance Communications, Inc. Refer to that documentation to determine which return codes to handle in your application.



Which return code to process

The **`WhichReturnCode?`** Switch block takes the path of the return code that was passed as a parameter to the container in **`Params.ExceptionCode`** datacard.

Process No Error return code

The **`NoError`** Compute box sets the **`Params.Retry`**, **`Params.SevereError`**, and **`Params.Re-Enroll`** datacards to `false`.

Speak message to caller based on return codes

The **TooShort** Speak block is an example of a message spoken to the caller in the case when the utterance was too short. Similar messages are spoken when the message is too long, too noisy, too soft, to loud, or truncated. The caller is prompted to retry speaking an utterance.

- **TooShort**: speaks a prompt informing the user that the utterance was too short.
- **TooLong**: speaks a prompt informing the user that the utterance was too long
- **TooNoisy**: speaks a prompt informing the user that there was too much noise embedded in the utterance.
- **TooSoft**: speaks a prompt informing the user that the utterance was too soft.
- **TooLoud**: speaks a prompt informing the user that the utterance was too loud.
- **Truncated**: speaks a prompt informing the user that the utterance was truncated.

Process error Inconsistent

The **Inconsistent** Speak block informs the user the enrollment utterances were inconsistent. The **Re-enroll** Compute block sets the **Params.Re-Enroll** datacard to true.

Process error No User

The **IsGroupVerify?** Compute block checks if it was a group verification or single user verification. The **GroupUserNotFound** Speak block speaks a prompt informing the user that one or more of the group member IDs was not found. The **UserNotFound** Speak block speaks a prompt informing the user that the user ID was not found.

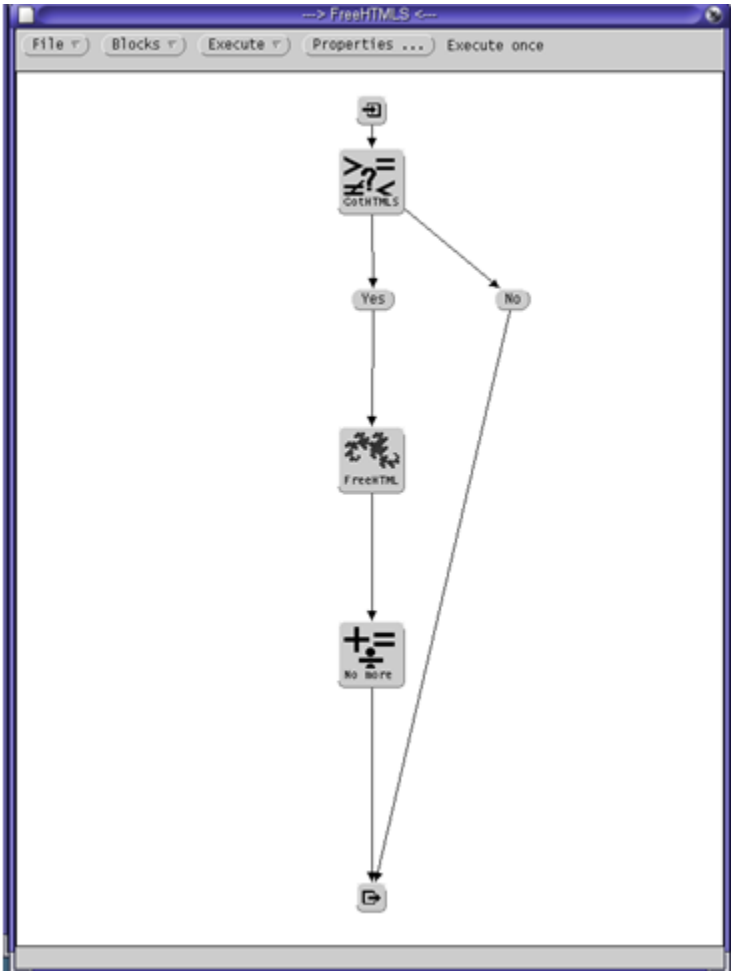
Process error No Model or Severe

The **VoiceModelNotFound** Speak block speaks a prompt informing the user that the voice model was not found.

The **SevereError** Compute block sets the **Params.SevereError** datacard to true. The **Severe** Speak block speaks a prompt informing the user that a severe error has occurred and advising the user to check the system logs.

Free the HTMLS resource

The **FreeHTMLS** Container block determines if the HTMLS resource is allocated, and if allocated, it frees the resource. The **GotHTMLS?** Switch block checks if the HTMLS resource is allocated. The value of true or false resides in the **AppFolder.GotHTMLSResource** datacard.

