VoiceXML Reference

BEVOCAL

Version 1.6
June 2001

BeVocal, Inc.
1380 Bordeaux Drive
Sunnyvale, CA 94089

Table of Contents

Preface

VoiceXML is a markup language for writing telephone-based speech applications. This document describes BeVocal's VoiceXML, which is compliant with the W3C VoiceXML Version 1.0 Specification.

Audience

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

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
- Filenames and pathnames

`Italic fixed width` font is used for:

- Variables

Italic font is used for:

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

How to Use This Guide

The first part of this guide explains who to use VoiceXML features:

- Chapter 1, "Getting Started" introduces VoiceXML and its major features
- Chapter 2, "Forms" describes VoiceXML forms.
- Chapter 3, "Event Handling" describes events that can be thrown during the execution of a VoiceXML application and how events are handled.
- Chapter 4, "Fetching Resources" explains how an application can control the way VoiceXML documents and other resources are fetched and cached.

- Chapter 5, “Go-Back Facility” describe the BeVocal Café *go-back facility*, an experimental extension to VoiceXML.

A new application developer typically reads these chapters complete and in order.

The remainder of this guide provides reference descriptions of the various components of the VoiceXML language:

- Chapter 6, “Tag Reference” describes the tags that make up VoiceXML.
- Chapter 7, “Property Reference” describes the properties that can be set to control the behavior of a VoiceXML application.
- Chapter 8, “Variable Reference” describes predefined variables that are available in VoiceXML applications.
- Chapter 9, “Function Reference” describes predefined JavaScript functions that are available in VoiceXML applications.

Application developers typically do not read these chapters from start to finish, but instead use them to look up information about the various tags, properties, and so on.

References

For additional or related information, you can refer to:

- VoiceXML Version 1.0 Specification. VoiceXML Forum.
  (http://www.w3.org/TR/2000/NOTE-voicexml-20000505)
- VoiceXML Tag Summary. BeVocal.
  (http://cafe.bevocal.com/docs/tagSummary/index.html)
- Grammar Reference. BeVocal.
  (http://cafe.bevocal.com/docs/grammar/index.html)
- JavaScript Quick Reference. BeVocal.
  (http://cafe.bevocal.com/docs/javascript_quick_reference/index.html)
- SpeechObjects Quick Reference. BeVocal.
  (http://cafe.bevocal.com/docs/so_quick_reference/index.html)
- Geo API Quick Reference. BeVocal.
  (http://cafe.bevocal.com/docs/geo_api/index.html)

1 Getting Started

VoiceXML is a markup language derived from XML for writing telephone-based speech applications. Users call applications by telephone. They listen to spoken instructions and questions and provide input using the spoken word, as opposed to viewing a screen display and entering information with a keyboard or mouse.

VoiceXML

Just as a web browser renders HTML documents visually, a VoiceXML interpreter renders VoiceXML documents audibly. You can think of the BeVocal VoiceXML interpreter as a telephone-based voice browser.

As with HTML documents, VoiceXML documents have web URLs and can be located on any web server. Yet a standard web browser runs locally on your machine, whereas the VoiceXML interpreter is run remotely — at the BeVocal hosting site, for example. And you use your telephone to access the VoiceXML interpreter.

Tags and Elements

VoiceXML uses markup tags and plain text. A *tag* is a keyword enclosed by the angle bracket (< and >) characters. A tag may have *attributes* inside the angle brackets. Each attribute consists of a *name* and a *value*, separated by an equal (=) sign; and the value must be enclosed in quotes.

Tags occur in pairs; corresponding to the start tag `<`*`keyword`*`>` is the end tag `</`*`keyword`*`>`. Between the start and end tag, other tags and text may appear. Everything from the start tag to the end tag, is called an *element*. For example, the following three lines constitute a prompt element:

```
<prompt>
  What is your telephone number?
</prompt>
```

If there are no other tags or text between the start and end tag, a syntactic shorthand is permitted. You can omit the end tag by replacing the final `">"` of the start tag with `"/>"`. For example, instead of writing a value element as:

```
<value expr="result"></value>
```

you can use the shorthand notation:

```
<value expr="result"/>
```

Because the syntax specifies the end of each element, the VoiceXML interpreter can check that the entire document has been received.

If one element contains another, the containing element is called the *parent* element of the contained element. The container element is called a *child* element of its containing element. The parent element may also be called a *container*.

Although both HTML and VoiceXML use markup tags, the two languages use tags differently. Whereas the markup tags in HTML describe how to render the data, the markup tags in XML (and consequently in VoiceXML) describe the data itself. This allows an XML interpreter or browser to display the data in whatever way is appropriate.

BeVocal's VoiceXML generally complies with the VoiceXML 1.0 Specification. It also includes several handy extensions that you can use if you choose. VoiceXML Tag Summary lists any differences between VoiceXML and the standard.

Tip:

VoiceXML conforms to XML standards, so the formats for VoiceXML tags are more strictly defined than are the formats in HTML. if you are used to HTML and not XML, remember that all container elements require end tags and all attribute values must be in quotes.

Simple Example

In VoiceXML, the `<form>` element is analogous to an HTML form that contains items for the user to enter. In VoiceXML forms, each logical piece of information to be collected from the user is identified with a `<field>` tag.

In the following example, the form collects one piece of information from the user. Once this information is obtained, execution proceeds to the field's `<filled>` element.

```
<?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">

  <!-- Phone to city/state example application -->
  <form>
    <!-- Variable to hold result -->
    <var name="result">

    <field name="phoneNumber" type="phone">
      <prompt>please say your work phone number</prompt>
      <filled>
        <assign
          name="result"
          expr="geoApi.PhoneToCityState(phoneNumber)"/>
        <prompt>
          You work in <value expr="result"/>
          The city is <value expr="result.city"/>
          The state is <value expr="result.state"/>
        </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

Other tags used in the example include:

- The `<var>` tag declares a variable to be used within the form.
- The `<prompt>` tag requests user input.
- The `<assign>` tag assigns a value to a variable.
- The `<value>` tag evaluates an expression.

VoiceXML contains no explicit instructions about how to present the prompt, “please say your phone number” or how to present the results. In theory, these could be presented textually on a different kind of browser.

In practice, this document is run as a telephone application and would result in a conversation such as the following:

Application:	Please say your work phone number.
User:	My work number is 408-907-3200.
Application:	You work in Sunnyvale, California. The city is Sunnyvale. The state is California.

Environment

In order to support a telephone interface, the VoiceXML interpreter runs within an execution environment that includes a telephony component, a text-to-speech (TTS) speech-synthesis component, and a speech-recognition component.

The VoiceXML interpreter transparently interacts with these infrastructure components as needed. For example:

- Text strings in output elements are rendered using TTS.
- Telephone connection issues (picking up the incoming call, detecting a hang-up, transferring a call) are handled by the telephony component.
- Listening to spoken input from the user and identifying its meaning is handled by the speech-recognition component.

Documents

An executable VoiceXML file is called a *document*. The VoiceXML interpreter loads a document file to execute it.

Every VoiceXML document must start with header information that conforms to the XML standard:

```
<?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">
```

These headers describe the language in which the document is written:

- The first tag indicates that the document is an XML document.

- The second tag identifies the Document Type Definition (DTD), which is used to validate that the contents represent well-formed VoiceXML.

  A DTD describes the format of the data that might appear in an XML document. That is, the DTD defines the valid tags by specifying what attributes each tag can have and what child tags or other content each tag can contain.

  BeVocal's VoiceXML is described by its DTD, which is identified in the second header tag given above.
- The third tag identifies the version of VoiceXML used in this document.

Apart from headers and possibly comments, all the content in a VoiceXML document is contained within a vxml element, that is, between the `<vxml version="1.0">` start tag and the `</vxml>` end tag.

Applications

A VoiceXML *application* consists of one or more documents. Any multidocument application has a single *application root document*. Each document in an application identifies the application root document with the `application` attribute of the `<vxml>` tag:

```
<vxml version="1.0" application="myAppRoot.vxml">
```

Whenever the interpreter executes a document, it loads that document. If the document specifies an application root document, that document is also loaded.

You can use an application root document for global items or interactions that you want to be active throughout the application. For example, suppose the application root document `myAppRoot.vxml` declares a variable named `company` that has an initial value of `BeVocal`:

```
<vxml version="1.0">
  <var name="company" expr="BeVocal">
  ...
```

This variable has *application scope*. That is, any document in the application can use the variable. If the variable's name is ambiguous, or if you want to make it explicit that the variable is defined in application scope, you can qualify the variable name by prepending `"application."` when you use the variable in other documents:

```
<value expr="application.company">
```

Dialogs

Within a document, a user interacts with *dialogs*, in which the application produces auditory output, typically asking for information, and the user provides input by speaking or pressing keys on the telephone. VoiceXML has two kinds of dialogs: forms and menus.

- A *form* interacts with the user to fill in a number of *fields*. Every field has an associated variable, called its *field item variable*. Initially, the variable has a value of `"undefined"`. It is filled in when the speech-recognition engine recognizes a valid response in a user utterance.
- A *menu* presents the user with a number of choices; it transitions to a different dialog based on the user's selection.

Forms

The VoiceXML `<form>` tag defines a form and the `<field>` tag defines a field in a form. You specify the name of the field item variable with the `name` attribute of the `<field>` tag. You can use the field variable's name in expressions to refer to the stored value.

In the preceding example, the field item variable is named `phoneNumber`:

```
  <field name="phoneNumber" type="phone">
```

When the user says the phone number, the number is stored in the `phoneNumber` variable. Then the interpreter proceeds to execute the field's `<filled>` element. Here, the `phoneNumber` variable in the `<assign>` element is evaluated before being passed as the parameter to the `geoApi.PhoneToCityState` utility function (which converts a phone number to a city/state pair).

```
<assign
    name="result"
    expr="geoApi.PhoneToCityState(phoneNumber)"/>
```

(More information on this BeVocal utility is available in the Geo API Quick Reference.)

Menus

The `<menu>` tag defines a menu; each choice consists of a `<choice>` element. The `next` attribute of a `<choice>` element specifies the destination dialog to which the interpreter should transition when the user selects that choice. If a `<form>` or `<menu>` element is to be the destination of a transition, it can be given a name with the an `id` attribute.

For example, the following menu consists of three choices.

```
<menu>
  <prompt>
    Please choose one of <enumerate/>
  </prompt>
  <choice next="#MovieForm">
    local movies
  </choice>
  <choice next="localBroadcast.vxml#RadioForm">
    local radio stations
  </choice>
  <choice next="http://www.nationTV.org/tv.vxml">
    national TV listings
  </choice>
</menu>
```

The prompt in this menu includes an `<enumerate>` tag. This tag lets you set up a template for an automatically generated description of the choices. By default, the `<enumerate>` template simply lists all the choices. In the above example, the prompt is "Please choose one of local movies, local radio stations, national TV listings."

The destination dialog specified by the `next` attribute can be in the current document or in a different document:

- If the user says "local movies", the interpreter transitions to the dialog named `MovieForm` in the same document.

- If the user says “local radio stations”, the interpreter transitions to the named `RadioForm` in the document `localBroadcast.vxml`.
- If the user says “national TV listings”, the interpreter transitions to the first dialog in the document `tv.vxml` in the national TV web site.

Properties

You can set properties to customize the behavior of the interpreter. The `<property>` tag specifies the property to set and the value for the property.

Various properties control how the interpreter behaves when prompting the user for input, recognizing speech or DTMF input (telephone key tone signals), and fetching documents and other resources. For additional information, see Chapter 7, “Property Reference”.

Grammars

The speech-recognition engine uses *grammars* to interpret user input.

Each field in a form can have a grammar that specifies the valid user responses for that field. An entire form can have a grammar that specifies how to fill multiple field item variables from a single user utterance. Each choice in a menu has a grammar that specifies the user input that can select the choice.

A VoiceXML application can use built-in grammars and application-defined grammars.

Built-in Grammars

Some basic grammars are built into all standard VoiceXML interpreters. You can reference a built-in grammar in two ways:

1. From a `<field>` element, you can use the `type` attribute to refer to a built-in grammar. Our Simple Example uses the built-in `phone` grammar:

   ```
   <field name="phoneNumber" type="phone">
   ```

   This means that the speech-recognition engine will try to interpret what the user says as a telephone number.

2. In a `<grammar>` element, you can use the `src` attribute to specify a URL with the `"builtin:"` prefix. For example:

   ```
   <grammar src="builtin:grammar/boolean"/>
   ```

The standard built-in types include:

- `boolean`
- `date`
- `digits`
- `currency`
- `number`
- `phone`
- `time`

Applications cannot modify built-in grammars. However, two built-in grammars, `digits` and `boolean`, can be parameterized. Specifically, you can set limits on the length of a digit string, and you can set DTMF key presses to mean yes or no.

The `digits` and `boolean` built-in grammars can be parameterized as follows:

Grammar	Parameters
`boolean`	• `y` - The DTMF key press for an affirmative response. • `n` - The DTMF key press for a negative answer.
`digits`	• `minlength` - The minimum number of digits in a valid response. • `maxlength` - The maximum number of digits in a valid response. • `length` - The exact number of digits in a valid response.

You express parameter information using URL-style query syntax of the form:

`typeName ? parameter = value`

More than one parameter may be specified separated by semicolons.

You can supply parameter information in the `type` attribute of a `<field>` element, for example:

```
<field type="digits?lenth=10"/>
```

Similarly, you can supply parameter information in the `src` attribute of a `<grammar>` element, for example:

```
<grammar
  src="builtin:grammar/boolean?y=7;n=9"/>
```

Application-Defined Grammars

Although the built-in grammars can be useful, you typically need to define your own grammars, using the `<grammar>` tag.

An application-defined grammar can be specified in either of two forms:

- Nuance Grammar Specification Language (GSL)
- XML Speech Recognition Grammar Format

See Grammar Reference for additional information about grammars.

A simple grammar can be defined in the document. An *inline grammar* is defined within the `<grammar>` element itself. For example, the following inline GSL grammar matches the words “add” and “subtract”.

