Speaker Verification in BeVocal VoiceXML

Version 1.5
May 2001

BeVocal, Inc.
1380 Bordeaux Drive
Sunnyvale, CA 94089

Table of Contents

Preface

This document describes a BeVocal experimental extension to VoiceXML that provides speaker verification.

Audience

This document is for software developers using the BeVocal Café development environment. It assumes you are familiar with the basic concepts of HTML and with VoiceXML.

Conventions

Bold font is used for:

- Headings

`Fixed width` font is used for:

- Code examples
- Tags and attributes
- Values or text that must be typed as shown

`Italic fixed width` font is used for:

- Variables

Italic font is used for:

- Introducing terms that will be used throughout the document
- Emphasis

References

For additional or related information, you can refer to VoiceXML Reference. BeVocal.

(http://cafe.bevocal.com/docs/vxml/index.html)

1 Speaker Verification

BeVocal Café now supports speaker verification, which consists of two phases: registering a voice print for a user and authenticating a user's identify. Speech samples can be obtained from an authorized user of an application and used to create and register a voiceprint—that is, a model of that user's voice. When the same user later interacts with the application, new speech samples can be compared with the stored voiceprint to authenticate the speaker's identity. BeVocal has added two new tags to VoiceXML to provide this capability.

Note: Speaker verification is an *experimental extension* to VoiceXML; its implementation and behavior are subject to change. BeVocal is providing the current implementation before the feature has been standardized so that our developers may provide feedback. If this capability becomes a standard part of a future version of VoiceXML, the BeVocal implementation will change as necessary to match the VoiceXML standard.

Voice Prints

A *voice print* models physiological characteristics of a particular person's voice and can be used to authenticate that person's identity. Authentication against a voice print is based on inherent properties of the speaker's voice, so it provides a higher level of security than prompting for a password or personal identification number.

Creating a Voice Print

A BeVocal VoiceXML application can create a voice print that identifies a particular user. The voice print is created and refined based on speech samples collected during a training dialog. The application collects these speech samples with the new `<register>` VoiceXML tag.

Keys

A voice print is identified by a unique key.

Note: The BeVocal Café VoiceXML Interpreter combines a Café account-specific value with any specified key and uses the combination to identify a voice print.

Because keys are used in conjunction with account information:

- When different VoiceXML applications run under *different accounts*, each application will associate a *different voice print* with a given key (for example, the key "1234").
- When different applications run under the *same account*, all these application will associate the *same voice* print with a given key.

As a developer, you are responsible for ensuring that each user of your various applications has a unique key. That is, you should not use the same key to identify different users to different applications that will run under your account.

Comparing Voice Prints

To verify the identify of a user, a BeVocal VoiceXML application first obtains whatever information it needs to determine the key for the user. It then compares the user's speech with the stored voice print for the user with that key. The comparison is performed by the new `<verify>` VoiceXML tag.

Registering a Voice Print

An application that uses speaker verification must register voice prints for its authorized users, identifying each user with a unique key. Typically, the application obtains a key for the user and then enters a *training dialog*, so called because it trains the verifier to recognize a particular voice. During a training dialog, the application uses a number of `<register>` elements to obtain the speech samples from which the voice print is created.

Training

The `<register>` element is a new field item; it is similar to `<field>` in that it prompts for spoken input that matches a particular grammar—either one of the standard built-in grammars, or a grammar specified within the `<register>` element. When the user's response matches the grammar, it is sent to the verifier. The verifier uses the speech samples provided by `<register>` to "learn" the user's voice.

The verifier creates a voice print from the first speech sample it receives for a particular key. The `mode` attribute of `<register>` controls what happens when a voice print is already registered for the specified key. The default is to use the new sample to refine the existing voice print. Alternatively, you can delete the existing voice print and use the new sample to create a new voice print for the key. Or, you can skip execution of the entire `<register>` element if a voice print is already stored for the specified key.

Designing a Training Dialog

A training dialog creates a voice print for a particular user and stores that voice print with the user's identifying key. Each speech sample provided during the dialog allows the verifier to refine the voice print. In other words, the interaction trains the verifier to recognize the user's voice; additional training results in better recognition.