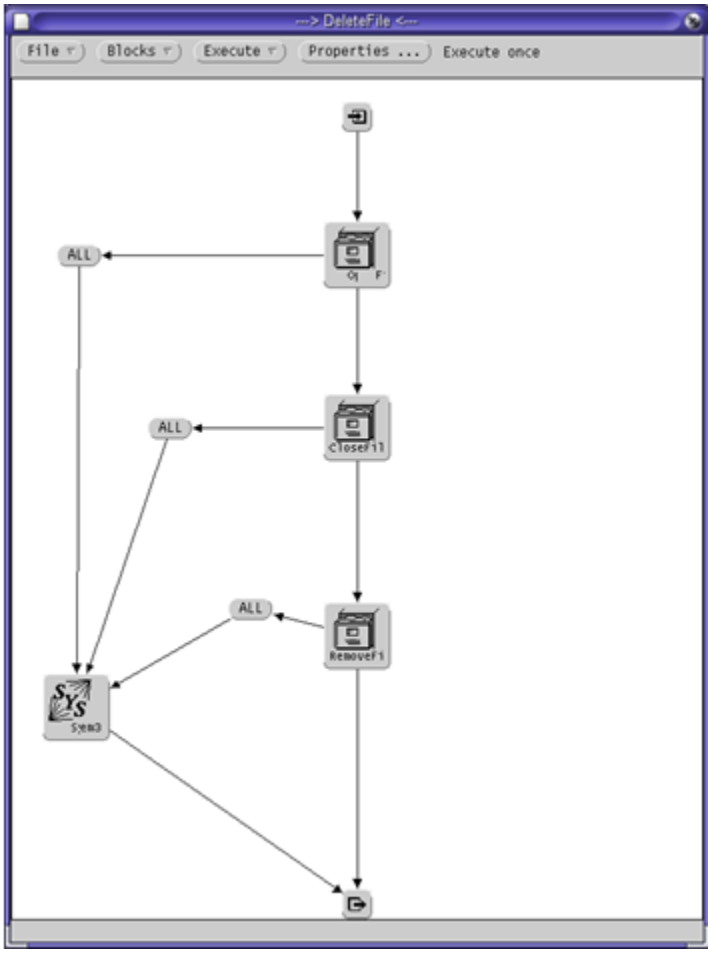


Free resource and set allocation to False

If the HTMLS resource is allocated, the **`FreeHTMLS`** Resource block frees the resource. Once the resource is free, the **`NomoreHTMLS`** Compute block, sets the value of the **`AppFolder.GotHTMLSResource`** datacard to false, which prevents future attempts to free the HTMLS resource.

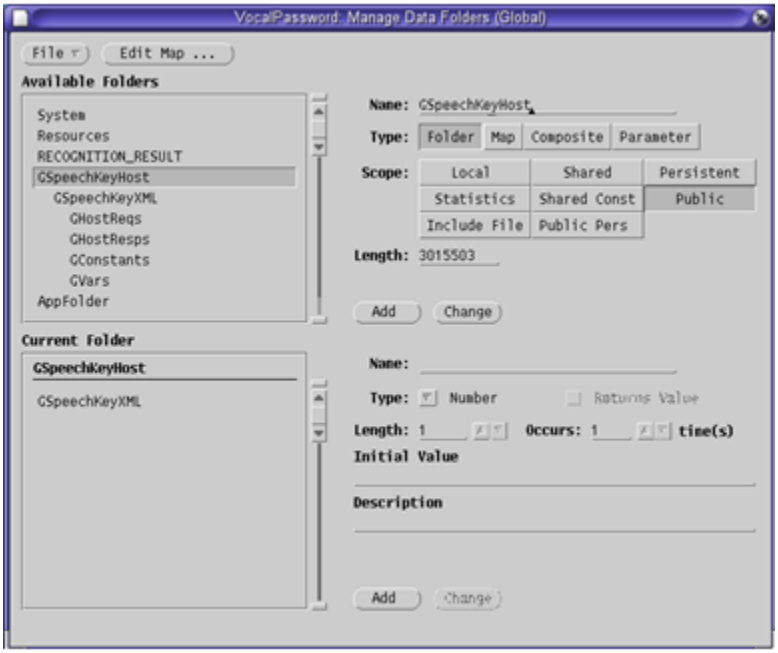
Delete file

The **`DeleteFile`** Container block removes the file from the database and generates an alarms if it cannot remove the file. The name of the file to delete is passed to the **`OpenFile`** DISK I/O block, and the block opens the file. The **`CloseFile`** and **`RemoveFile`** DISK I/O blocks close and remove that file. The **`Alarm`** System block generates an application alarm if there is a problem removing the file.



GSpeechKeyHost folder

This application uses the `GSspeechkeyhost`.folder, which is the top level folder where all subfolders and datacards that are key to the VocalPassword 8.x sample application reside. It consists of four subfolders



`GSspeechkeyhost`.folder subfolders:

- `GHostReqs` – It contains subfolders and datacards that contain information that apply to the caller's requests.
- `GHostResps` – It contains subfolders and datacards that contain information that apply to the caller's responses.
- `GConstants` – It contains subfolders and datacards that with constants that apply to the message and transaction type.
- `GVars` – It contains datacards that hold variables used during caller's transaction.

Chapter 4: VoiceXML Application Development

This chapter covers:

1. Nuance VocalPassword VoiceXML application programming
2. Sample application functional description
3. Debugging tips for VoiceXML applications
4 VoiceXML application flow charts

Nuance VocalPassword VoiceXML application programming

This chapter describes how to program a VoiceXML application to use Nuance VocalPassword 8.x. The sample application shows how to implement the Nuance VocalPassword 8.x features. The documentation is directed at skilled programmers who are familiar with developing VoiceXML application for the Avaya MPS environment. The sample application and the files that support the application are installed with the Avaya software. See Install and configure MPS Application Processor Node on page 24.

Note:

The StartSession API of VP8.3 requires externalSessionId parameter, comprision to VP8.1.

Sample application functional description

Both the MPS Developer and VoiceXML sample applications follow the same basic design and call flow. The application opens by greeting the caller and asks for the caller's nine digit account number. The VocalPassword Server does not require the account number to be nine digits or even to be numeric. This request is simply a function of this sample application. The application then asks the user to select one of six actions that demonstrate the VocalPassword features of enrollment, single user verification, group verification, unenrollment, and liveness

verification. For a description of these features, see Sample application functional description on page 9.