```
<field name="operator">
  <grammar> [add subtract] </grammar>
  ...
```

With this grammar, if the user says “add,” the field item variable `operator` is set to `"add"`.

More complex grammars can be written externally. An *external grammar* is defined in a file separate from the VoiceXML document file and is referenced by the `src` attribute of the `<grammar>` element. For example, the following field uses a grammar rule named `Colors` in an external GSL grammar defined in the file `partGrammar.grammar`.

```
<field name="part">
  <grammar
    src="http://www.mySite/partGrammar.grammar#Colors"/>
  ...
```

The named rule (`Colors` in the preceding example) is called the *root rule* for the grammar. The specified file may include other grammar rules, which may be used as subgrammars of the root rule.

The grammar for a menu choice can be specified explicitly with a `<grammar>` child of the `<choice>` element. Alternatively, a grammar can be generated automatically from the choice text. The user can say any of the words in the choice text to select that choice. In the preceding example, the user could say “TV listings” or just “TV” to select the third choice.

```
<menu>
  ...
  <choice ...>
    national TV listings
  </choice>
</menu>
```

However, the words must be spoken in the correct order, so “listings, TV” would not work.

Active Grammars

The speech-recognition engine uses *active* grammars to interpret user input. A field grammar is active whenever the interpreter is executing that field. A menu-choice grammar is active whenever the interpreter is executing the containing menu. A form grammar is active whenever the interpreter is executing the containing form.

A form grammar or the collection of choice grammars in a menu can optionally be made active at higher scopes:

- A grammar with *document scope* is active whenever the interpreter is executing any dialog in the document.
- A grammar with *application scope* is active whenever the interpreter is executing any document in the application.

If the interpreter is executing one dialog and the user’s input matches an active grammar for a different dialog, control transfers to the latter dialog. If the grammar is in application scope, control might transfer to a dialog in a different document.

Note that within a field, you can temporarily turn off grammars from higher scopes by setting the field’s `modal` attribute to `"true"`.

Universal Grammars

Universal grammars are an extension to the BeVocal VoiceXML 1.0 platform.

A *universal grammar* is one that is can be active all the time—that is, whenever the interpreter is executing the application. The interpreter’s predefined `"help"`, `"exit"`, `"cancel"` and `"goback"` grammars are universal grammars.

The application can declare an application-defined grammar to be universal by naming the grammar with the `universal` attribute of the `<grammer>` tag. For example, the following element defines a universal grammar named `"mainmenu"`:

```
<grammar universal="mainmenu">
  main menu
</grammar>
```

In VoiceXML 1.0 applications, all universal grammars are activated by default. The application can choose to deactivate some or all universal grammars by setting the `universals` property. This property specifies which of the universal grammars should be active; all other universal grammars are deactivated.

For example, you might choose to deactivate the `"exit"` and `"cancel"` grammars in your application:

```
<grammar universal="mainmenu">
  main menu
</grammar>
<!-- Make mainmenu, help, and goback active -->
<!-- but deactivate exit and cancel. -->
<property
  name="universals"
  value="help mainmenu goback"/>
```

You can deactivate all universal grammars by specifying the value `"none"`:

```
<property
  name="universals"
  value="none" />
```

All universal grammars are deactivated by default when the `vxml` tag's `version` attribute equals `2.0`. The application can activate some or all universal grammars by setting the `universals` property. To activate all universal grammars, you can put the following element in your application root document:

```
<property
  name="universals"
  value="all" />
```

Events

The VoiceXML interpreter can throw a number of predefined *events* based on errors, telephone disconnects, or user input. For example:

- A *no-input* event is thrown if the user does not respond to a question.
- A *no-match* event is thrown when the user does not respond intelligibly—that is, when the user's utterance does not match any active grammar.
- A *help* event is thrown when the user requests help.
- An *error* event is thrown when any kind of error occurs.

An application can define additional events and can use a `<throw>` element to throw an event of a specified kind.

An application can catch an event and take the appropriate response in an *event handler*. A `<catch>` element is a general-purpose event handler; its `event` attribute specifies the kinds of event that it handles. Additional event-handling tags are syntactic shorthand: `<noinput>`, `<nomatch>`, `<help>`, and `<error>`. Each of these shorthand tags catches one type of event, indicated by its name. For example, a `<nomatch>` element catches no-match events.

When an event is thrown, the associated event handler, if it exists, is invoked. If the handler did not cause the application to terminate, execution resumes in the element that was being executed when the event was thrown.

For more information, see Chapter 3, "Event Handling".

Links

A *link* specifies a grammar that is independent of any particular dialog.

A `<link>` element defines a link. Each `<link>` element contains a `<grammar>` element. A link's grammar is active in the scope of the element that contains the link. For example, if the link is in a form, its grammar is active when the interpreter is executing that form. If a link is under a `<vxml>` element, its grammar has document scope; if the link is in the application root document, its grammar has application scope. Links in a `<vxml>` element can implement global behaviors.

A link can specify one of two possible actions to take if the speech-recognition engine detects a match its grammar:

- The link can cause a transition to a different location; in that case, its `next` attribute specifies the destination of the transition. Links, like menu choices, can cause transitions to other dialogs or documents.
- The link can throw an event; in that case, its `expr` attribute specifies the event to throw. After the event is handled execution resumes with the element that was being executed when the link grammar was matched.

For example, the following link is defined at document level, so its grammar (specified in GSL) is active whenever the interpreter is executing any dialog in the document. If the user says "operator," the link transfers control to a different document.

```
<vxml>
  <link next="operator_xfer.vxml">
  <grammar> operator </grammar>
</link>
...
```

Procedural Logic

You can use procedural logic, called *executable content*, within a few basic elements: `<block>`, `<filled>`, and event handlers. Within executable content, you can declare and assign values to variables, use simple conditional logic, output speech or audio to the user, or run a JavaScript.

Variables

Variables are declared by the `<var>` tag. Declarations can appear in a document, a form, or executable content. The `<var>` tag can optionally specify the variable's initial value; if it doesn't, the variable will be initialized to `"undefined"`.

A variable has the scope of the element that contains the declaration:

- A variable has *document scope* if it is declared in a `<vxml>` element, or in a `<block>` or event handler that is a child of the `<vxml>` element. If the document is the application root document, then the variable has *application scope*.

  You can refer to a variable `x` with document scope either as `x` or `document.x` (for clarity or to resolve ambiguity). If the variable is in the application root document, then you can refer to it in other documents as `application.x`.
- A variable has *dialog scope* if it is declared in a `<form>` element, or in a `<block>` or `<filled>` element or an event handler that is a child of a `<form>` element.

  You can refer to a variable `x` with dialog scope either as `x` or `dialog.x`.

- A variable has an *anonymous scope,* local to a field, if it is declared in an event handler or `<filled>` element that is a child of a `<field>` element.

If a `<var>` element specifies a variable that is already in scope, it does not declare a new variable with the same name, but simply assigns a value to the existing variable. If the `<var>` element has an `expr` attribute, the variable is assigned the specified value; otherwise, the variable is assigned the value `"undefined"`.

You can set a variable's value with the `<assign>` tag.

VoiceXML variables are in all respects equivalent to JavaScript variables—they are part of the same variable space. For additional information, see "Scripts" on page 23.

Conditional Logic

You can use an `<if>` element to execute a block of code if a condition is satisfied. Within that element, you can use a sequence of `<elseif>` elements to execute alternative blocks of code if all previous conditions failed and the condition of the `<elseif>` element is satisfied. You can use an `<else>` element to execute and alternative block of code if all previous conditions failed.

The conditions in `<if>` and `<elseif>` elements are expressed as Boolean-valued JavaScript expressions.

Output

A `<prompt>` or `<reprompt>` element generates speech output; an `<audio>` element plays a prerecorded audio clip.

Prompts can appear in executable contents, as well as in elements for collecting user input. Anywhere a `<prompt>` is valid, text is interpreted as a prompt even if the enclosing `<prompt>` and `</prompt>` tags are omitted.

A field item and the `<initial>` item of a mixed-initiative form has a *prompt counter* that lets you play different prompts if the user revisits the item several times. For example, you may want to play shorter descriptions after the first or second time the user is prompted for the same information. The prompt counters are reset on each form invocation.

Scripts

A `<script>` element executes a JavaScript, which is run in the scope of the parent element. A `<script>` element can also define functions that can be called by JavaScript expressions in the same scope.

VoiceXML variables are equivalent to JavaScript variables and are part of the same variable space. VoiceXML variables can be used in a script just as variables defined in a `<script>` element can be used in VoiceXML. Declaring a variable using a `<var>` element is equivalent to using a `var` statement in a `<script>` element.

User Interaction

VoiceXMLsupports both application-directed and mixed-initiative interactions with a user.

In an *application-directed* (or simply *directed*) interaction, the application prompts for the information it needs and the user supplies the requested information by answering the prompts. The application controls the interaction; the user cannot volunteer information. To be more accurate, the application does not understand volunteered information:

- If the application is executing a form, the only active grammar is the one for the current field of the form. The only valid user input is one that provides a value for the current field’s variable.
- If the application is executing a menu, the only active grammars are the grammars of the menu’s choices. The only valid user input is one that selects a choice for the current menu.

In a *mixed-initiative* interaction, the user and the application both participate in determining what the application does next. A single utterance from the user may provide input for multiple field item variables in a form. In response to a prompt in one dialog, the user may provide input that matches a grammar defined in a different form. When this happens, the interpreter transitions to that dialog and fills its field item variables from the user input. Similarly, the user may provide input that selects a choice from a different menu or that matches a link grammar, causing a transition to the destination specified by that choice or link.

If an application does not use links or grammars with document or application scope, it may still include mixed-initiative forms. A *mixed-initiative form* includes a form grammar. It can include an `<initial>` element to control the initial interaction in the form. This element can request user input or perform other non-interactive initialization tasks. In response to a prompt from the `<initial>` element, the user could provide input that fills in multiple field item variables. If the form prompts for individual fields, any user input that matches the form grammar is valid—even if that input does not fill in the field for which the user was just prompted.

Note: Fewer speech-recognition errors occur in directed interactions than in mixed-initiative interactions.

Flow of Execution

Execution within a VoiceXML document flows in document order until a dialog (form or menu) is entered. Execution flows from the current dialog to a different dialog or document, based on either:

- An explicit transition statement in the current dialog.
- Speech recognition in the current dialog that causes a transition to a different dialog.

In addition, execution can temporarily leave the current dialog to execute a subdialog, returning to the current dialog when execution of the subdialog is complete.

If the current dialog completes execution without transitioning to a different location, the application exits. In addition, you can use an `<exit>` element to end the application explicitly.

Explicit Transition

You can set up explicit transitions to other dialogs or documents in your application using `<goto>` or `<submit>` tags. These transition elements can be placed inside `<block>` or `<filled>` elements or event handlers.

The `<goto>` element lets you transition to another field item in the current form, to another dialog in the current document, or to another document. When you make the transition to the new location, the local variables from the old form or document are lost. This happens even if you transition to the same form you were in before. However, the values of local variables are not affected when you use `<goto>` to transition between items *within* a form.

The `<submit>` tag lets you pass values to another document using an HTTP GET or POST request. Since you use a URL to specify the next document, it need not be a VoiceXML document; for example, it could be a CGI script document.

Recognition-Triggered Transition

User input to a dialog may cause a transition to a different location:

- If the speech-recognition engine matches a grammar with document or application scope that is defined in a different dialog, the interpreter transitions to that dialog.
- If the speech-recognition engine matches the grammar of a `<link>` element that has a `next` attribute, the interpreter transitions to the destination specified by the `next` attribute.

Subdialogs

A subdialog is a reusable VoiceXML dialog that you can pass data to and get return values from:

- The current dialog passes control to a subdialog with a `<subdialog>` element. It can pass data to the subdialog with `<param>` elements inside the `<subdialog>` element.
- A subdialog returns control to the calling dialog with the `<return>` element. it can pass values back using the `namelist` attribute of the `<return>` element.

2 Forms

The main elements of a document (within the `<vxml>` element) are forms. VoiceXML forms are analogous to Web forms; you use them to collect (voice) input from the user.

Form Items

So far, the only form item we've discussed is the `<field>` element. However, forms can contain either field items or control items:

Field items are elements for collecting user input or results. Field item is any one of the following:

- A field, defined with the `<field>` tag, asks the user for a piece of information.
- A record item, defined with the `<record>` tag, records what the user says (perhaps for a voicemail message);
- A object item, defined with the `<object>` tag, invokes a complex, reusable speech component that can gather user input.
- A subdialog, defined with the `<subdialog>` tag, invokes a reusable dialog.
- A transfer item, defined with the `<transfer>` tag, transfers the user to another telephone number.

Control items are tags that can contain procedural items for audio output or computation. A control items is either of the following:

- A block, defined with the `<block>` tag, is a container for procedural elements.
- An initial item, defined with the `<initial>` tag, controls the initial interaction of a mixed-initiative form.

Form Item Variables

Each form item has an associated *form item variable*. When a form is entered, all form item variables are initially undefined. When a form item is visited, its variable is set to the result of interpreting that form item. For example, visiting a `<block>` element sets its form item variable `"true"`. The form item variable for a field item is also called a *field item variable*; after a field item is visited, its field item variable is set to the value collected from the user.

Execution of a Form

Within a form, the flow of execution is governed by the *Form Interpretation Algorithm* (FIA), a looping algorithm. One each iteration, the FIA selects the form item to visit next.

A form item's *guard conditions* determine whether it can be selected on a given iteration:

- The value of the form item variable must be `"undefined"`.
- The value of any `cond` expressions contained in the form item must evaluate to `"true"`.

Both guard conditions must be met in order for a form item to be selected. The FIA examines the form items in document order, selecting the first one whose guard conditions are met. If the guard conditions for all form items fail, the form (and the application) exits.

By default, every form item variable has an initial value of `"undefined"` so every form item that does not specify a `cond` expression is eligible for selection. After the form item is visited, its variable is set to a value, which prevents the same form item from being selected again on the next iteration.

You can explicitly control the execution of any form item if you give its variable a name and an initial value other than `"undefined"`. Doing so prevents the form item from being eligible for selection until you explicitly use the `<clear>` tag to reset its variable. Typically, field item variables are given names but control item variables are not.

User Interaction

User interaction with a form can be directed or mixed initiative.

A *directed form* has no form grammar, only grammars for its individual fields. A directed form gives the user explicit directions about what to say and when. For example, a directed form might result in the following dialog:

Application:	Would you like to buy, sell, or receive a stock quote?
User:	Get a quote.
Application:	What stock or stocks would you like a quote for?
User:	Intel.

A form that includes its own grammar is a mixed-initiative form. The form grammar allows several field item variables to be filled in as a result of a single user utterance. A mixed-initiative form allows the user to speak more naturally. For example, a mixed-initiative form might result in the following dialog:

Application:	Stock assistant here. How can I help you?
User:	I'd like to get a quote for Intel.

One disadvantage of mixed-initiative forms is that form grammars are more complicated and can result in more recognition errors.

The grammar for a field sets a value for the field’s variable. For example, the grammar in the following field, specified in GSL, assigns the value `"june"` to the variable `month` if the user says “June.”