Invoking Training

You typically perform the training dialog only once for a particular user. You first obtain the key that identifies the user. If no voice print is registered for that key, the training dialog registers a new voice print.

You can use the `mode` attribute of `<register>` elements to ensure that a given user goes through the training dialog only once. This attribute determines what happens when a voice print already exists for the specified key:

- If the mode is `skip`, the entire `<register>` element is skipped. This mode is typically used for the first `<register>` element in the training dialog.
- If the mode is `delete`, the existing voice print is deleted and a new voice print is created. You can use this mode for the first `<register>` element in the training dialog when you have a reason to replace an existing voice print. For example, suppose each client company authorizes a single agent to use your application;

you use the clients' names as keys identifying voice prints. You may want to implement a special dialog to be invoked when a client changes its authorized agent. That interaction would replace the voice print of the former agent with the voice print of the new agent.

- If the mode is `adapt`, the existing voice print is refined with the new voice sample from by the `<register>` element. This is the default mode; it should be used in all but the first `<register>` element in a training dialog.

If no voice print exists for the specified key, the `<register>` element creates a new voice print regardless of the setting of its `mode` attribute.

You can also use an event handler to ensure that the training dialog is invoked only if there is no voice print for a given key. Your application can invoke an authentication dialog as soon as it obtains the user's key. A `<verify>` element throws an `error.verify.keynotfound` event if no voice print has been registered for the specified key. An error handler that catches this event can transfer control to the training dialog. At the end of the training interaction, you can transfer control back to the authentication dialog.

Collecting Speech Samples

The verifier uses the first speech sample in a training dialog to create a voice print; it uses each additional speech sample to refine the voice print. You should design a training dialog to obtain a variety of speech samples without making the interaction seem long and tedious to the user. Because training is a one-time dialog, you can make it longer than a dialog that will be repeated frequently (for example, an authentication dialog).

A `<register>` element sends a speech sample to the verifier only when the spoken input matches the `<register>` element's grammar. Any DTMF input is rejected, causing a nomatch event. An utterance that does not match the grammar is discarded, so it does not affect the voice print. Similarly, an utterance that matches a universal grammar or other active grammar (such as "help") is not used as a speech sample.

Tips:

- Your training dialog should ask for the information that will be requested for authentication. For example, both training and authentication might prompt for the user's home phone number.
- The training dialog should ask for additional information that may not be requested by the authentication dialog.
- You should get at least two speech samples in any training dialog.
- You should get speech samples of different types, for example, phone number and digits.
- The dialog should ask for some information twice.

  You could repeat a question. For example, one `<register>` element could ask, "Please say your date of birth." The next one could ask, "Please repeat your date of birth."

  Or, you could have the user say a sequence of digits twice. For example, "Please say five six seven nine, five six seven nine".
- Don't ask for short and simple text like "yes" and "no" during training.

Authenticating a User's Identity

Once a user has registered a voice print, subsequent interactions can require authentication of the user's identity before proceeding with sensitive interactions that access private or restricted information. Typically, the application obtains a key for the user and then enters an *authentication dialog*, that compares the user's voice with the stored voice print for that key. During an authentication dialog, the application uses `<verify>` elements to obtain speech samples to be compared with the user's stored voice print.

Authentication

The `<verify>` element is a new field item; it is similar to `<field>` in that it prompts for spoken input that matches a particular grammar—either one of the standard built-in grammars, or a grammar specified within the `<verify>` element. When the user's response matches the grammar, it is sent to the verifier. The verifier compares the speech sample provided by `<verify>` with the stored voice print for a specified key. It throws an `error.verify.keynotfound` event if no voice print has been registered for the specified key. The comparison produces one of three possible results: the speech sample may match the voice print, it may fail to match, or the verifier may be unsure whether it matches.

Authentication takes place at the form level:

- All `<verify>` elements in a given form must compare speech samples against the same voice print—that is, the voice print with the same identifying key.
- The authentication process ends only when the form is exited—not after the execution of the last `<verify>` element in the form. For this reason, it is a good practice to perform the authentication dialog in a separate form.