Debugging tips for VoiceXML applications

The following information is helpful when debugging VoiceXML applications.

- Use the utility `validatedoc` to inspect a static VoiceXML page and determine if the VoiceXML code is valid. The utility does not check ECMA script. For example, to use the utility to validate the code in an application named `VP-Main.vxml` type the following:

```
validatedoc VP-Main.vxml
```

- Type the following command to observe a VXMLI trace of a VoiceXML call. Replace X with the line number associated with this application.

```
vsh vxmli dlogdbgon file,line=X
vsh vxmli dlogdbgon file,channel
```

- VXMLI process logging resides in the following three files.

```
$MPSHOME/common/log/vxmli_diag.mps.3.log
$MPSHOME/common/log/vxmli_event.mps.3.log
$MPSHOME/common/log/dlog/vxmli_def.mps.3.dlog
```

- Perform the following actions to see the last 20 pages of VoiceXML code (static or dynamic), browsed by the VoiceXML Interpreter (VXMLI) per line.

  - Open the file `$PERIVXMLHOME/SBclient.cfg` file and edit the following two lines, setting the value to `1`.

    ```
    client.log.diagTag.8004 VXIInteger 1
    client.log.diagTag.8005 VXIInteger 1
    ```

  - Type the following lines to restart VXMLI.

    ```
    vsh srp vxmli -stop
    vsh srp vxmli -start
    ```

  - The VXML pages are written in round robin order at the following location.

    ```
    $MPSHOME/common/log/vxidump/mpsN_vxmli/00000XYZ/com.avaya.vxi.000000AB
    ```

    Replace the variables as follows.

    - XYZ is the three digit line number
    - AB is a two digit number 00 to 19

    After page 19 is written, page 00 is written to maintain the round robin order.

VoiceXML application flow charts

This section presents a set of flow charts that illustrate the call flow of the sample VoiceXML applications. These call flow charts are summarized on the following table.

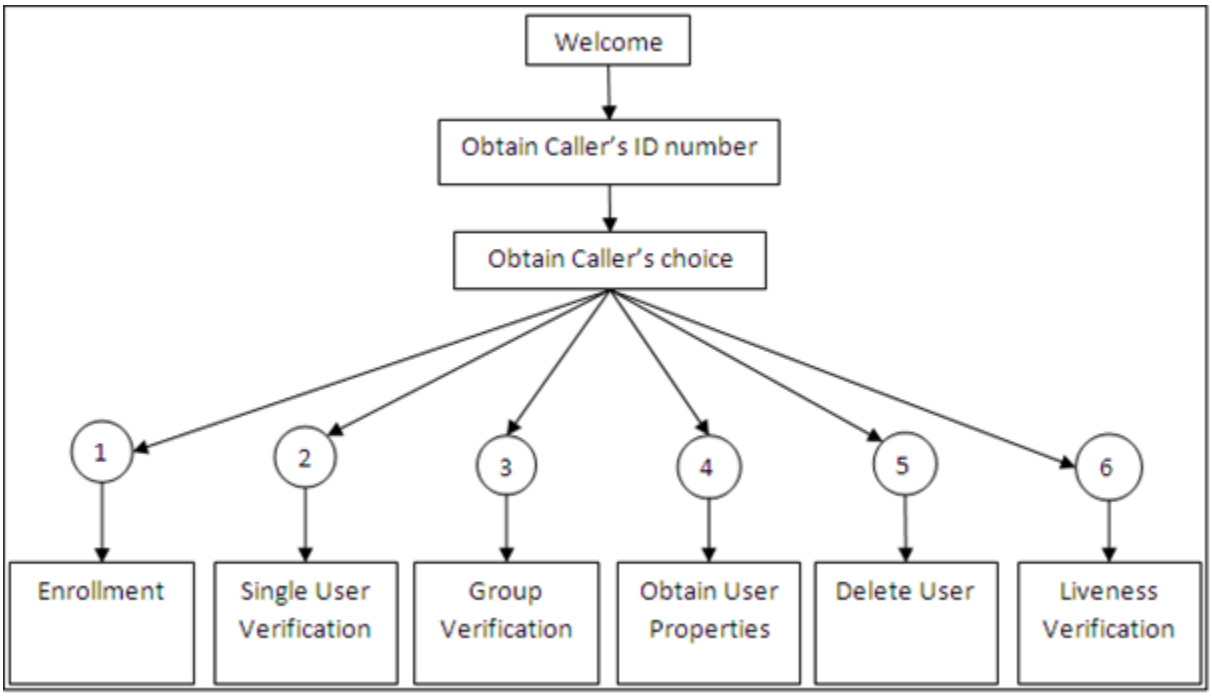
Table 1: VoiceXML Application Call Flow Charts

Application	Flow chart description
`VP-main.vxml`	Main page that is executed when a caller dials into the application and asks the caller to choose one of six actions: enrollment, verify a single user, verify a user from a group, obtain user properties, delete a user, and liveness verification. See VP-Main flow chart on page 86.
`VP-enrollment.vxml`	VoiceXML page that guide the caller through the enrollment process. See Enrollment flow chart on page 86.
`VP-verifyUser.vxml`	VoiceXML page that guide the caller through the verification process. See Single User Verification flow chart on page 87.
`VP-verifyGroup.vxml`	VoiceXML page that guide the caller through the group verification process. See Group Verification flow chart on page 88.
`VP-queryUserExt.vxml`	VoiceXML page that tells the user that querying extended user's info is not supported by VocalPassword. See Obtain User Properties flow chart on page 89.
`VP-deleteUser.vxml`	VoiceXML page which informs the user that he has been removed from the VocalPassword database. See Delete User flow chart on page 89.
`VP-verifyLiveness.vxml`	VoiceXML page that guide the caller through the liveness verification process. See Liveness Verification flow chart on page 89.