```
<field name="month">
  <grammar>
    [june july august]
  </grammar>
<field>
```

The grammar for a form must specify both the field item variable to be set by a grammar rule and the value for that variable. For example, the GSL grammar in the following file, `foo.grammar`, sets values for two variables, `quantity` and `fruit`.

```
Main [
  (?Quantity Fruit)
  (Quantity ?Fruit)
  (Quantity Fruit) ]
Quantity [
  one {<quantity "one">}
  two {<quantity "two">} ]
Fruit [
  [apple apples] {<fruit "apples">}
  [orange oranges] {<fruit "oranges">} ]
```

This grammar is used by the following mixed-initiative form:

```
<form id="foo">
  <initial>
    <grammar src="foo.grammar#Main"/>
    How many apples or oranges do you want?
  </initial>
  <!-- If user doesn’t respond to initial -->
  <!-- prompt, ask for each field -->
  <field name="fruit"/>
    <grammar src="foo.grammar#Fruit"/>
    Do you want apples or oranges?
  </field>
  <field name="quantity">
    <grammar src="foo.grammar#Quantity"/>
    How many <value expr="fruit"/> do you want?
  </field>
</form>
```

3 Event Handling

The VoiceXML interpreter can throw a number of predefined events based on errors, telephone disconnects or user requests. You can also throw events you define that are specific to your application. When an event is thrown, the associated event handler, if it exists, is invoked. Then execution resumes in the element that was being executed when the event was thrown.

Predefined Events

The following standard events are predefined:

- `exit` - The user asked to exit.
- `help` - The user asked for help.
- `noinput` - The user did not provide timely input.
- `nomatch` - The user did not provide meaningful input.
- `cancel` - The user asked to cancel the prompt that is playing.
- `telephone.disconnect.hangup` - The user hung up.
- `telephone.disconnect.transfer` - The user's call was transferred.

The following additional events are defined as BeVocal extensions:

- `goback` - User wants to retract the last response and go back to an earlier part of the interaction.

The following standard errors are predefined:

- `error.badfetch` - An error occurred while the interpreter was fetching a document or resource.
- `error.semantic` - A runtime error occurred in the VoiceXML code.
- `error.noauthorization` - The user is not authorized to perform the requested action.
- `error.unsupported.format` - The requested resource format is not supported.
- `error.unsupported.`*`element`* - The requested element is not supported (for example, `error.unsupported.subdialog`).

The following additional errors are defined as BeVocal extensions:

- `error.bevocal.maxdialogerrors_exceeded` - The maximum number of speech errors was exceeded in a particular execution of a particular form.
- `error.bevocal.maxerrors_exceeded` - The maximum number of speech errors was exceeded during the call.

Default Error Handlers

The BeVocal environment provides the following default event handlers the predefined events and errors:

- `exit` - Exit the interpreter.
- `help` - Play a default audio help message and reprompt. The help message says: “No help available right now.”
- `noinput` - Play a default audio message and reprompt. The message says: “I’m sorry, I didn’t hear you.”
- `nomatch` - Play a default audio nomatch message and reprompt. The says: “I’m sorry, I didn’t understand you.”
- `cancel` - Stop audio.
- `error` - Exit the interpreter.
- `telephone.diconnect.hangup` - Exit the interpreter.
- All others - Play a default audio error message and exit the interpreter.

Application-Defined Event Handlers

Although the system provides default handlers for the predefined events, you can override these handlers by providing your own event handlers in any element that can throw an event. The `<catch>`, `<error>`, `<help>`, `<noinput>`, and `<nomatch>` elements are event handlers.

An element in which an event may be thrown also inherits event handlers defined in its ancestor elements. For example, an event thrown within a field element may be caught by a handler in that element, or in its form, or in its document, or in its application. This inheritance of event handlers allows you to provide consistency in event handling by defining handlers at a higher level.

Form items contain *event counters* that let you respond differently if the same event is thrown multiple times. For example, you may want to provide more details each time the user asks for help. The event counters are reset on each form invocation.

When an event occurs, its counter is used to select applicable event handlers. All handlers in the scope in which the event occurred and its containing scopes are considered. A handler for the event is eligible if its `count` attribute is less than or equal to the event’s counter. Those eligible reprompt. The with the highest `count` are selected as applicable. The applicable handlers are ordered by scope, with the innermost event handlers first; within a given scope, the applicable handlers are examined in the order in which the occur in the VoiceXMLdocument. The first applicable handler in this ordering is selected to handle the event.

You can set up event handlers that catch all events with a given prefix (for example, `error.unsupported`). Note, however, that the interpreter selects a handler based on count, scope, and document order only. A more specific handler does not take precedence. For example, if an `error.unsupported.format` event is thrown and the first applicable handler is for all events beginning with the prefix `error.unsupported`, that handler will be invoked even if the next applicable handler is for the specific event `error.unsupported.format`.

Within an event handler, the `_event` variable contains the name of the event currently being handled; the `_message` variable contains the message string that provides additional information about the event. If no message was supplied when the event was thrown, the `_message` variable is `"undefined"`.

Tips:

- Always set up default `<help>`, `<nomatch>`, and `<noinput>` messages of your own, at top level scope. For example:

  ```
  <vxml>
    <help>
      I’m sorry. There’s no help available here.
    </help>
    <noinput>
      I’m sorry. I didn’t hear anything.
      <reprompt/>
    </noinput>
    <nomatch>
      I didn’t get that.
      </reprompt>
    </nomatch>
    ...
  ```
- If you want to execute both an event handler in an inner scope and a handler for the same event in an outer scope, the inner handler can use a `<rethrow>` element to rethrow the event.

Events in Subdialogs

When a subdialog throws an event, the result depends on whether the subdialog is modal. Subdialogs are modal by default; a subdialog can be made non-modal by setting the `modal` attribute to `"false"`.

- If an event is thrown within a modal subdialog and no handler for the event is found in the subdialog’s execution context, a fatal error occurs, causing the interpreter to exit.
- If an event is thrown within a non-modal subdialog and no handler for the event is found in the subdialog’s execution context, the interpreter causes the subdialog’s context to return and rethrows the event in the calling context, restarting the search for the event handler in that context.

Throwing Events

An application can throw events as follows:

- A `<throw>` element throws an event; it can occur within executable content, that is, in a block or `<filled>` element, or an event handler.
- A `<link>` element can specify an event to be thrown when the link's grammar is matched.
- A `<choice>` element in a menu can specify an event to be thrown when the choice's grammar is matched.
- A `<return>` element in a subdialog can specify an event to be thrown after control returns to the calling dialog.

Application-Defined Events

An application can define additional events implicitly. If an element that throws an event specifies an event other than one of the predefined events, it implicitly defines the specified event. For example, the following tag implicitly defines an event named `myEvent` and throws that event.

```
<throw event="myEvent"/>
```

An application can use a `<catch>` element to catch and handle an application-defined-event. For example:

```
<catch event="myEvent">
  ...
</catch>
```

4 Fetching Resources

Currently, two protocols are supported for fetching resources or documents: “http” and “https” (secure HTTP). When the VoiceXML interpreter needs to fetch resources or documents specified by a URL, its behavior is governed by the application’s *fetch policies*. The fetch policies cover caching, optimization and timing out of fetch operations, and the use of background audio during fetch operations. All policies have default settings.

An application can change any default setting with a `<property>` element that sets a property corresponding to the policy to be changed.

- A property set in the `<vxml>` element of a single-document application or the application root document of a multidocument application sets the policy for that document and the application, overriding the default setting.
- A property set in the `<vxml>` element of a non-root document of a multidocument application sets the policy for that document, overriding the setting for the application.
- A property set in a `<form>` or `<menu>` element sets the policy for that dialog, overriding the setting for the containing document.
- A property set in a form item sets the policy for that form item, overriding the setting for the containing form.

Any tag that requests a fetch operation includes attributes that can be set to override the current policy settings during that fetch operation only.

Fetch Policies and Properties

The VoiceXML interpreter uses fetch policies to control its behavior when it fetches resources or documents. The fetch policies govern the following aspects of fetching:

- Caching a fetched file and using cached files instead of repeating fetch operations
- Optimizing fetch operations
- Allowing fetch operations to time out
- Using background audio during fetch operations

The following sections describe the policies and their default settings, and they also list the properties that can be used to set each policy. For a detailed description of the various properties, see Chapter 7, “Property Reference”.

Cached Files

A file that is fetched can be *cached* so that it is available for use in the near future without having to be fetched again. VoiceXML document files, grammar files, audio files, object files, and script files can all be cached.

Most files are cached until they expire. However, if the VoiceXML interpreter encounters any of the following HTTP headers, it does not cache the document:

- `Expires: 0`
- `Pragma: no-cache`
- `Cache-control: no-cache`
- `Cache-control: no-store`

Either of the following two HTTP headers can specify how long to keep the document in the cache, either *N* seconds for the first header or until the specified date for the second:

- `Cache-control: max-age` *N*
- `Expires:` *date*

If headers do not specify when a cached file expires, the file expires immediately. It is still stored in the cache. If the same file is needed in the future, properties control whether the cached file is used as is, revalidated with a "get if modified", or refetched unconditionally.

Using Cached Files

If the interpreter needs a resource that is not in the cache, it fetches and caches the resource. If t a copy of the resource that is in the cache, three policies govern whether the interpreter uses the cached copy:

- The maximum age for files
- The caching policy for files (VoiceXML 1.0 only)
- The maximum stale time for files

You can set properties to control these policies.

Maximum Age

An application can specify the *maximum age* of cached files that the application will use. An unexpired cached file whose age does not exceed the maximum age will be used; a cached file that is older that the maximum will be refetched.

The following properties specify the maximum age, in seconds, for files of the specified kinds:

- `audiomaxage` - audio files
- `datamaxage` - XML data files
- `documentmaxage` - VoiceXML document files
- `grammarmaxage` - grammar files
- `objectmaxage` - object files
- `scriptmaxage` - script files

All these properties are extensions to the BeVocal VoiceXML 1.0 platform.

No default is set for these properties. If you set a maximum-age property to a non-zero value, you ensure that:

- The interpreter uses an unexpired resource whose age is less than or equal to the maximum age—without doing a "get if modified" to verify that the cached file is up to date.
- The interpreter fetches a fresh copy of a resource whose age is more than the maximum age—even if the cached file has not yet expired.

For example, suppose you fetch a VoiceXML document file that expires in 60 seconds, and after 40 seconds you need the same file. If `documentmaxage` is set to

30, the application will refetch the document file; if `documentmaxage` is set to 60, it will use the cached file.

You can set a maximum-age property to 0 to ensure that a fresh copy is retched if the resource has been modified since it was last fetched.

If a resource is within the maximum age but the cached file has expired, the relevant maximum-stale-time policy determines whether the interpreter uses the expired cached file.

Caching

In a VoiceXML 1.0 application, when the relevant maximum-age property is not set, the caching policy determines whether the interpreter uses an unexpired cached copy of a file.

The `caching` property specifies the caching policy for all kinds of files:

- If the caching property is `"fast"` (the default), the cached copy is used.
- If the caching property is `"safe"`, the interpreter does a “get if modified” to update the cached file, if necessary.

If the cached file has expired, the relevant maximum-stale-time policy determines whether the interpreter uses the expired cached file.

Note: The `caching` property is ignored when the `vxml` tag’s `version` attribute equals `2.0`, and an unexpired cached copy of a file is used if the relevant `maximum-age` property is not set.

Maximum Stale Time

An application can specify the *maximum stale time* during which expired files will be used without being refetched. The maximum stale time for a file is the time by which its expiration time can be exceeded. Within this allowable stale time, the expired cached file will still be used; if the file is needed after its maximum stale time has been exceeded, the file will be refetched.

The following properties specify the maximum stale time, in seconds, for files of the specified kinds:

- `audiomaxstale` - audio files
- `datamaxstale` - XML data files
- `documentmaxstale` - VoiceXML document files
- `grammarmaxstale` - grammar files
- `objectmaxstale` - object files
- `scriptmaxstale` - script files

The default for `audiomaxstale` is 300 seconds (5 minutes); the default for the other properties is 0. All these properties are extensions to the BeVocal VoiceXML 1.0 platform.

The maximum stale time is relevant in when the expired file is within the maximum age and when no maximum age is set for the file. If the number of seconds since the the cached file expired is less than or equal to the maximum stale time, the cached file is used. If the file has been expired for longer than the maximum stale time, the interpreter does a “get if modified” to update the cached file, if necessary.

This property allows you to specify that an expired file that is “not too stale” to be used. For example, if you do not have direct server-side control of the expiration dates of large static files, you might use this property to avoid performing repeated gets on the files.

Fetch Optimization

The interpreter can attempt to optimize dialog interpretation by prefetching files that might be needed or by streaming fetched audio files.

By default, the interpreter loads a VoiceXML document file only when it is needed. It can, however, prefetch grammar, audio, object, or script files, and it can stream audio files.

Note: The ability to stream audio files in a Beta feature.

An application can control whether the interpreter may attempts to optimization fetch operations with the following properties that set the policy for files of the specified kinds:

- `audiofetchhint` - audio files
- `datafetchhint` - XML data files
- `documentfetchhint` - VoiceXML document files
- `grammarfetchhint` - grammar files
- `objectfetchhint` - object files
- `scriptfetchhint` - script files

Timeouts

By default, the interpreter waits up to one minute for a resource or document to be fetched. The application can control this behavior with the `fetchtimeout` property, which sets the timeout period for fetch operations. The interpreter waits for the specified timeout period. Then, if the resource still has not been returned, the interpreter throws an `error.badfetch` event.

Background Audio

By default, the user does not hear any audio output while the interpreter is fetching a file of any kind.

The application can change this behavior with the `fetchaudio` property, which specifies a “background audio” file to be played while the interpreter fetches a VoiceXML document or object file.

Note: Background audio is never played while the interpreter fetches a grammar, audio, or script file.

When a background audio file is specified for a fetch operation, the fetching of that background audio file is governed by the `caching`, `audiofetchhint`, `audiomaxage`, `audiomaxstale`, and `fetchtimeoput` properties that are in effect at the time of the fetch. The following properties govern the playing of the background audio clip:

- `fetchaudiodelay` - the amount of time to wait after a VoiceXML download is started before background audio clip is played.
- `fetchaudiominimum` - the minimum time interval to play the background audio clip, once started, even if the fetch result arrives in the mean time.

Both these properties are extensions to the BeVocal VoiceXML 1.0 platform.

Fetch Attributes

Currently, two protocols are supported for fetching resources or documents: “http” and “https” (secure HTTP). When the VoiceXML interpreter needs to fetch resources or documents specified by a URL, its behavior is governed by the values of six attributes of the associated tags.

- `caching`
- `fetchaudio`
- `fetchhint`
- `fetchtimeout`
- `maxage`
- `maxstale`

caching

The `caching` attribute specifies the caching policy for the resource being fetched. It can be set to either of the following values:

- `safe` - Ensures the most recent version of the resource will be fetched.
- `fast` - Uses a cached copy of the file if it has not expired.

Note: This attribute is used *only* in a VoiceXML 1.0 application when the `maxage` attribute does not have a value and the cache contains an unexpired copy of the resource.

Optional. If not specified, the current value of the `caching` property is used.

The following tags can have a `caching` attribute:

- `<audio>`
- `<choice>`
- `<data>`
- `<dtmf>`
- `<goto>`
- `<grammar>`
- `<link>`
- `<object>`
- `<script>`
- `<subdialog>`
- `<submit>`

Note: The `caching` property is ignored when the `vxml` tag’s `version` attribute equals `2.0`, and an unexpired cached copy of a file is used if the `maxage` attribute does not have a value.

fetchaudio

The `fetchaudio` attribute specifies the URL of an audio clip to play while the VoiceXML document is being fetched.

Optional. If not specified, the current value of the `fetchaudio` property is used.

When a background audio file is specified for a fetch operation,

- The fetching of the background audio file is governed by the `caching`, `audiofetchhint`, `audiomaxage`, `audiomaxstale`, and `fetchtimeoput` properties that are in effect at the time of the fetch.

- The playing of the audio clip is governed by the `fetchaudiodelay` and `fetchaudiominimum` properties that are in effect at the time of the fetch.

The following tags can have a `fetchaudio` attribute:

- `<choice>`
- `<goto>`
- `<link>`
- `<object>`
- `<send>`
- `<subdialog>`
- `<submit>`

fetchhint

The `fetchhint` attribute specifies whether the interpreter can attempt to optimize dialog interpretation by prefetching the resource. This property can be set to one of the following values:

- `prefetch` - Fetch the resource when the page is loaded.
- `safe` - Fetch the resource only when it is needed.
- `stream` - For audio files only. Stream the fetched audio file, that is, process data in the file as it arrives without waiting for the full retrieval.

  Note: The ability to stream audio files in a beta feature.

Optional. Default is the current value of the relevant property:

- `audiofetchhint` for audio files
- `datafetchhint` for XML data files
- `documentfetchhint` for VoiceXML document files
- `grammarfetchhint` for grammar files
- `objectfetchhint` for object files
- `scriptfetchhint` for script files

The default for the `documentfetchhint` property is `"safe"`; the default for the other properties is `"prefetch"`.

The following tags can have a `fetchhint` attribute:

- `<audio>`
- `<choice>`
- `<data>`
- `<dtmf>`
- `<goto>`
- `<grammar>`
- `<link>`
- `<object>`
- `<script>`
- `<subdialog>`

fetchtimeout

The `fetchtimeout` attribute specifies the interval to wait for the resource to be returned before throwing a `error.badfetch` event. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds; `"ms"` for time in milliseconds (the default).

Optional. If not specified, the current value of the `fetchtimeout` property is used.

The following tags can have a `fetchtimeout` attribute:

- `<audio>`
- `<choice>`
- `<data>`
- `<dtmf>`
- `<goto>`
- `<grammar>`
- `<link>`
- `<object>`
- `<script>`
- `<send>`
- `<subdialog>`
- `<submit>`

maxage

The `maxage` attribute specifies the maximum acceptable age, in seconds, of the cached resource. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again.

Optional. Default is the current value, if any, of the relevant property:

- `audiomaxage` for audio files
- `datamaxage` for XML data files
- `documentmaxage` for VoiceXML document files
- `grammarmaxage` for grammar files
- `objectmaxage` for object files
- `scriptmaxage` for script files

When no value is set for this attribute, the `caching` attribute controls whether an unexpired cached file is used.

The following tags can have a `maxage` attribute:

- `<audio>`
- `<choice>`
- `<data>`
- `<goto>`
- `<grammar>`
- `<link>`
- `<object>`
- `<script>`
- `<subdialog>`
- `<submit>`

maxstale

The `maxstale` attribute specifies the maximum acceptable time, in seconds, during which an expired cached resource can still be used. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

A cached file that has been expired for more that the maximum stale time will be refetched; one that has been stale for less than or equal to the maximum stale time will be used.

Optional. Default is the current value of the relevant property:

- `audiomaxstale` for audio files
- `datamaxstale` for XML data files
- `documentmaxstale` for VoiceXML document files
- `grammarmaxstale` for grammar files
- `objectmaxstale` for object files
- `scriptmaxstale` for script files

The following tags can have a `maxstale` attribute:

- `<audio>`
- `<choice>`
- `<data>`
- `<goto>`
- `<grammar>`
- `<link>`
- `<object>`
- `<script>`
- `<subdialog>`
- `<submit>`

5 Go-Back Facility

The BeVocal Café *go-back facility* allows the user to retract the last response or to transition back to the last location in an application.

Note: The go-back facility 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.

Retracting User Responses

If the go-back facility is enabled, the user can retract the last response to a VoiceXML application by saying “go back.” After the interpreter “removes” the user’s response, it prompts for the information again.

For example, the following form asks for the user’s home and work phone numbers:

```
<form>
  <field name="home" type="phone">
    <prompt>
      What is your home phone number?
    </prompt>
  </field>
  <field name="work" type="phone">
    <prompt>
      What is your work phone number?
    </prompt>
  </field>
</form>
```

Suppose a user inadvertently gives the work number when asked for the home number. The go-back facility would allow the user to correct this mistake.

Application:	What is your home phone number?
User:	408-555-3200.
Application:	What is your work phone number?
User:	Go back.
Application:	What is your home phone number?
User:	408-555-3042.
Application:	What is your work phone number?
User:	408-555-3200.

The go-back facility also allows users to change their minds after requesting one of several alternatives. For example, it would permit the following interaction:

Application:	Would you like News, Weather, or Traffic?
User:	Weather.
Application:	What city?
User:	Go back.
Application:	Would you like News, Weather, or Traffic?
User:	Traffic.

Go-Back Stack

When user says “go back,” the interpreter undoes whatever actions resulted from the last response, then it prompts the user for a new response. The user can retract a sequence of responses by saying “go back” repeatedly.

Each request for user input is called a *go-back destination*. When the user provides the requested input, the interpreter saves information about the go-back destination as an entry on its *go-back stack*.

If the user says “go back,” the interpreter uses the saved information for the most recent go-back destination on the stack to undo the actions that resulted from the user’s response. It then goes back to that go-back destination, popping the corresponding entry off the stack.

Stack Entries

Each entry on the go-back stack saves information about one step the interpreter performed during the execution of the application.

Go-Back Entries

The entries corresponding to go-back destinations are called *go-back entries*; they correspond to the user-visible steps in the interaction. As the user retraces these steps, the interpreter goes back to the appropriate elements within the VoiceXML application, transparently moving between dialogs and documents as necessary. For example:

- After the user fills the last field in a form, the form may transition to a different form in a different document. If the user says “go back” to the first question on the new form, the interpreter returns to the first form in the original document. It clears the last field in that form, but restores the values of all other form item variables in the form.
- A user’s response may match a link grammar that transitions to a different form or that throws an event that causes a transition. Or a user’s response may match a document-scoped grammar in a different form, causing a transition to that form. If the user says “go back” to the first question in the new location, the interpreter returns to the form and field that was being visited at the time of the user’s last response.

Internal Entries

In addition to the go-back entries, the go-back stack saves *internal entries*, which correspond to non-user-visible steps, such as transitions between forms. When the interpreter goes back to the most recent go-back destination, it also “undoes” each non-user-visible step that occurred after the last go-back destination and pops the corresponding internal entry off the stack.

A `<block>` form item does not request user input and so is not a possible go-back destination. However, any block items that are executed between one input request and the next are saved as internal stack entries that can be undone when the interpreter goes back to the preceding input request.

Setting Stack Size

The `bevocal.mingoback` property specifies the minimum size of the go-back stack. The interpreter keeps at least this many entries on the stack, except at the beginning of the call when fewer steps have been executed, and after the user has said “go back” so many consecutive times that the stack has been depleted.

By default, this property is set to 0, which means that the go-back stack is always empty and the go-back facility is effectively disabled.

If you want your application to provide the go-back facility, you must use the `<property>` tag to set the `bevocal.mingoback` to 1 or more.

For example, the following application sets the minimum stack size to 20 entries.