At the end of the authentication process (when the form is exited), if speech samples provided by `<verify>` elements matched the stored voice print, the verifier automatically uses those speech sample to refine the stored voice print, thus improving performance in future authentication dialogs.

Designing an Authentication Dialog

An authentication dialog compare's the user's voice with the stored voice print for the user's identifying key. If necessary, the dialog can compare more than one speech sample against the voice print.

Invoking Authentication

You typically perform the authentication dialog at the beginning of any interaction that accesses private or restricted information. You first obtain the key that identifies the user. The authentication dialog then determines whether the user's voice matches the voice print registered for that key.

Comparing Speech Samples

Because authentication happens whenever a user performs a particular sensitive interaction, it should be a short as possible. You should design an authentication dialog to obtain no more speech sample than the verifier needs to can determine whether a match has occurred.

A `<verify>` element sends a speech sample to the verifier only when the spoken input matches the `<verify>` element's grammar. Any DTMF input is rejected, causing a nomatch event. An utterance that does not match the grammar is

discarded, so it is not compared to the voice print. Similarly, an utterance that matches a universal grammar or other active grammar (such as "help") is not used as a speech sample.

After a comparison occurs, the result is returned in the `decision` property of the `<verify>` element's shadow variable. For example, if the element's name is `comparePhone`, the result is available as the value of the expression `comparePhone$.decision`.

You should check the result of each comparison. The possible results are:

- `accepted` - The speaker's voice matches the voice print.
- `rejected` - The speaker's voice does not match the voice print.
- `unsure` - The verifier could not determine whether the speaker's voice matches the voice print.

If comparison accepted the user's identity, the application can proceed with the restricted interaction. If it rejected the speaker's identity, the application can explain that the user is unauthorized and exit or transfer to a different form. If the verifier was unable to accept or reject the speaker, the application can compare additional speech samples. For example, it might repeat the same `<verify>` element or continue to the next `<verify>` element in the dialog to prompt for different information.

Tip:

- Your authentication dialog should ask for a piece of information that was provided during the training dialog. For example, both training and authentication might prompt for the user's home phone number.

2 Tag Reference

This chapter provides detailed information about the VoiceXML tags that support Speaker Verification. Each entry includes:

Syntax	Summary of how the tag is used.
Description	Description of attributes or other details.
Usage	Table of parent and children tags. Parent tags can contain this tag and children tags can be used within this tag.
See Also	Links to related information.
Examples	Short examples you can run as simple, standalone applications.

<register>

Register a voice print that can be used to verify caller identity.

Syntax

```
<register
    name="string"
    type="boolean"|"date"|"digits"|"currency"|
         "number"|"phone"|"time"
    keyExpr="js_expression"
    mode="adapt"|"delete"|"skip">
  Child Elements
</register>
```

Description

Field item that prompts for a value matching a particular grammar and then registers the user's response as a voice print for the specified key.

Any utterances that match a universal grammar or a grammar in document or application scope are recognized as they would be in a `<field>` element. However, only those utterances that match the grammar of the `<register>` element itself are used to create or adapt a voice print.

Note: Unlike other field items, a `<register>` element is not a possible go-back destination for the BeVocal go-back facility. That is, the user cannot say "go back" to return to a `<register>` element, retracting earlier input to that element. See VoiceXML Reference for a description of the go-back facility.

Any DTMF input is rejected and results in a nomatch event.