VP-Main flow chart

The VP-Main.vxml file is the main page that is executed when a caller dials into the application. It asks the caller to enter his/her account number and to choose one of the following six call flows:

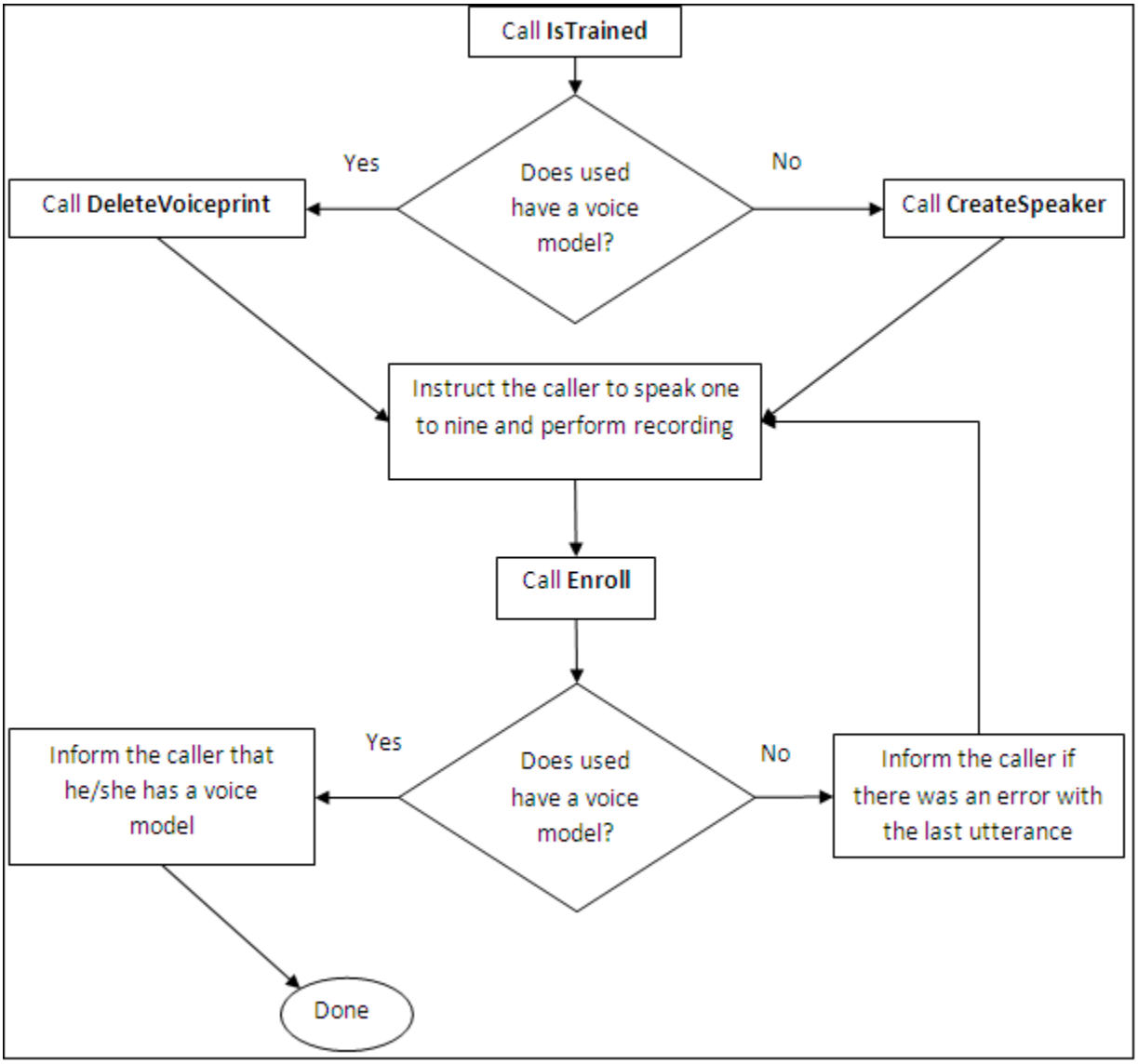
- Enrollment
- Single User Verification
- Group Verification (the application prompts the caller for three account numbers to form the “group”)
- Obtain User Properties
- Delete User
- Liveness Verification