```
<?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">
  <!-- Save at least 20 entries on the go-back stack -->
  <property name="bevocal.mingoback" value="20"/>
  <form>
    <field name="home" type="phone">
      <prompt>
        What is your home phone number?
      </prompt>
    </field>
    <field name="work" type="phone">
      <prompt>
        What is your work phone number?
      </prompt>
    </field>
  </form>
</vxml>
```

Go-Back Destinations

A VoiceXML application can request user input in a menu, in the initial item of a mixed-initiative form, and in a field item. These elements, therefore, can be go-back destinations.

Menus

A `<menu>` element asks the user to select a choice. The `<menu>` element is the go-back destination for the user's response. If the user says "go back" after selecting a menu choice, the menu is executed again.

Mixed-Initiative Forms

The `<initial>` element of a mixed-initiative form asks the user for initial input to the form. This element is the go-back destination for the user's response. If the user says "go back" after providing initial input, the initial element is executed again.

The user's answer to the initial prompt may provide values for several of the form's field item variables. When the interpreter "undoes" an initial element, it clear's not only the initial form item variable, but also any field item variables that were set by the user's response.

Field Items

For the purposes of the go-back facility, field items can be classified as follows:

- The `<field>` and `<record>` items accept a single user input. These field items appear to the user as a single request for information.
- The `<transfer>` item may involve a long interaction between the user and a third party. It provides the application with a single piece of information, namely the result of the transfer. During a transfer, however, the user may provide various pieces of information to the third party and may later want to retract some or all of that information.
- The `<object>` and `<subdialog>` items may accept multiple user inputs and so they may appear to the user as multiple requests for information.

Single-Input Field Items

A `<field>` item asks the user for the value of its field item variable. A `<record>` item asks the user for input to be recorded. These field items are go-back destinations. If the interpreter goes back to one of these items, it clears the corresponding field item variable and executes the item again. Going back to a `<field>` item allows the user to give a different answer; going back to a `<record>` item allows the user to provide different input to be recorded.

Transfer Items

A `<transfer>` item transfers the user to another destination, allowing the user to carry on a conversation with a third party.

- At the end of a bridging transfer, the interpreter resumes execution of the form containing the transfer item. The user might then say "go back" to the next request for input.
- In a blind transfer, the current session terminates when the transfer is made; the user has no opportunity to invoke the go-back facility at the end of the call.

(Currently, the BeVocal interpreter supports bridging transfers only.)

A transfer item is a go-back destination. If the interpreter goes back to a transfer item, the transfer call is repeated. Any change in the information exchanged during the original transfer and during the repeated transfer is determined by the user's conversation with the third party and does not affect the VoiceXML application. For example, the original transfer might place a call in which the user orders a pizza. After that call, the user might say "go back," and add a salad to the original order.

Object Items

An `<object>` item invokes a reusable Speech Object component. That component may make one or more requests for information from the user. The information supplied by the user (and other information) may be returned to the application in properties of the field-item variable.

An object item is a go-back destination; however, the individual requests for input made during the execution of the object are *not* go-back destinations.

Go-back behavior during the execution of the object is determined entirely by the object implementation. The object might ignore a "go back" request completely. Alternatively, the object might implement its own go-back facility, allowing the user to retract answers to questions asked by the object.

Once control returns from the object to the VoiceXML interpreter, all the object's internal state is lost. If the user says "go back" to the next question after executing the object, the object is executed again.

If an object requests a single user input, the go-back behavior is the same as for any other single-input field item. If an object requests more than one user input, however, the go-back behavior may not be what the user expects. For example, suppose a form contains the following field items:

```
<field name="A">...</field>
<object name="B" ...>
  ...
  <!-- Object asks questions C, D, and E -->
</object>
<field name="F">...</field>
```

A user who says "go back" when prompted for field F, might expect to provide a different answer to question E. However, the interpreter goes back to object B. It executes the object from the beginning, asking questions C, D, and E again.

Subdialogs

A `<subdialog>` item invokes another dialog as a subdialog of the current one. Each request for input made by the subdialog is a go-back destination. The `<subdialog>` element itself is also a go-back destination.

If the user says "go back" to a request for input inside the subdialog, the go-back behavior is the same as in any other form. Within the subdialog's execution context, the go-back stack is initially identical to the go-back stack in the calling dialog's execution context. As each new input is requested, another go-back destination is pushed onto the stack:

- If the user says "go back" to the first input request in the subdialog, the interpreter returns to the last go-back destination in the calling dialog.
- If the user says "go back" to a subsequent input request in the subdialog, the interpreter returns to the preceding go-back destination in the subdialog

Once the subdialog returns to the calling dialog, however, the subdialog's execution context terminates. The go-back stack in the calling dialog's execution context does not contain any go-back destinations for the input requests made by the subdialog. A new go-back destination is added for the subdialog itself.

If the subdialog requests a single user input, the go-back behavior is the same as for any other single-input field item. If an subdialog requests more than one user input, however, the go-back behavior may not be what the user expects. For example, suppose a document contains the following forms:

```
<form id="main">
  <field name="A">...</field>
  <subdialog name="B" src="#sub">
    ...
  </subdialog>
  <field name="F">...</field>
</form>
<form id="sub">
  <field name="C">...</field>
  <field name="D">...</field>
  <field name="E">...</field>
  <filled>
    <return namelist="A B C"/>
  </filled>
</form>
```

A user who says "go back" when prompted for field F, might expect to provide a different answer for field E in the subdialog. However, the interpreter goes back to subdialog B. It executes the subdialog from the beginning, prompting again for fields C, D, and E.

Note: In a future release, you may be able to specify whether "go back" will go back into the subdialog (to ask for field E in the preceding example) or to the beginning of the subdialog (as currently happens).

Controlling Go-Back Behavior

You can control the application's use of the go-back facility in the following ways:

- You can prevent the user from retracting certain inputs.
- You can customize the application's response to a go-back request from the user.
- You can deactivate the go-back facility in the entire application, in a particular document, in a particular dialog, or in a particular go-back destination.

Suppressing Retraction

You can prevent the user from retracting certain inputs by setting the `bevocal.goback` property. This property controls whether requests for user input are legal go-back destinations. By default, the property is set to `"true"` and each request for input is a legal go-back destination. When the user provides the requested input, the interpreter pushes a go-back entry for the request onto its go-back stack.

If the `bevocal.goback` property is `"false"`, however, a request for input is a not a legal go-back destination. When the user provides the requested input, the interpreter pushes an internal entry for the request onto its go-back stack. The internal stack entry enables the interpreter to undo the information request if the user returns to an earlier go-back destination; however, it prevents the user from going back to the request itself.

A user's response is called "retractable" if a corresponding go-back entry is added to the stack; if, instead, an internal entry is added to the stack, the response cannot be retracted.

If you set the `bevocal.goback` property to `"false"` in a field, the user's input for the field is not retractable. The user cannot go back to that field, but may skip back to retract the preceding retractable input. If you set this property to `"false"` in a form, you prevent the user from retracting any input to that form.

When several fields are treated as a single conceptual unit, you may want to suppress retraction of all but the first field. For example, the go-back facility treats the `city` and `state` fields as a unit in the following form:

```
<form>
  <field name="city">
    <prompt>Choose a city</prompt>
    <grammar>...</grammar>
  </field>
  <field name="state">
    <property name="bevocal.goback" value="false"/>
    <prompt>What state?</prompt>
    <grammar>...</grammar>
  </field>
  <field name="first" type="boolean">
    <prompt>
      Do you want to fly first class?
    </prompt>
  </field>
</form>
```

The user cannot retract an answer to the question about state, but can skip past it to retract the city, al illustrated in the following interaction.

Application:	Choose a city.
User:	Albany
Application:	What state?
User:	Georgia
Application:	Do you want to fly first class?
User:	Go back.
Application:	Choose a city.

Customizing Go-Back

When the speech-recognition engine matches the `"goback"` grammar, a `"goback"` event is thrown. The default handler undoes entries on the go-back stack until it reaches the most recent go-back entry, corresponding to the user's last retractable response. If the go-back stack is empty, the default handler plays an audio message that says "Sorry, you can't go back."

If you want the application to take different actions, you can add your own event handler for go-back events. For example, an application might keep information about each user's default location. If the user requests a traffic report from the main menu, the traffic form might start to fetch the report for the user's default location without requesting the user's city. The application could use the go-back facility to allow the user to provide a different location.

```
<form id="traffic">
  <catch event="goback">
    <clear/>
  </catch>
  <field name="city" expr="document.defaultCity">
    <prompt>What city?</prompt>
    <grammar>...</grammar>
  </field>
  <block>
    <prompt>
      Retrieving traffic data for
      <value name="city">
      Say Go Back to choose another city.
    </prompt>
    <!-- Retrieve and play traffic report -->
  </block>
</form>
```

An interaction with the application might proceed as follows.

Application:	Would you like news, weather, or traffic?
User:	Traffic
Application:	Retrieving traffic data for San Francisco. Say Go Back to choose another city.
User:	Go back.
Application:	What city?
User:	San Jose.
Application:	Retrieving traffic data for San Francisco. Say Go Back to choose another city.

In this case, saying "go back" takes the user to a question that has never been asked before.

If the application's go-back handler needs to take some actions and then proceed as normal to undo the user's response, it can perform the appropriate actions and then rethrow the event to the default handler:

```
<catch event="goback">
  ...
  <rethrow/>
</catch>
```

Activating and Deactivating Go-Back

The universal `"goback"` grammar recognizes the spoken “go back” request. Like all universal grammars, it is activated by default in a VoiceXML 1.0 application and deactivated by default in an application where the `vxml` tag’s `version` attribute equals `2.0`. See “Universal Grammars” on page 20.

You can set the `universals` property to activate or deactivate the `"goback"` grammar, either in the entire application, or in particular documents, forms, or fields. For example, the go-back facility is activated by default in the following document, but deactivated during the execution of a the first form:

```
<vxml version="1.0">
  <form>
    <!-- Activate only help and exit universals -->
    <property name="universals" value="help exit"/>
    ...
  </form>
  ...
</vxml>
```

When the go-back facility is deactivated, the speech-recognition engine does not recognize to the input “go back.” If the user says “go back” to the prompt for a field, a no-match event is thrown. Note that this behavior is different from the default behavior when the `bevocal.mingoback` property is set to 0. In that case, the “go back” request is recognized. However, the go-back stack is empty, so the user hears the message, “Sorry, you can’t go back.”

Using the Go-Back Facility

This section contains guidelines for using the go-back facility

Setting the Stack Size

You need to set the size of the go-back stack large enough to enable a user to retrace as many steps as you think are likely. The size of the stack limits the number of consecutive times the user can say “go back.” Remember, however, that the stack must be large enough to accommodate internal entries as well as go-back entries. When you set the stack size, you should allow for a few internal entries for each go-back entry.

Using Blocks

You can safely put blocks between go-back destinations in a form. For example, in the following form, if the user goes back to the `home` field, the interpreter “undoes” the subsequent block, clearing its item variable and allowing the block to be visited again after the user provides an new answer for the `home` field:

```
<form>
  <field name="home" type="phone">
    <prompt>
      What is your home phone number?
    </prompt>
  </field>
  <block>
    Your home number is <value expr="home"/>
  </block>
  <field name="work" type="phone">
    <prompt>
      What is your work phone number?
    </prompt>
  </field>
</form>
```

The interaction with the user might proceed as follows.

Application:	What is your home phone number?
User:	408-555-3200.
Application:	Your home number is 408-555-3200. What is your work phone number?
User:	Go back.
Application:	What is your home phone number?
User:	408-555-3042.
Application:	Your home number is 408-555-3042. What is your work phone number?
User:	408-555-3200.

A `<block>` in a form is saved as an internal stack entry only if it occurs after the first go-back destination in the form. If the form’s first item is a block containing a welcoming prompt, no internal stack entry is saved for the block, so it will not be revisited if the go-back facility returns to first input request in the form.

Tip:

- In a mixed-initiative form, put any welcoming prompt in the `<initial>` element, not in a separate `<block>` element.

The internal stack entry for a block is undone and redone only if the interpreter returns to a go-back destination *before* the block. As a consequence, a block that is used to prompt for information in the subsequent field is not redone if the interpreter goes back to the field. In the following form, if the user says “go back” when asked for a work phone number, the request for the home phone number would not be replayed.