Attribute	Description
name	Name of the field item variable that will hold the recognition result. The variable name may not be a JavaScript reserved keyword. The field item variable has dialog (form) scope.
type	Specifies an internal grammar. *Optional* (as alternative to a child <grammar> element). • boolean - Grammar for recognizing positive and negative responses. Returns "true" for yes and "false" for no. • date - Grammar for recognizing dates. Returns string with format "*yyyymmdd*"; "????" is used for an unknown year and "??" is used for an unknown month or day. • digits - Limited grammar for recognizing a sequence of digits. Returns a string of digits. • currency - Grammar for recognizing amounts of money, in dollars (*Not Implemented*: International currency designation). Returns a string with format "*mm.nn*". • number - More general grammar for recognizing numbers. Returns a string that could include digits, a decimal point, or positive or negative sign. • phone - Grammar for recognizing a telephone number adhering to the North American Dialing Plan (with no extension). Returns a sequence of digits. • time - Grammar for recognizing a time. Returns a string with format "*hhmmx*" where and *x* is one of: "a" for AM, "p" for AM, "h" for 24 hour notation, or "?" for an ambiguous time (could be AM or PM).
keyExpr	Javascript expression whose value is the key identifying this user's voice print.
mode	Action to take if a voice print exists for the key specified by the keyExpr attribute. *Optional* (default is "adapt"). • delete - Delete the existing voice print and start creating a new voice print. • adapt - Refine the existing voice print with the user's response. • skip - Skip execution of the <register> tag.

Usage

Parents	Children
<form>	<audio> <value> <catch> <help> <noinput> <nomatch> <error> <filled> <grammar> <prompt> <property>

See Also

- Related tag:
  “<verify>” on page 16

Examples

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

<!DOCTYPE vxml PUBLIC "-//BeVocal Inc//VoiceXML 1.0//EN"
"http://cafe.bevocal.com/libraries/dtd/vxml1-0-bevocal.dtd">

<vxml version="1.0" >
  <form id="register_user">
    <field name="account" type="digits">
      <prompt>What is your account number.</prompt>
      <filled>
        You will now be registered as the only authorized
        caller for this account.
      </filled>
    </field>
    <!-- ********************************************************** -->
    <!-- * Use the account number as the speaker's key. Create a   * -->
    <!-- * new voice print for that key, deleting any  voice print * -->
    <!-- * that already exists for the key.                        * -->
    <!-- ********************************************************** -->
    <register name="phone1" type="phone" keyExpr="account" mode="delete">
      <prompt>Please say your telephone number.</prompt>
    </register>
    <register name="phone2" type="phone" keyExpr="account">
      <prompt>Please repeat your telephone number.</prompt>
      <filled>
        <if cond="phone1 != phone2">
          <prompt>
            You did not say the same phone number both times.
            Let's try again.
          </prompt>
          <clear namelist="phone1 phone2"/>
        </if>
    </register>
    <register name="numbers" type="digits" keyExpr="account">
      <prompt>
        Please say one two three four five,
        one two three four five
      </prompt>
    </register>
    <register name="color" keyExpr="account">
      <grammar> [red blue yellow green] </grammar>
      <prompt>
        Please say one of the the colors red, blue, yellow, or green.
      </prompt>
    </register>
  </form>
</vxml>
```

<verify>

Verify that the speakers voice matches a stored voice print.

Syntax

```
<verify>
    name="string"
    type="boolean"|"date"|"digits"|"currency"|
         "number"|"phone"|"time"
    keyExpr="js_expression"
  Child Elements
</verify>
```

Description

Field item that prompts for a value matching a particular grammar and compares the user's voice against the stored voice print for the specified key. Throws an `error.verify.keynotfound` event if no voice print has been registered for that key.

Any utterances that match a universal grammar or a grammar in document or application scope are recognized as they would be in a `<field>` element. However, only those utterances that match the grammar of the `<verify>` element itself are used to verify the speaker's identity.

Any DTMF input is rejected and results in a nomatch event.

Note: Unlike other field items, a `<verify>` element is not a possible go-back destination for the BeVocal go-back facility. That is, the user cannot say "go back" to return to a `<verify>` element, retracting earlier input to that element. See VoiceXML Reference for a description of the go-back facility.

Attribute	Description
`name`	Name of the field item variable that will hold the recognition result. The variable name may not be a JavaScript reserved keyword. The field item variable has dialog (form) scope.

Attribute	Description
type	Specifies an internal grammar. *Optional* (as alternative to a child <grammar> element). • boolean - Grammar for recognizing positive and negative responses. Returns "true" for yes and "false" for no. • date - Grammar for recognizing dates. Returns string with format "*yyyymmdd*"; "????" is used for an unknown year and "??" is used for an unknown month or day.; • digits - Limited grammar for recognizing a sequence of digits. Returns a string of digits. • currency - Grammar for recognizing amounts of money, in dollars (*Not Implemented*: International currency designation). Returns a string with format "*mm.nn*". • number - More general grammar for recognizing numbers. Returns a string that could include digits, a decimal point, or positive or negative sign. • phone - Grammar for recognizing a telephone number adhering to the North American Dialing Plan (with no extension). Returns a sequence of digits. • time - Grammar for recognizing a time. Returns a string with format "*hhmmx*" where and *x* is one of: "a" for AM, "p" for AM, "h" for 24 hour notation, or "?" for an ambiguous time (could be AM or PM).
keyExpr	Javascript expression whose value is the key identifying this user's voice print.

Corresponding to the field item variable *name* is a "shadow variable" whose name is *name$*. The verifier returns the result of the verification in the decision property of this shadow variable. The possible results are:

- accepted - The speaker's voice matches the voice print.
- rejected - The speaker's voice does not match the voice print.
- unsure - The verifier could not determine whether the speaker's voice matches the voice print.

For a <verify> element whose name is *name*, you access the decision property with the syntax:

```
name$.decision
```

Your application can use the value of the decision property to decide how to proceed. For example, if the value is "unsure", it could repeat the verification step.

Usage

Parents	Children
<form>	<audio> <value> <catch> <help> <noinput> <nomatch> <error> <filled> <grammar> <prompt> <property>

See Also

- Related tag:
“<register>” on page 13

Examples

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

<!DOCTYPE vxml PUBLIC "-//BeVocal Inc//VoiceXML 1.0//EN"
"http://cafe.bevocal.com/libraries/dtd/vxml1-0-bevocal.dtd">

<vxml version="1.0" >
  <form id="verify_user">
    <field name="account" type="digits">
      <prompt>What is your account number.</prompt>
    </field>

    <verify name="check1" type="phone" keyExpr="account">
      <prompt>Please say your telephone number.</prompt>
      <filled>
        <if cond="check1$.decision==’accepted’">
          <prompt>
            You have been verified as the authorized caller for
            account number <value expr="account"/>.
          </prompt>
        <elseif cond="check1$.decision==’rejected’" />
          <prompt>
            Sorry. You are not the authorized caller for
            account number <value expr="account"/>.
          </prompt>
          <exit/>
        <elseif cond="check1$.decision==’unsure’" />
          <prompt>Unable to verify that you are the authorized caller for
            account number <value expr="account"/>. Let’s try again.
          </prompt>
          <clear namelist="check1"/>
        </if>
      </filled>
    </verify>
```

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

3 Demonstration

This chapter presents a demonstration application that can be used to test the training and authentication steps.

The application asks the user to enter and confirm a 4-digit number, which is then used as the key for storing the user's voice print.

Note: The BeVocal Café VoiceXML Interpreter combines a Café account-specific value with any specified key and uses the combination to identify a voice print. As a consequence, VoiceXML applications running in under two different accounts can use the same key (for example, 1234) to identify different users.

If a voice print is already registered for the key, the user skips the training dialog and goes directly to authentication. To effect this flow of control, the first `<register>` element has its `mode` attribute set to `"skip"`. If no voice print is registered for the user's key, this element registers a new voice print and sets the `user_status` variable to `"new"`. After that, the application is redirected to the authentication dialog if the `user_status` variable still has the initial value of `"old"`.

Testing Speaker Verification

You can run this demo from a BeVocal Café account and experiment to test how BeVocal Speaker Verification works. Try these tests with a number of different speakers:

- Have one user register a voice print in the training dialog.Then have different people (imposters) call and provide the same key.

  If an imposter is accepted by the authentication dialog, a *false acceptance* has occurred.

- Have the same person register a voice print and go through the authentication dialog a number of times.

  If the same person who provided the voice print (the "true speaker") is rejected during authentication, a *false rejection* has occurred.

As different speakers experiment with the demonstration application, try to get a sense of how frequently an imposter was accepted and how frequently a true speaker was rejected.