Enrollment flow chart

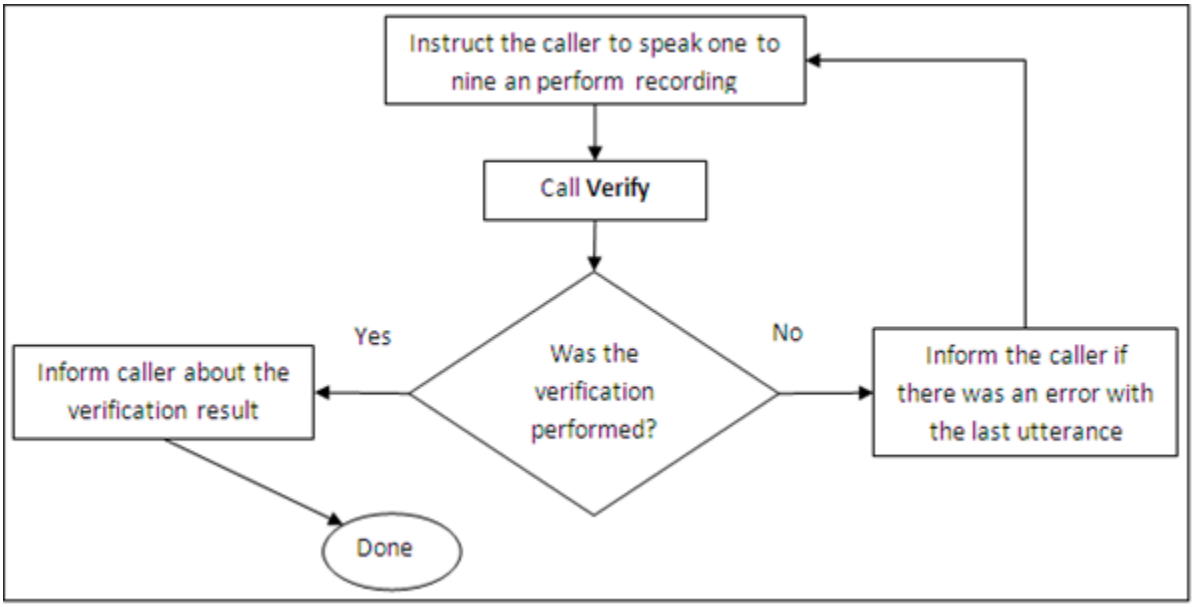
The `VP-enrollment.vxml` application leads the caller through a speaker verification enrollment session. It first checks to see if the caller already has a voice model associated with his/her account number. If so, the application clears the voice model. Otherwise, the application creates a new user in the VocalPassword database. Then, the application repeatedly (at least

three times) asks the caller to speak the digits one through nine until a voice model is created.



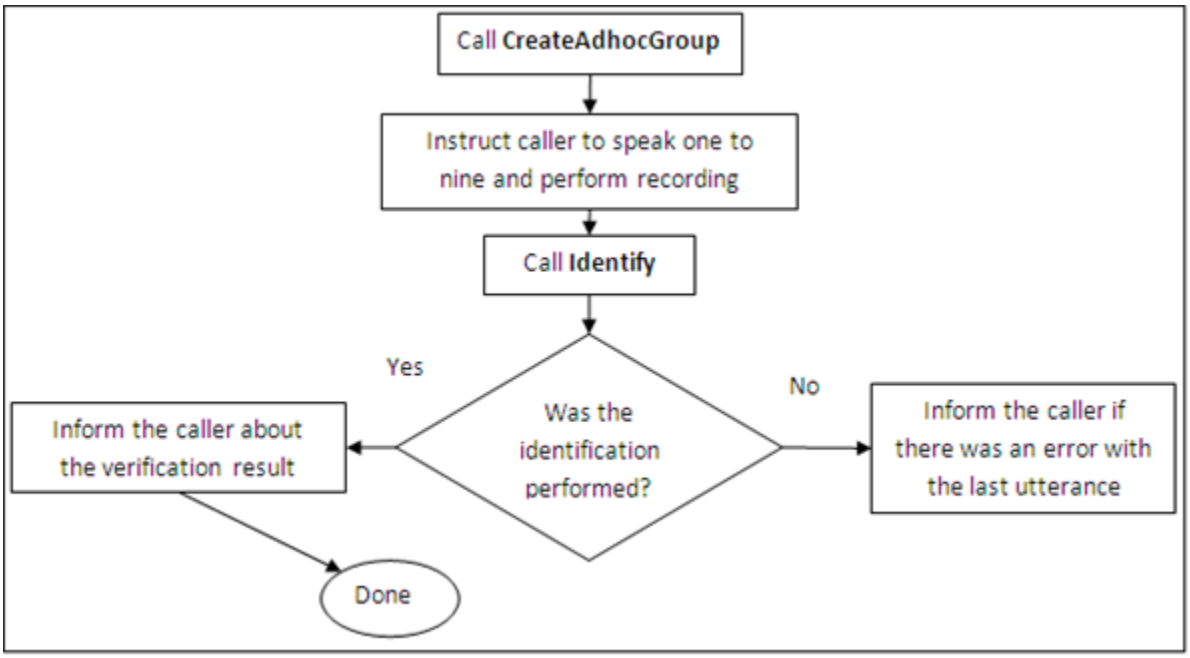
Single User Verification flow chart

The `VP-verifyUser.vxml` applications ask the caller to speak the digits one through nine. Then, the application checks the caller's recorded speech against the voice model of the user whom the caller claims to be. The application informs the caller of the result.



Group Verification flow chart

The `VP-verifyGroup.vxml` applications ask the caller to speak the digits one through nine. Then, the application checks the caller's recorded speech against each voice model in the sample group. The application informs the caller of the result against each voice model in the group.

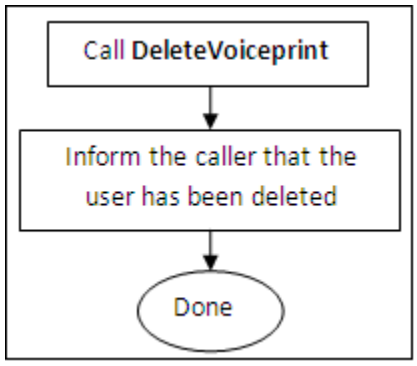


Obtain User Properties flow chart

The `VP-queryUserExt.vxml` application informs the user that such functionality is not supported by Nuance Vocal Password.