```
<form>
  <block>
      What is your home phone number?
  </block>
  <field name="home" type="phone">
  </field>
  <block>
    Your home number is <value expr="home"/>
  </block>
  <field name="work" type="phone">
    <prompt>
      What is your work phone number?
    </prompt>
  </field>
</form>
```

Tip:

- Be sure to put the prompt for a field value inside the `<field>` element and not in a separate `<block>` element.

Using Subdialogs

To avoid any confusion that can occur if the user says “go back” after returning from a subdialog, try to limit your use of subdialogs to requests for confirmation or disambiguation. In addition, you should prevent the subdialog itself from being a legal go-back destination by setting the `bevocal.goback` property to `"false"` inside the `<subdialog>` element. If the user says “go back” after the subdialog returns, the interpreter will go back to the question preceding the subdialog—presumably the question whose answer required confirmation or clarification.

Using Variables

When the interpreter returns to a particular go-back destination in a form, it clears form-item variables for every block, initial item, and field item that needs to be undone. However, it does not change the values of any other variables declared in dialog, document, or application scope. If the interpreter undoes a transition, going back to a different form, it does not restore the variables declared in the form to the values they had when that transition left the form.

If you use the go-back facility, you should avoid saving state information in variables that cannot be reset by a go-back operation.

In limited circumstances, you may be able to reset variables in an error handler for go-back events. In general, however, the event handler will not have enough context to know what variables need to be reset because the event is thrown at the location where the user says “go back”, not at the go-back destination.

6 Tag Reference

This chapter provides detailed information about each VoiceXML tag. 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.

In the cases where the BeVocal interpreter deviates from the VoiceXML 1.0 Specification, the difference is clearly marked below in the following ways:

- *Not Implemented* - Functionality not currently available.
- *Extension* - Added functionality.
- *Experimental Extension* - Added functionality that may be included in a later specification for VoiceXML. If the extension is standardized, the BeVocal implementation will change as necessary to match the VoiceXML standard.
- *Deprecated* - Non-standard or superseded feature that was supported by an earlier version but has been replaced by a new feature.

<assign>

Assign a variable a value.

Syntax

```
<assign
    name="string"
    expr="js_expression"/>
```

Description

Attribute	Description
`name`	Name of variable.
`expr`	JavaScript expression that evaluates to the value assigned to this variable.

Tip:

- In JavaScript, `"+"` means both string concatenation and add. If you want to add two numbers represented by string variables a and b in an `<assign>`, use:
  `<assign name="x" expr="Number(a) + Number(b)"/>`

  This is because `expr="a + b"` will concatenate the string values. Multiply, divide, and subtract are not ambiguous in this way.

Usage

Parents	Children
`<block>` `<catch>` `<error>` `<help>` `<noinput>` `<nomatch>` `<if>` `<filled>`	None

See Also

- VoiceXML 1.0 Specification:
  <assign>
- JavaScript Quick Reference
- Related tag:
  “<var>” on page 193

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="foo">
    <var name="a"/>
    <var name="b"/>
    <var name="result"/>
    <block>
      <assign name="a" expr="’Pine’"/>
      <assign name="b" expr="’Apple’"/>
      <assign name="result" expr="a + b"/>
    </block>
    <block>
      <prompt>
        This is the test for the assign tag.
        If you put <value expr="a"/> and <value expr="b"/> together, it
        would make <value expr="result"/>
      </prompt>
    </block>
  </form>
</vxml>
```

<audio>

Play an audio clip within a prompt.

Syntax

```
<audio
   src="URL"
   expr="js_expression"
   caching="safe"|"fast"
   fetchhint="prefetch"|"safe"|"stream"
   fetchtimeout="time_interval"
   maxage="time_interval"
   maxstale="time_interval" >
  Optional Content
</audio>
```

Description

Attribute	Description
`src`	The URL of the audio file. *Optional* (as alternative to `expr`). If not specified or invalid (that is, the interpreter was unable to perform the fetch from the specified URL), any content of the `<audio>` element will be played instead. The content can include text or valid child tags.
`expr`	*Extension.* JavaScript expression that evaluates to either a string or an array or strings. If it evaluates to a string, the string is interpreted as a URL and the audio file at the location is fetched and played. It it is an array, each element is treated as an audio file URL, each of which is fetched and played, in turn. *Optional* (as alternative to `src`).
`caching`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchhint`	See Chapter 4, "Fetching Resources". *Optional.* **Notes:** • The interpreter can prefetch an audio file specified by the `src` attribute, but not by the `expr` attribute. • The ability to stream audio files in a beta feature.
`fetchtimeout`	See Chapter 4, "Fetching Resources". *Optional.*
`maxage`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*
`maxstale`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*

The following formats are supported for audio files:

Audio Format	MIME Type
AU files (with the "au" header format)	`audio/basic`
Raw (headerless) 8 KHz 8-bit mono mu-law [PCM] single channel	`audio/basic`
Raw (headerless) 8 KHz 8-bit mono A-law [PCM] single channel	`audio/x-alaw-basic`

Audio Format	**MIME Type**
WAV (RIFF header) 8 KHz 8-bit mono mu-law [PCM] single channel	`audio/wav`
WAV (RIFF header) 8 KHz 8-bit mono A-law [PCM] single channel	`audio/wav`

Tips:

- Using `<audio>` rather than `<prompt>` makes it easy to set up TTS prompts now and add professional prompts later. With `<prompt>`, you can only use TTS.
- Audio files must be 8 KHz mono WAV files, AU files, or MP3 files. If the specified file is of a different type, any alternative audio content of the `<audio>` element (text, prompts, and so on) is played instead.
- Note that you can use `<audio>` within a `<prompt>`. If you do, it will inherit the attributes of the `<prompt>` element, such as `bargein`.
- If you name your audio files consistently, you can use the `expr` attribute to simplify the way you construct audio file names in your VoiceXML. For example:

```
<prompt>
  <audio expr="'resources/prompts/hello'
    + sign +'.wav'"/>
</prompt>
<prompt>
  <audio expr="'resources/prompts/' + sign
    +'.wav'"/>
</prompt>
```

- If you use the `expr` attribute in place of the `src` attribute, the interpreter cannot prefetch the audio file, which may cause a minor performance degradation. Once a given file is in the interpreter's cache, however, this difference is typically not noticeable.
- Using JavaScript functions makes it easy to update the location of your audio files. For example:

```
function female1(a) {
return("audio/female1/en_us/" + a); }

function common(b) {
return(female1("common/" + b + ".wav")); }

function number(b) {
return(female1("number/" + b + ".wav")); }
```

  For an expanded example, see factorial sample code.

  (http://cafe.bevocal.com/docs/samples/factorial/factorial.vxml)

- When calling JavaScript functions, use apostrophes in place of double quotes:

```
<audio expr="common('bevocal_chimes')"/>
```

Usage

Parents	Children
`<menu>`	`<audio>`
`<choice>`	`<enumerate>`
`<prompt>`	`<value>`
`<enumerate>`	`<break>`
`<field>`	`<div>`
`<initial>`	`<emp>`
`<block>`	`<pros>`
`<catch>`	`<sayas>`
`<error>`	`<say-as>`
`<help>`	
`<noinput>`	
`<nomatch>`	
`<audio>`	
`<div>`	
`<emp>`	
`<pros>`	
`<record>`	
`<transfer>`	
`<if>`	
`<filled>`	
`<subdialog>`	
`<object>`	

See Also

- VoiceXML 1.0 Specification:
  <audio>

Examples

Example 1 - using src:

```
<?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="foo">
    <block>
      <audio>
        Welcome to BeVocal Cafe, the number One place to build and
        deploy your voice applications.
      </audio>
      <audio caching="safe"
src="http://cafe.bevocal.com/libraries/audio/female1/en_us/common/bevocal_cafe.wav"
/>
      BeVocal Cafe.
    </block>
  </form>
</vxml>
```

Example 2 - using expr:

```
<?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">

  <!-- Define functions; one returns a string;
    many returns an array of strings -->

  <script>
    <![CDATA[
      base = "http://cafe.bevocal.com/libraries/audio/female1/en_us/number/";
    ]]>

    function one() {
      return base + "6000-e.wav";
    }

    function many() {
      var result = new Array(4);
      result[0] = base + "6000-b.wav";
      result[1] = base + "300.wav";
      result[2] = base + "37_and.wav";
      result[3] = base + "1-32.wav";
      return result;
    }
  </script>

  <form>
    <block>
      <prompt>
        Playing result of one
        <audio expr="one()">one</audio>
        <break/>
        Playing result of many
        <audio expr="many()">many</audio>
      </prompt>
    </block>
  </form>
</vxml>
```

<block>

Contain (non-interactive) executable code.

Syntax

```
<block
   name="string"
   expr="js_expression"
   cond="js_expression">
  Executable Content
</block>
```

Description

Control item container of executable code. As with all form items, a block's form item variable must have a value of `"undefined"` before the block can execute. Just before the block is entered, the interpreter sets its form item variables to `"true",` so a block is typically executed only once per form invocation.

Attribute	Description
`name`	Name of form item variable, which may not be a JavaScript reserved keyword. *Optional* (default is an unusable internal name). The form item variable has dialog (form) scope; its name must be unique among all VoiceXML and JavaScript variables within the form's scope. Generally, you use this attribute only if you want to control block execution explicitly.
`expr`	JavaScript expression that assigns the initial value of the form item variable. *Optional* (default is `"undefined"`). If you set the form item variable to a value other than `"undefined",` then you'll need to clear it before the block can execute. Note that you need to give the block a name if you want to clear it separately from other form item variables.
`cond`	JavaScript boolean expression that must also evaluate to `"true"` for the block to execute. *Optional* (default is `"true"`). If not specified, the value of the form item variable alone determines whether or not the block can execute.

Usage

Parents	Children

`<form>`	`<audio>`
	`<value>`
	`<assign>`
	`<clear>`
	`<disconnect>`
	`<exit>`
	`<goto>`
	`<if>`
	`<prompt>`
	`<reprompt>`
	`<return>`
	`<script>`
	`<submit>`
	`<throw>`
	`<var>`
	`<log>`
	`<send>`

See Also

- VoiceXML 1.0 Specification:
  <block>

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="foo">
    <block name="hello">
      <prompt>
        Welcome to BeVocal Cafe. It is the best known place
        for Voice X M L Development.
        <audio src="bevocal_chimes.wav"> </audio>
      </prompt>
    </block>
  </form>
</vxml>
```

<break>

Java Speech Markup Language (JSML) element to insert a pause in output.

Syntax

```
<break
    msecs="time_interval"
    size="none"|"small"|"medium"|"large"/>
```

Description

Attribute	Description
`msecs`	Amount of time to pause, in milliseconds. *Optional* (default is 1 second). Express time interval as an unsigned number followed by `"s"` for time in seconds; `"ms"` for time in milliseconds (the default).
`size`	How long to pause, specified qualitatively. *Optional* (as alternative to `msecs`). • `none` - No pause. • `small` - Short pause (500 milliseconds). • `medium` - Longer pause (1 second). • `large` - Long pause (2 seconds).

Usage

Parents	Children

`<choice>`
`<prompt>`
`<enumerate>`
`<audio>`
`<div>`
`<emp>`
`<pros>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <break>
- Related tags:
  “<div>” on page 80
  “<emp>” on page 87
  “<pros>” on page 156
  “<sayas>” on page 168

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="demo-break">
    <block>
      <prompt>
        Voice X M L allows the programming of silence
        <break size="medium"/>
        with the  break tag.
      </prompt>
    </block>
  </form>
</vxml>
```

<catch>

Catch an event.

Syntax

```
<catch
    event="event1 ..."
    count="integer"
    cond="js_expression">
  Executable Content
</catch>
```

Description

Container for event handling code. Like `<block>`, you can put non-interactive executable code (procedural logic) in a `<catch>` element to handle an event.

Attribute	Description
`event`	Name of the event(s) to catch. *Extension*. You can specify an empty string to catch all events.
`count`	Minimum number of times the event must have occurred during a form or menu invocation. Lets you handle different occurrences of the same event differently. *Optional* (default is `"1"`).
`cond`	JavaScript boolean expression that must also evaluate to `"true"` for an event to be caught. *Optional* (default is `"true"`).

Although you can define your own events, there is a set of predefined events. The BeVocal interpreter provides a standard set of default event handlers for the predefined events.

If multiple handlers for a given event are defined in, or inherited by, the element in which the event occurs, one handler is chosen based on count, scope, and document order. See Chapter 3, “Event Handling”.

Tips:

- Because you can throw events from within a `<catch>`, be sure to avoid infinite loops. For example, the following handler would result in an infinite loop:
  ```
  <catch event="foobar">
    <throw event="foobar"/>
  </catch>
  ```
- You can use a `<submit>` within a `<catch>` for a `telephone.disconnect.hangup` event to notify the server that the call has ended. Because the call is no longer connected, any VoiceXML document returned from the server will be ignored and the interpreter will exit. Similarly, if you use a `<goto>` within a `<catch>` for this event, it will be ignored and the interpreter will exit.
- Within an event handler, the `_event` variable contains the name of the event currently being handled; the `_message` variable contains the message string

that provides additional information about the event. If no message was supplied when the event was thrown, the `_message` variable is `"undefined"`.

Usage

Parents	Children

`<vxml>`
`<form>`
`<menu>`
`<field>`
`<initial>`
`<record>`
`<transfer>`
`<subdialog>`
`<object>`
	`<audio>`
	`<enumerate>`
	`<value>`
	`<assign>`
	`<clear>`
	`<disconnect>`
	`<exit>`
	`<goto>`
	`<if>`
	`<prompt>`
	`<reprompt>`
	`<return>`
	`<script>`
	`<submit>`
	`<throw>`
	`<var>`
	`<log>`
	`<send>`

See Also

- VoiceXML 1.0 Specification:
  <catch>
- Variable:
  "_event" on page 212
  "_message" on page 212
- Related tags:
  "<error>" on page 90
  "<help>" on page 120
  "<noinput>" on page 136
  "<nomatch>" on page 138
  "<rethrow>" on page 162
  "<throw>" on page 183

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="stars">
    <catch event="nomatch">
      <prompt>
        Sorry, I did not hear any number more than ten.
      </prompt>
      <reprompt/>
    </catch>
    <block name="numbergame">
      You can create games with numbers by using Javascript
    </block>
    <field name="mynumber" type="number">
      <prompt>
        Tell me a number and I can repeat it for you.
        You can say help for information.
      </prompt>
      <catch event="help">
        <prompt>
          Please say a number more than 10
          and less than infinity.
        </prompt>
      </catch>
      <filled>
        <if cond="mynumber &gt; 10">
          <prompt>
            The number you said is <value expr="mynumber"/>
          </prompt>
        <else/>
          <clear namelist="mynumber"/>
          <throw event="nomatch"/>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

<choice>

Define a menu item.