For some applications, false acceptance is a more serious concern. To maintain security, it is better to reject an authorized user occasionally than to allow an unauthorized user to access sensitive information. For other applications, security it not as crucial and the inconvenience of rejecting an authorized user outweighs any problems caused by accepting an unauthorized user.

The verification mechanism can be adjusted to reduce either false acceptance or false rejection. When the verifier attempts to minimize false acceptance, however, false rejection increases and vice-versa. The BeVocal Café VoiceXML Interpreter uses settings that try to achieve a balance, favoring a lower incidence of false

acceptance. We are interested in knowing whether these settings are appropriate for your application.

BeVocal encourages you to try this experimental VoiceXML extension for Speaker Verification, and is very interested in your feedback! Please post your thoughts, observations, and suggestions to the BeVocal Café News groups.

Sample Interactions

In this interaction with the demonstration, the user's registers a voice print and is verified to be the authentic user.

Application: Welcome to the Speaker Verification demo.
Please say or enter the last 4 digits of your telephone number. This number will identify you for access to the secure application.

User: 3 0 4 2

Application: I heard three zero four two. Is that correct? Say Yes or No.

User: Yes

Application: Okay, Let's continue.
I need some samples of your speech to create an identifying voice print. Please say your 10-digit telephone number.

User: 408-656-3042

Application: Please say any 5 digit number and repeat it.

User: 2 4 5 8 9 2 4 6 8 9

Application: Your voice print has been registered.
Do you wish to continue with authentication?

User: Yes

Application: Please say your 10-digit telephone number.

User: 408-656-3042

Application: Your identity has been verified.
Thanks for using the secure application. Good bye.

The same user calls back and goes through the authentication step only.

Application: Welcome to the Speaker Verification demo.
Please say or enter the last 4 digits of your telephone number. This number will identify you for access to the secure application.

User: 3 0 4 2

Application: I heard three zero four two. Is that correct? Say Yes or No.

User: Yes

Application: A voice print has already been registered for the identifying number three zero four two. Let's see if your voice matches that voice print.

Application: Please say your 10-digit telephone number.

User: 408-656-3042

Application: Your identity has been verified.

Thanks for using the secure application. Good bye.

Next, an impostor calls and is rejected by the authentication step.

Application: Welcome to the Speaker Verification demo.

Please say or enter the last 4 digits of your telephone number. This number will identify you for access to the secure application.

User: 3 0 4 2

Application: I heard three zero four two. Is that correct? Say Yes or No.

User: Yes

Application: A voice print has already been registered for the identifying number three zero four two. Let's see if your voice matches that voice print.

Application: Please say your 10-digit telephone number.

User: 408-656-3042

Application: Please say any five digit number.

User: 1 2 3 4 5

Application: I'm still not sure of your identity. Lets try again

Please say your 10-digit telephone number.

User: 408-656-3042

Application: Please say any five digit number.

User: 8 9 7 8 9

Application: Sorry. You are not authorized to access the secure application. Goodbye

Application Code

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

<!DOCTYPE vxml PUBLIC "-//BeVocal Inc//VoiceXML 1.0//EN"
"http://cafe.bevocal.com/libraries/dtd/vxml1-0-bevocal.dtd">

<vxml version="1.0" >

  <!-- ********************************************************** -->
  <!-- * Variables:                                             * -->
  <!-- *    key will be set to the user's identifying key.      * -->
  <!-- *    user-status will be set to 'old' if a voice print   * -->
  <!-- *        already exists for this user and 'new' if a new * -->
  <!-- *        voice print is created for the user.            * -->
  <!-- ********************************************************** -->
  <var name="key" />
  <var name="user_status" expr="'old'" />

  <!-- ********************************************************** -->
  <!-- * Training Dialog                                        * -->
  <!-- ********************************************************** -->
  <form id="training_form">
    <block>
      <prompt>
        Welcome to the Speaker Verification demo
      </prompt>
    </block>