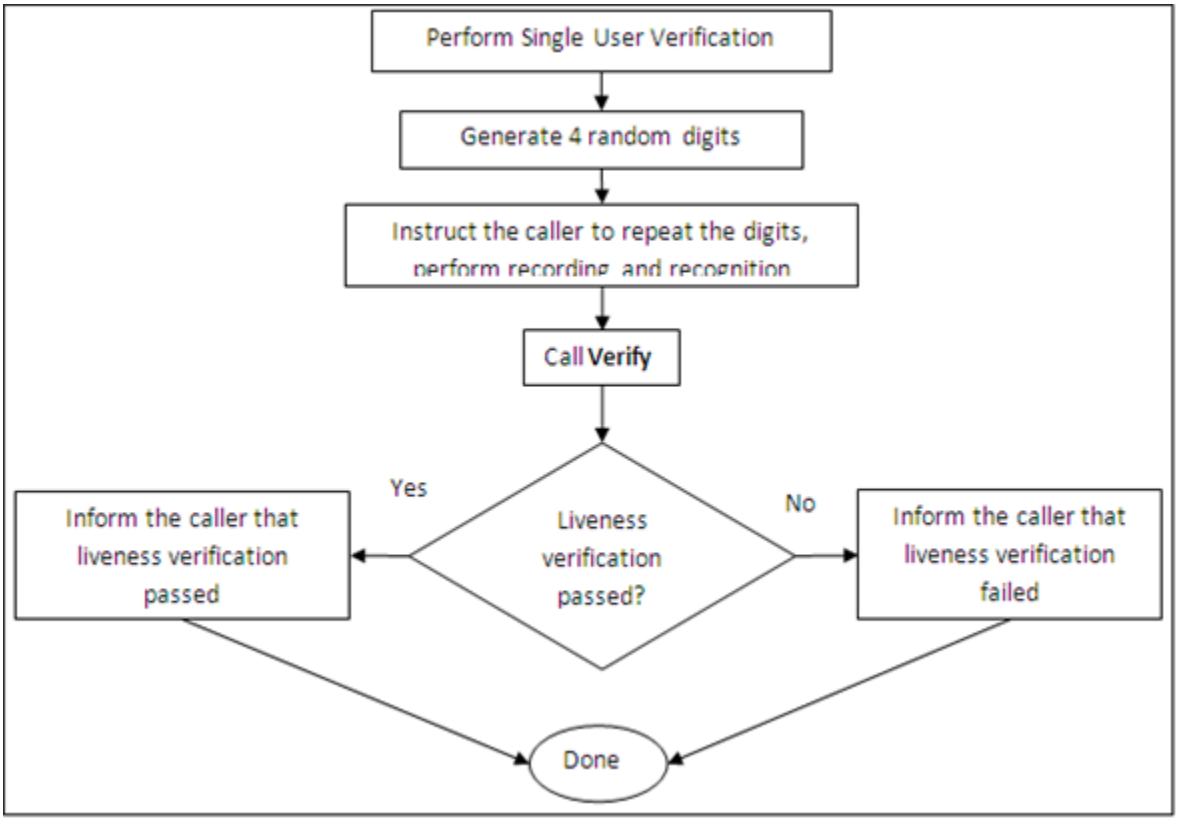
Delete User flow chart

The `VP-deleteUser.vxml` application deletes the caller's entry from the voice model database of VocalPassword.



Liveness Verification flow chart

The `VP-verifyLiveness.vxml` application first asks the caller to speak the digits one through nine and verify his/her identity using the Verify Single User logic. Then the application asks the caller to repeat a sequence of random digits. The application will recognize the digits to see if they are the ones the caller was asked to speak and then authenticate the liveness utterance comparing the liveness utterance to the most recent verification utterance


Perform Single User Verification
Generate 4 random digits
Instruct the caller to repeat the digits, perform recording and recognition
Call Verify
Liveness verification passed?
Yes
No
Inform the caller that liveness verification passed
Inform the caller that liveness verification failed
Done

Chapter 5: Alarms

This chapter covers:

1. Alarm overview
2. Alarm log files
3. Alarm format
4. Nuance VocalPassword-specific alarms
5. MPS Developer alarm generation
6. VoiceXML alarm generation

Alarm overview

This section of the document describes Nuance VocalPassword 8.x-specific alarms in the Avaya Media Processing Server (MPS) environment.

Be aware that the only alarms that can be generated for Nuance VocalPassword 8.x are those that were explicitly created during application development. No alarms are integrated from Nuance VocalPassword itself.

Warning:

Avaya highly recommends that you follow the design of the sample applications and generate application alarms. An application alarm template is provided specifically for use with the VocalPassword Server in the `VocalPassword.alarm` file.

For information about:

- developing alarms in an MPS Developer application, see Nuance VocalPassword MPS Developer Application programming on page 41
- how to generate alarms from an MPS developer application, see MPS Developer alarm generation on page 94
- developing alarms in an VoiceXML application, see Nuance VocalPassword VoiceXML application programming on page 83
- how to generate alarms from an VoiceXML application, see VoiceXML alarm generation on page 94.