Syntax

```
<choice
    accept="exact"|"approximate"
    next="URL"
    event="event"
    expr="js_expression"
    dtmf="dtmf_sequence"
    caching="safe"|"fast"
    fetchhint="prefetch"|"safe"
    fetchtimeout="time_interval"
    fetchaudio="URL"
    maxage="time_interval"
    maxstale="time_interval" >
  Choice Text
</choice>
```

Description

Attribute	Description
`accept`	*Extension.* Specifies whether the default grammar generated for this `<choice>` element requires all words or accepts a subset of the words; overrides the accept attribute of the parent `<menu>` element. • `exact` - Requires the user to say the exact phrase that appears in the <choice> element. • `approximate` - Allows the user to say a subset of the words in the <choice> element. **Note:** For backward compatibility with his extension, the default is `"approximate"` if the `version` attribute of the containing `<vxml>` element is less than `2.0` or unspecified. The default is `"exact"` if the `version` attribute specifies `2.0` or greater.
`next`	URL of the dialog or document to visit when this choice is selected. *Optional* (as an alternative to `event` or `expr`).
`event`	An event to throw when this choice is selected. *Optional* (as an alternative to `next` or `expr`).
`expr`	JavaScript expression that evaluates to a URL of the dialog or document to visit when this choice is selected. *Optional* (as an alternative to `next` or `event`).
`dtmf`	DTMF sequence that can be used to select this choice. *Optional* (default is no DTMF value).
`caching`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchhint`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchtimeout`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchaudio`	See Chapter 4, "Fetching Resources". *Optional.*

Attribute	Description
`maxage`	*Extension*. See Chapter 4, “Fetching Resources”. *Optional.*
`maxstale`	*Extension*. See Chapter 4, “Fetching Resources”. *Optional.*

One and only one of the `next`, `event`, or expr attributes must be specified.

Usage

Parents	Children

`<menu>`	`<audio>`
	`<value>`
	`<grammar>`
	`<break>`
	`<div>`
	`<emp>`
	`<pros>`
	`<sayas>`
	`<say-as>`

See Also

- VoiceXML 1.0 Specification:
  <choice>
- Related tag:
  “<menu>” on page 131

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">
  <menu>
    <prompt bargein="true">
      Welcome to BeVocal Home.
      <enumerate>  For
        <value expr="_prompt"/> say <value expr="_prompt"/>
      </enumerate>
    </prompt>
    <choice next="SODrivingDirections.vxml">driving directions</choice>
    <choice next="SOStockQuotes.vxml">stock quotes</choice>
    <choice next="SOBiznessFinder.vxml"> bizness finder</choice>
  </menu>
</vxml>
```

<clear>

Clear one or more form item variables.

Syntax

```
<clear>
   namelist="variable1 ..."/>
```

Description

Attribute	Description
`namelist`	Space separated list of form item variables to reset. *Optional* (default behavior clears all form item variables in the current form).

When the interpreter resets a form item, it:

- Sets the form item variable's value to `"undefined"`.
- Resets the form item's prompt and event counters to `"0"`.

Usage

Parents	Children

`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <clear>
- Related tags:
  "<field>" on page 95
  "<record>" on page 157
  "<object>" on page 140
  "<subdialog>" on page 177
  "<transfer>" on page 185
  "<block>" on page 63
  "<initial>" on page 124

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="form1">
    <field name="ssn" type="digits">
      <prompt>
        Please say your  S S N   number.
      </prompt>
    </field>
    <field name="passcode" type="digits">
      <prompt>
        Please say your Pass code number
      </prompt>
    </field>
    <field name="choice" type="boolean">
      <prompt>
        Your S S N is
        <value expr="ssn"/>
        and your   passcode is
        <value expr="passcode"/>.
        Are the values right ?
      </prompt>
      <filled>
        <if cond="choice">
          <prompt> That is good. </prompt>
        <else/>
          <clear/>
          <prompt>Let’s do it again </prompt>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

<data>

Experimental Extension. Fetch arbitrary XML data from an HTTP server, or submit values to a server.

Syntax

```
<data
   src="URL"
   name="string"
   expr="js_expression"
   method="get"|"post"
   namelist="variable1 ..."
   enctype=MIME_type
   caching="safe"|"fast"
   fetchhint="prefetch"|"safe"
   fetchtimeout="time_interval"
   maxage="time_interval"
   maxstale="time_interval" />
```

Description

The `<data>` tag fetches or submits data without transitioning to a new VoiceXML document.

The XML data fetched by the `<data>` element is returned in a read-only JavaScript variable via an object model as specified in the W3C Document Object Model (DOM).

Note: BeVocal is providing the current implementation before the standard is finalized to give our developers the opportunity to use the tag and provide feedback, which we can pass on to the W3C. If `<data>` is standardized, the BeVocal implementation will change as necessary to match the VoiceXML standard. If such changes occur, we will attempt to maintain backwards compatibility with the current implementation.

Attribute	**Description**
`src`	URL specifying the location of the XML data. *Optional* (as alternative to `expr`).
`name`	Variable name, which must be a valid JavaScript identifier and may not be a reserved keyword in either JavaScript or Java. If the `name` attribute is omitted, the HTTP request is submitted, but the retrieved data is ignored.
`expr`	JavaScript expression that evaluates to the URL of the XML data. *Optional* (as alternative to `src`).
`method`	The query request method, either `"get"` or `"post"`. *Optional* (default is `"get"`).

Attribute	Description

`enctype`	MIME encoding used when submitting data with the "post" method. *Optional* (default is `application/x-www-form-urlencoded`).
	The supported types are:
	`application/x-www-form-urlencoded`
	`multipart/form-data`
	The type `multipart/form-data` is more efficient when submitting large amounts of binary data.

`namelist`	Space separated list of variables to submit to the server. *Optional* (default is to submit no variables). This attribute can specify both VoiceXML variables and JavaScript variables, including variables that have not been explicitly declared.
`caching`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchhint`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchtimeout`	See Chapter 4, "Fetching Resources". *Optional.*
`maxage`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*
`maxstale`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*

If a `<data>` element names a variable that is already in scope, it does not declare a new variable with the same name, but simply assigns a value to the existing variable—the variable is assigned a reference to the DOM returned from the server.

Usage

Parents	Children

`<form>`
`<vxml>`
`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

None

<debug>

Deprecated. Generate debug information.

Syntax

```
<debug
    expr="js_expression" />
```

Description

Extension: The `debug` tag is a BeVocal extension to the VoiceXML 1.0 specification. It has been replaced by the `<log>` tag.

Note: It is recommended that you use the `<log>` tag, instead of this tag. The `<log>` tag is an extension which will be supported going forward.

Attribute	Description
`expr`	JavaScript expression to evaluate and enter into the debug log.

Usage

Parents	Children

`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

- Related tag:
  "<log>" on page 129

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="foo">
    <block name="dbg">
      <debug expr="num"/>
      <debug expr="fruit"/>
    </block>
    <field name="num" type="number">
      <prompt>Say a number.</prompt>
    </field>
    <field name="fruit">
      <grammar> [ apples oranges ] </grammar>
      <prompt>Do you want apples or oranges?</prompt>
    </field>
    <filled mode="any" namelist="num fruit">
      <clear namelist="dbg"/>
    </filled>
    <block>
      <debug expr="’end of form foo reached’"/>
    </block>
  </form>
</vxml>
```

<disconnect>

Disconnect a telephone session.

Syntax

```
<disconnect/>
```

Description

Forces the execution environment to disconnect the telephone call with the user. Throws a `telephone.disconnect.hangup` event. If an event handler catches this event, it can perform one last `<submit>` to notify the server that the call has ended. Because the call is no longer connected, any VoiceXML document returned from the server will be ignored and the interpreter will exit.

Usage

Parents	Children

`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <disconnect>
- Related tag:
  "<transfer>" on page 185

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="foo">
    <catch event="telephone">
      <debug expr="code"/>
    </catch>
    <field name="code" type="digits">
      <prompt> Say your passcode now.  </prompt>
    </field>
    <block>
      <prompt> That was the last form item. </prompt>
      <disconnect/>
    </block>
  </form>
</vxml>
```

<div>

Java Speech Markup Language (JSML) element to classify a region of text as a particular type.

Syntax

```
<div
    type="sentence"|"paragraph">
  Text
</div>
```

Description

Identifies enclosed text as a particular type for interpretive purposes.

The contained text is spoken normally; `<div>` has no effect.

Usage

Parents	Children

`<choice>`
`<prompt>`
`<enumerate>`
`<audio>`
`<div>`
`<emp>`
`<pros>`
	`<audio>`
	`<enumerate>`
	`<value>`
	`<break>`
	`<div>`
	`<emp>`
	`<pros>`
	`<sayas>`
	`<say-as>`

See Also

- VoiceXML 1.0 Specification:
  <div>
- Related tags:
  "<break>" on page 65
  "<emp>" on page 87
  "<pros>" on page 156
  "<sayas>" on page 168

<dtmf>

Deprecated. Specify a touch-tone key grammar.

Syntax

```
<dtmf
    scope="document"|"dialog"
    src="URL"
    expr="js_expression"
    type="MIME_type"
    caching="safe"|"fast"
    fetchhint="prefetch"|"safe"
    fetchtimeout="time_interval"
    universal="string" >
  Optional Inline DTMF Grammar

</dtmf>
```

Description

Defines a grammar for telephone key press sequences.

Note: This tag may be removed in a future release of the BeVocal VoiceXML Interpreter. We strongly recommend that all new applications use the `<grammar>` tag instead.

Attribute	Description
`scope`	Sets the scope of the DTMF grammar. • `document` - the grammar will be active throughout the current document. If the document is the application root document, then it will be active throughout the application (application scope). • `dialog` - the grammar is active throughout the current form. **Note:** A `<dtmf>` element can include a scope attribute *only* if its parent is a `<form>` element. *Optional* (default is `dialog`). The scope of any other `<dtmf>` element is determined by its parent: • If the parent is a field item, the grammar has field scope. • If the parent is a link, the scope is the element that contains the link. • If the parent is a menu choice, the grammar scope is specified by the `scope` property of the containing `<menu>` element (or dialog scope by default).
`src`	URL of the DTMF grammar specification, when it is contained in an external file. *Optional* (as an alternative to an inline DTMF grammar).
`expr`	*Extension.* JavaScript expression that evaluates to grammar file URL. Optional (as alternative to `src`).

Attribute	Description
`type`	MIME type of the DTMF grammar. *Optional* (default is `"application/x-gsl"`). The currently supported types are: • `application/x-gsl` - Nuance GSL • `application/x-gsc` - Nuance Condensed Grammar If you specify an unsupported type, an error is thrown.
`caching`	See Chapter 4, “Fetching Resources”. *Optional.*
`fetchhint`	See Chapter 4, “Fetching Resources”. *Optional.*
`fetchtimeout`	See Chapter 4, “Fetching Resources”. *Optional.*
`universal`	*Extension.* Makes this grammar a “universal” grammar with the specified name so that it can be activated and deactivated using the `universals` property. This attribute does not affect the scope of the grammar; it simply assigns it to a universal category.

Usage

Parents	Children

`<form>`
`<field>`
`<link>`
`<transfer>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <dtmf>
- Related tag:
  “<grammar>” on page 115

<else>

Mark an else clause within an `<if>` element.

Syntax

```
<else/>
```

Description

Empty tag that marks an else clause within an `<if>` element.

Usage

Parents	Children
`<if>`	None.

See Also

- VoiceXML 1.0 Specification:
  <else>
- Related tags:
  "<if>" on page 122
  "<elseif>" on page 85

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="foo">
    <field name="color">
      <grammar>[black white green blue purple yellow red]</grammar>
      <prompt>
        If you say your favorite color, I shall tell you its
        hexa decimal color code. What <emp>color? </emp>
      </prompt>
      <filled>
        <if cond="color == ’black’">
          <assign name="color_code" expr="’000000’"/>
        <elseif cond="color == ’white’"/>
          <assign name="color_code" expr="’FFFFFF’"/>
        <elseif cond="color == ’green’"/>
          <assign name="color_code" expr="’00FF00’"/>
        <elseif cond="color == ’blue’"/>
          <assign name="color_code" expr="’0000FF’"/>
        <elseif cond="color == ’purple’"/>
          <assign name="color_code" expr="’7D26CD’"/>
        <elseif cond="color == ’yellow’"/>
          <assign name="color_code" expr="’8B8B00’"/>
        <elseif cond="color == ’red’"/>
          <assign name="color_code" expr="’CD0000’"/>
        <else/>
          <assign name="color_code" expr="’?’"/>
        </if>
        <prompt>
          The code for <value expr="color"/> is <value expr="color_code"/>
        </prompt>
        <clear namelist="color color_code"/>
      </filled>
    </field>
  </form>
</vxml>
```

<elseif>

Mark an else-if clause within an `<if>` element.

Syntax

```
<elseif
    cond="js_expression"/>
```

Description

Attribute	Description
`cond`	JavaScript boolean expression that must evaluate to `"true"` for the clause to execute.

Usage

Parents	Children
`<if>`	None.