    <!-- ***************************************************** -->
    <!-- * Ask the user for a 4-digit number, which will be  * -->
    <!-- * used as a key for Speaker Verification.           * -->
    <!-- ***************************************************** -->
    <field name="phone4" type="digits">
      <prompt>
        Please say or enter the last 4 digits of your telephone number.
        This number will identify you for access to the
        secure application
      </prompt>
    </field>

    <field name="confirm_key" type="boolean">
      <prompt> I heard <value expr="phone4"/> </prompt>
      <prompt> Is that correct? Say Yes or No </prompt>
      <filled>
        <if cond="confirm_key">
          <prompt> Okay, Let's continue </prompt>
          <assign name="key" expr="phone4"/>
        <else/>
          <clear namelist="phone4 confirm_key"/>
        </if>
      </filled>
    </field>
```

```
<!-- ******************************************************** -->
<!-- * Create a new voice print unless one is already       * -->
<!-- * registered for this user's key.                      * -->
<!-- ******************************************************** -->
<register name="register_step1" type="phone" keyExpr="key" mode="skip">
  <prompt>
    I need some samples of your speech to create an
    identifying voice print. Please say your 10-digit telephone number
  </prompt>
  <filled>
    <assign name="user_status" expr="'new'"/>
  </filled>
 </register>
<!-- ******************************************************** -->
<!-- * Proceed to authentication if the user already had a  * -->
<!-- * registered voice print.                              * -->
<!-- ******************************************************** -->
<block>
  <if cond="user_status == 'old'">
    <prompt>
      A voice print has already been registered for
      the identifying number <value expr="key"/>.
      Let's see if your voice matches that voice print
    </prompt>
    <goto next="#authentication_form"/>
  </if>
</block>
<!-- ******************************************************** -->
<!-- * Refine the  voice print with another speech sample.  * -->
<!-- ******************************************************** -->
<register name="register_step2" type="digits" keyExpr="key">
  <prompt>
    Please say any 5 digit number and repeat it.
  </prompt>
</register>

<filled namelist="register_step1 register_step2">
  <prompt>
    Your voice print has been registered
  </prompt>
</filled>

<field name="confirm_continue" type="boolean">
  <prompt>
    Do you wish to continue with authentication?
  </prompt>
  <filled>
    <if cond="confirm_continue">
      <goto next="#authentication_form"/>
    <else/>
      <prompt>
        Thank you for registering to use this application.
      </prompt>
```

```
        <exit/>
      </if>
    </filled>
  </field>
</form>

<!-- ********************************************************** -->
<!-- * Authenticaion Dialog                                   * -->
<!-- ********************************************************** -->
<form id="authentication_form" >
  <verify name="compare1" type="phone" keyExpr="key">
    <prompt>
      Please say your 10-digit telephone number
    </prompt>
    <filled>
      <if cond="compare1$.decision == 'accepted'">
        <prompt>
          Your identity has been verified
        </prompt>
        <goto next="#secure_form">
      <elseif cond="compare1$.decision == 'rejected'" />
        <prompt>
          Sorry you are not authorized to access
          the secure application. Goodbye
        </prompt>
        <exit/>
      </if>
    </filled>
  </verify>

  <verify name="compare2" type="digits" keyExpr="key">
    <prompt>
      Please say any five digit number
    </prompt>
    <filled>
      <if cond="compare2$.decision == 'accepted'">
        <prompt>
          Your identity has been verified
        </prompt>
        <goto next="#secure_form">
      <elseif cond="compare2$.decision == 'rejected'" />
        <prompt>
          Sorry. You are not authorized to access
          the secure application. Goodbye
        </prompt>
        <exit/>
      <elseif cond="compare2$.decision == 'unsure'" />
        <prompt>
          I'm still not sure of your identity. Lets try again
        </prompt>
        <clear namelist="compare1 compare2"/>
      </if>
    </filled>
  </verify>
</form>
```

```
  <!-- ********************************************************** -->
  <!-- * Simulated Secure Interaction                           * -->
  <!-- ********************************************************** -->
  <form id="secure_form" >
    <block>
      <prompt>
        Thanks for using the secure application. Good bye.
      </prompt>
    </block>
  </form>
</vxml>
```