Alarm log files

There are three types alarm files which capture alarms generated on the MPS Application Processor node (alarm, warning, notification). Each type is written to an alarm log file named for the type of alarms being captured, in the following format. In the actual file, N is replaced with the MPS number.

```
alarm.mps.N.log
warning.mps.N.log
info.mps.N.log
```

There are also three types alarm files that will capture common alarms generated on the MPS Application Processor node (alarm, warning, information). Each type is written to an alarm log file named for the type of alarms being captured.

```
alarm.common.0.log
warning.common.0.log
info.common.0.log
```

There is one file that captures all application alarms, regardless of the severity. The file name contains the number of the MPS node where the alarm was generated, in the following format:

```
app.mps.N.log
```

In the actual file, N is replaced with the MPS number.

Alarm format

Alarm messages contain the information you must work toward a resolution of the issues that generated alarms.

- The first line of an alarm message includes the name of the day of the week, date, time, process that generates the alarm, alarm number, line number (in the case of channellevel alarm messages), severity, and Speech Server component number.
  - Channel-level alarms display a line number in the first line.
  - System-level alarms do not display a line number in the first line.
- The second line of the alarm message contains descriptive information.

The following examples illustrate the first lines of alarm, warning, and notification messages (`info` alarms).

- Alarm Messages

Alarm messages are all rated Severity 9 and reside in the file named with the syntax:

```
alarm.<component_type>.<component_number>.log
```

- Sample: system-level severity 9 alarm

```
Fri May 31 15:02:55 <rcm> 20002 Severity 9 Comp #oscar.1851/ibm-oscar-8
```

- Sample: channel-level severity 9 alarm

```
Fri Jun 07 17:58:57 <rcm> 22002 Line 35 Severity 9 Comp #oscar.1851/
IBMOSCAR-8
```

• Warning Messages

Warning messages are all rated Severity 4 and reside in the file named with the syntax:

```
warning.<component_type>.<component_number>.log
```

- Sample: system-level severity 4 alarm

```
Fri May 31 15:02:55 <rcm> 20204 Severity 4 Comp #oscar.1851/ibm-oscar-8
```

- Sample: channel-level severity 4 alarm

```
Tue Jun 18 15:21:35 <rcm> 20205 Line 1 Severity 4 Comp #oscar.1851/
IBMOSCAR-8
```

• Notification Messages (`info` alarms)

Notification messages (`info` alarms) are all rated Severity 1 and reside in the file named with the syntax:

```
info.<component_type>.<component_number>.log
```

- Sample: system-level severity 1 alarm

```
Fri May 31 09:18:42 <rcm> 20401 Severity 1 Comp #oscar.1851/ibm-oscar-8
```

- Sample: channel-level severity 1 alarm

```
Fri May 31 09:18:55 <rcm> 22401 Line 4 Severity 1 Comp #oscar.1851/
ibmoscar-8
```

Nuance VocalPassword 8.x-specific alarms

The following table describes how the application alarms display. Remember that the only alarms that can be generated for Nuance VocalPassword 8.x are those that were explicitly created during application development. Thus, the actual text is dependent on what the application developer programmed.

For information about:

- how to generate alarms from an MPS developer application, see MPS Developer alarm generation on page 94.
- how to generate alarms from an VoiceXML application, see VoiceXML alarm generation on page 94.

Alarm category	Alarm message and description
VocalPassword 91000 Sev 1	`Fri Oct 26 15:13:03 <VocalPassword> 91000 info Comp #common.0/mpsaxmp` VocalPassword: This is an informational alarm from the VocalPassword application.
VocalPassword 91001 Sev 4	`Fri Oct 26 15:13:05 <VocalPassword> 91001 info Comp #common.0/mpsaxmp` VocalPassword: This is a warning alarm from the VocalPassword application.
VocalPassword 91002 Sev 7	`Fri Oct 26 15:13:33 <VocalPassword> 91002 warning Comp #common.0/mpsaxmp` VocalPassword: This is a severe alarm from the VocalPassword application.

MPS Developer alarm generation

Nuance VocalPassword 8.x alarms must be explicitly created during application development. This section describes how to generate an alarm in an MPS Developer application. For information about developing alarms in an MPS Developer application, see Nuance VocalPassword MPS Developer Application programming on page 41.

- First, define the **`alarmdbtask`** Vengine environment setting to be equal to `VocalPassword`. You need to do this one time only
- Use a System block to generate an alarm.

VoiceXML alarm generation

Nuance VocalPasswprd 8.x alarms must be explicitly created during application development. This section describes how to generate an alarm in a VoiceXML application. For information

about developing alarms in a VoiceXML application, see Nuance VocalPassword VoiceXML application programming on page 83

- Define a new variable named code and set it to `91000` for information alarms, `91001` for warning alarms, `91002` for severe alarms. Refer to Nuance VocalPassword 8.x-specific alarms on page 93.
- Set “key” to `VocalPassword`.
- Set “alcode” to the alarm code: `91000`, `91001` or `91002`.
- Set “ins1” to a text string that provides information about the error.

The following example code shows how to generate a severe alarm.

```
<form id="exitError">
    <object name="alm" classid="com.avaya.ivr.alarm">
        <param name="key" value="VocalPassword"/>
        <param name="alcode" expr="'91002'"/>
        <param name="ins1" expr="'<%=errorText%>'"/>
    </object>
    <block>
        <prompt>
            <audio src="builtin:verifier-severeerror"/>
        </prompt>
        <goto next="#exit"/>
    </block>
</form>
```

Chapter 6: Troubleshooting and Known Issues

This chapter covers:

1. Introduction
2. Troubleshooting

Introduction

This chapter addresses the topics of Troubleshooting and Known Issues.

Troubleshooting

The following table identifies some troubleshooting tips.

Table 2: Nuance VocalPassword 8.x Troubleshooting

Symptom	Probable cause	Resolution
There is an alarm on the MPS with the text: "`Cannot find service 'htmls 12'`"	The HTMLS process is not running on the correct MPS component.	Run `psg htmls`. The result should return `htmls -v`N (where N is the MPS component number). Ensure N is the MPS component number on which you are running a VocalPassword application. If HTMLS is not running, or is running on another MPS component number, open the file `$MPSHOME/common/etc/gen.cfg` and add/edit the following line: `htmls - - 0 0 "htmls -vN"` (where N is the MPS component). Restart the common component after editing.

Symptom	Probable cause	Resolution
	PERIhtmls was not installed on the MPS AP.	Install the PERIhtmls package on the MPS AP.
There is an alarm on the MPS with the text: "`Invalid program or function 'xml-tofol' name. Condition 'pgid'`"	PERIxmlc was not installed	Install the PERIxmlc package on the MPS AP.
The application sends an alarm indicating it cannot find the Web service.	The application cannot contact the VocalPassword Web service.	For MPS Developer applications: Check that the parameter VocalPasswordServiceURL from `VocalPasswordConfig.xml` contains the correct VocalPassword Service URL. For VoiceXML applications: Check that the VoiceXML file `VP-Main.vxml` contains the variable *serviceURL* and it has the correct VocalPassword Service URL.
When running a VoiceXML application, you hear "`There has been a serious error - exiting`"	There is a VoiceXML error.	• Check the VoiceXML logs in `$MPSHOME/common/log`. • Check the `vxidump` logs. • Run `validatedoc` on the VoiceXML file if it is a static file.
The application cannot find recordings.	The application receives an MPS alarm indicating "`Failed to open file...`"	Check that the parameter RecordingBaseDirectory from `VocalPasswordConfig.xml` contains the correct directory and application have a sufficient permissions to access that directory.
VP with the CCSS SIP application does not support the G729 audio codec..	VP with CCSS SIP application supports only the G711 audio codec.	VP with CCSS SIP application supports only the G711 audio codec.

Index

A

add enrollment utterance 73
alarms 91–93
  alarm format 92
  alarm log files 92
  alarm overview 91
  Nuance VocalPassword 8.x-specific alarms 93
application programming 83
  Nuance VocalPassword VoiceXML 83

C

clear user 76
configuration 28, 29
  Liveness Detection 29
  VocalPassword 28
create Adhoc group 77

D

database server node 9
debugging tips 84
  VoiceXML applications 84
delete file 80

E

end session 77
enrollment 10

F

fraudster detection 39
free the HTMLS resource 79

G

group verification 10
GSpeechKeyHost folder 81

H

hardware and software requirements 7–9

I

install and configure 15, 24
  VocalPassword server node 15
install and configure VocalPassword server node 16, 17
  install and configure database node 16
  installation roadmap VocalPassword node 17
installation and configuration required for VocalPassword 7.x 24
  MPS Developer application 24
installation and configuration required for VocalPassword 8.x 24
introduction 5
  Nuance VocalPassword 5

L

life cycle 42
  single user 42
Liveness Detection with Dictionary 29
liveness verification 11

M

main module . 44, 46, 47, 50–54, 59, 61, 64, 65, 68, 69, 72
  Action 51
  authenticate or verify a user 59
  Begin call 46
  delete user 72
  end the call 68
  Enrollment 54
  GetAccountNumber 50
  HTMLSResourceOperations 69
  liveness verification 65
  read configuration parameters 47
  start session 52
  unenrollment 64
  verify against members of a group 61
MPS application processor node 7
MPS Application Processor Node 24

N

Nuance VocalPassword 41
  MPS Developer Application programming 41
Nuance VocalPassword 8.x-specific alarms 93, 94

MPS Developer alarm generation 94
VoiceXML alarm generation 94
Nuance VocalPassword API methods used by sample applications 11

O

overview 6, 15
configure VocalPassword server nod 15
install VocalPassword server nod 15
installation and configuration 15
system architecture 6

P

prerequisite 16
using VocalPassword 8.x 16
process error codes for HTMLS message 78
programming pattern 43
request-response 43

Q

query user 76

S

sample application 83
functional description 83
sample application functional description 9–11
single user verification 10
start session 77
support for VocalPassword 6

T

troubleshooting 97

U

unenrollment 11

V

vendor reference documentation 6
Verification types 13
verify user 75
verify user from a group 76
VocalPassword 7.x installation and configuration 26
VoiceXML application installation and configuration 26
VocalPassword advanced configuration 28, 30, 32–34, 36
configuring for Text-Independent verification 30
configuring for Text-Prompted verification 30
configuring Playback detection 32
configuring the number of utterances required for enrollment 33
configuring to catch inconclusive verification results 32
Configuring unsupervised adaptation 34
creating Background Models with the Calibration Wizard 36
VocalPassword server node 8
VoiceXML application 85
flow charts 85
VoiceXML application flow charts 85–89
Delete User flow chart 89
Enrollment flow chart 86
Group verification flow chart 88
Liveness Verification flow chart 89
Obtain User Properties flow chart 89
Single User Verification flow chart 87
VP-Main flow chart 86

W

WhichAction 53