See Also

- VoiceXML 1.0 Specification:
  <elseif>
- Related tags:
  "<if>" on page 122
  "<else>" on page 83

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="foo">
    <field name="color">
      <grammar>[black white green blue purple yellow red]</grammar>
      <prompt>
        If you say your favorite color, I shall tell your its
        hexa decimal color code. What <emp>color? </emp>
      </prompt>
      <filled>
        <if cond="color == ’black’">
          <assign name="color_code" expr="’000000’"/>
        <elseif cond="color == ’white’"/>
          <assign name="color_code" expr="’FFFFFF’"/>
        <elseif cond="color == ’green’"/>
          <assign name="color_code" expr="’00FF00’"/>
        <elseif cond="color == ’blue’"/>
          <assign name="color_code" expr="’0000FF’"/>
        <elseif cond="color == ’purple’"/>
          <assign name="color_code" expr="’7D26CD’"/>
        <elseif cond="color == ’yellow’"/>
          <assign name="color_code" expr="’8B8B00’"/>
        <elseif cond="color == ’red’"/>
          <assign name="color_code" expr="’CD0000’"/>
        <else/>
          <assign name="color_code" expr="’?’"/>
        </if>
        <prompt>
          The code for <value expr="color"/> is <value expr="color_code"/>
        </prompt>
        <clear namelist="color color-code"/>
      </filled>
    </field>
  </form>
</vxml>
```

<emp>

Java Speech Markup Language (JSML) element to change the emphasis of speech output.

Syntax

```
<emp
    level="strong"|"moderate"|"none"|"reduced">
  Text
</emp>
```

Description

Any attribute is ignored; the contained text is spoken normally.

Attribute	Description
`level`	Level of emphasis to use when speaking the enclosed text. *Optional* (default is `"moderate"`). Possible values are: • `strong` • `moderate` • `none` • `reduced`

Usage

Parents	Children

`<choice>`
`<prompt>`
`<enumerate>`
`<audio>`
`<div>`
`<emp>`
`<pros>`
	`<audio>`
	`<enumerate>`
	`<value>`
	`<break>`
	`<div>`
	`<emp>`
	`<pros>`
	`<sayas>`
	`<say-as>`

See Also

- VoiceXML 1.0 Specification:
  <emp>
- Related tags:
  "<break>" on page 65
  "<div>" on page 80
  "<pros>" on page 156
  "<sayas>" on page 168

<enumerate>

Shorthand for enumerating the options in a field or the choices in a menu.

Syntax

```
<enumerate>
  Optional Template Content
</enumerate>
```

Description

Automatically generates a description of acceptable input based on the template you provide. An `<enumerate>` element may be used in prompts and event handlers within `<menu>` elements and within `<field>` elements that contain `<option>` elements; an `error.semantic` event is though if it is used elsewhere.

When this tag is in a `<menu>` element or a prompt or event handler that is executed while a menu is active, it enumerates all `<choice>` elements within the active menu. When this tag is in a `<field>` element or a prompt or event handler that is executed while a field is active, it enumerates all `<option>` elements within the active field.

Two special variables are available for use in template content for auto-generated text:

Variable	Meaning
`_prompt`	Prompt of current choice.
`_dtmf`	DTMF sequence assigned to current choice.

If this tag has no content, the generated text simply lists the prompts from the option or choice elements in the field or menu.

Usage

Parents	Children

`<menu>`
`<prompt>`
`<field>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<audio>`
`<div>`
`<emp>`
`<pros>`
`<if>`
	`<audio>`
	`<value>`
	`<break>`
	`<div>`
	`<emp>`
	`<pros>`
	`<sayas>`
	`<say-as>`

See Also

- VoiceXML 1.0 Specification:
  <enumerate>

- Related tags:
  “<menu>” on page 131
  “<field>” on page 95

Examples

If you run this example, the menu’s prompt will be:

“Welcome to BeVocal Home. For driving directions, say driving directions. For stock quotes, say stock quotes. For business finder, say business finder.”

```
<?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">
  <menu>
    <prompt bargein="true">
      Welcome to BeVocal Home.
      <enumerate>  For
        <value expr="_prompt"/> say <value expr="_prompt"/>
      </enumerate>
    </prompt>
    <choice next="SODrivingDirections.vxml">driving directions</choice>
    <choice next="SOStockQuotes.vxml">stock quotes</choice>
    <choice next="SOBiznessFinder.vxml"> bizness finder</choice>
  </menu>
 </vxml>
```

<error>

Catch an error event.

Syntax

```
<error
   count="integer"
   cond="js_expression">
  Executable Content
</error>
```

Description

Shorthand for `<catch event="error">`. Catches error events of all kinds.

Attribute	Description
`count`	Minimum number of times an error must have occurred during a form or menu invocation. *Optional* (default is `"1"`).
`cond`	JavaScript expression that must also evaluate to `"true"` for an event to be caught. *Optional* (default is `"true"`).

If multiple error handlers are defined in, or inherited by, the element in which the error occurs, one handler is chosen based on event count, scope, and document order. See Chapter 3, “Event Handling”.

Tips:

- Within an event handler, the `_event` variable contains the name of the event currently being handled; the `_message` variable contains the message string that provides additional information about the event. If no message was supplied when the event was thrown, the `_message` variable is `"undefined"`.

Usage

Parents	Children

`<vxml>`
`<form>`
`<menu>`
`<field>`
`<initial>`
`<record>`
`<transfer>`
`<subdialog>`
`<object>`
	`<audio>`
	`<enumerate>`
	`<value>`
	`<assign>`
	`<clear>`
	`<disconnect>`
	`<exit>`
	`<goto>`
	`<if>`
	`<prompt>`
	`<reprompt>`
	`<return>`
	`<script>`
	`<submit>`
	`<throw>`
	`<var>`
	`<log>`
	`<send>`

See Also

- VoiceXML 1.0 Specification:
  <error>
- Variable:
  "_event" on page 212
  "_message" on page 212
- Related tags:
  "<catch>" on page 67
  "<help>" on page 120
  "<noinput>" on page 136
  "<nomatch>" on page 138
  "<rethrow>" on page 162
  "<throw>" on page 183

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="foo">
    <block>
      <audio
src="http://cafe.bevocal.com/libraries/audio/female1/en_us/common/pleasetryback.wav
"/>
      I am sorry, There was a problem -- please try back later.
    </block>
    <error>
      <prompt>Error occurred</prompt>
      <exit/>
    </error>
    <block name="debuginfo">
      <debug name="error1" expr="Test for Error: PASSED"/>
    </block>
  </form>
</vxml>
```

<exit>

Exit a session.

Syntax

```
<exit
   expr - not implemented
   namelist - not implemented />
```

Description

Unloads all documents and returns control to the interpreter's execution environment.

Attribute	Description
`expr`	*Not implemented.* JavaScript expression that evaluates to the value to return to the execution environment. *Optional* (as alternative to namelist).
`namelist`	*Not implemented.* Space separated list of variable names to return to the interpreter execution environment. *Optional* (default is to return nothing).

The `expr` and `namelist` attributes are not meaningful because the BeVocal execution context does not accept return values from the execution of a VoiceXML document.

Usage

Parents	Children

`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <exit>

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="foo">
    <catch event="telephone">
      <debug expr="’user said disconnect’"/>
    </catch>
    <field name="choose">
      <grammar> [ exit disconnect ] </grammar>
      <prompt>Please say exit</prompt>
    </field>
    <block>
      <if cond="choose==’exit’">
        <exit/>
        <prompt> you should NOT hear this prompt! </prompt>
      </if>
      <disconnect/>
    </block>
  </form>
</vxml>
```

<field>

Declare an input field in a form.

Syntax

```
<field
   name="string"
   expr="js_expression"
   cond="js_expression"
   type="boolean"|"date"|"digits"|"currency"|
        "number"|"phone"|"time"|"airport"|
        "airline"|"equity"|"street"|
        "streetnumber"|"citystate"
   slot="string"
   modal="true"|"false">
  Child Elements
</field>
```

Description

Field item that prompts user for a value that matches a particular grammar.

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; its name must be unique among all VoiceXML and JavaScript variables within the form's scope.
`expr`	JavaScript expression that assigns the initial value of the field item variable for this field. *Optional* (default is `"undefined"`). If you set the field item variable to a value other than `"undefined",` you'll need to clear it before the field can execute.
`cond`	JavaScript boolean expression that also must evaluate to `"true"` for the field to execute. *Optional* (default is `"true"`). If not specified, the value of the field item variable alone determines whether or not the field can execute.

Attribute	Description
`type`	Specifies an internal grammar. *Optional* (as alternative to `<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). *Extensions:* • `airport` - Grammar for recognizing airport codes, such as `DFW` • `airline` - Grammar for recognizing airline codes, such as `AA` • `equity` - Grammar for recognizing company symbol or full name, such as `ibm` or `cisco systems` • `street` - Grammar for recognizing street name (with or without street number), such as `bordeaux drive` or `1380 bordeaux drive` • `streetnumber` - Grammar for recognizing street number • `citystate` - Grammar for recognizing city name and state name separated by a comma, such as `sunnyvale, california`
`slot`	If this field is part of a mixed-initiative dialog, the name of the grammar slot that will be used to assign a value to the field item variable for this field. *Optional* (defaults to variable name). This attribute is ignored by a field-level grammar. See BeVocal Grammar Reference for more information on grammar slots.
`modal`	Boolean value that must be `"true"` to temporarily turn off higher level grammars. *Optional* (default is `"false"`). Lets you alter default behavior so that only this field's grammars are active while the field executes. This attribute turns off both independent grammars and rule sets.

Properties of the Shadow Variable. Corresponding to the field item variable *`name`* is a "shadow variable" called *`name$`*. After the field item variable is filled, some additional information is available in the following properties of this shadow variable:

- `confidence` - The recognition confidence level (with 0.0 representing the lowest confidence and 1.0 representing the highest).
- `utterance` - A string representation of the words actually spoken by the caller.

- `inputmode` - The mode in which input was provided, one of `"voice"` or `"dtmf"`.

Note: Shadow variable properties are not set for a field of an extended built-in type, such are `"airline"`.

For a field whose name is *`name`*, you access the property *`propName`* of the shadow variable with the syntax:

```
name$.property
```

For example, you access the `confidence` property for the `color` field as:

```
color$.confidence
```

Built-In Types - Parameters. Some built-in field types can be parameterized to affect the built-in type's behavior. The following table shows the type that can be parameterized and indicates which input parameters must be specified.

Field Type	Input Parameters
`boolean`	• `y` - The DTMF keypress for an affirmative response. • `n` - The DTMF keypress for a negative answer.
`digits`	• `minlength` - The minimum number of digits in a valid response. • `maxlength` - The maximum number of digits in a valid response. • `length` - The exact number of digits in a valid response.
`street`	• `city` - The city in which the street is located (required). • `state` - The state in which the specified city is located (required).
`streetnumber`	• `street` - The street in which the street number is located (required). • `city` - The city in which the specified street is located (required). • `state` - The state in which the specified city is located (required).

A parameter is specified with in `type` attribute with syntax of the form:

```
typeName ? parameter = value
```

For example the following element specifies a field of type `digits` that must contain exactly five digits:

```
<field name="mydigits" type="digits?length=5">
  ...
```

More than one parameter may be specified separated by semicolons.

Extended Types - Input. The following table shows default prompts and example user inputs for each extended built-in type. The default prompt is the question with which the system prompts the user for a field value.

Field Type	Default Prompt	Example Inputs
`airport`	Name an airport or its city (Prompts for disambiguation)	San Jose DFW
`airline`	Please say the airline	American UA
`equity`	Name a company or index	Cisco ORCL
`street`	Say a street name in *city*	Bordeaux Drive
`streetnumber`	Say the street number	`1380`
`citystate`	Name a city and state	Sunnyvale, California

Extended Types - Properties. Some built-in field types have JavaScript properties that can be accessed once the value of the field has been filled. The following table lists the properties of the extended built-in types.

Field Type	Properties
`airport`	`city, state, code`
`airline`	`code, name`
`equity`	`name, symbol`
`street`	-
`streetnumber`	-
`citystate`	`city, state`

A property is accessed with an expression of the form:

`fieldName . propertyName`

For example the `city` property of a `citystate` field is accessed as follows:

```
<field name="mycity" type="citystate">
  <filled>
   <prompt>The city is <value expr="mycity.city"/>
  </filled>
</field>
```

Extended Types - Output. The following table describes the audio and string representation for a value of each extended built-in type.

Field Type	Audio Output	String Result
`airport`	Airport name	Dallas Fort Worth International Airport (DFW)
`airline`	Airline name	American Airlines (AA)
`equity`	Company name	ORCL

Field Type	Audio Output	String Result
`street`	Street name	Bordeaux Ave
`streetnumber`	Street number	1380 Bordeaux Ave
`citystate`	City State	Sunnyvale

Usage

Parents	Children

`<form>`	`<audio>`
	`<enumerate>`
	`<value>`
	`<catch>`
	`<help>`
	`<noinput>`
	`<nomatch>`
	`<error>`
	`<filled>`
	`<dtmf>`
	`<grammar>`
	`<link>`
	`<option>`
	`<property>`

See Also

- VoiceXML 1.0 Specification:
  <field>
- BeVocal Grammar Reference
- Related tags:
  “<filled>” on page 107
  “<grammar>” on page 115

Examples

Example 1 - no type:

```
<?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="foo">
    <field name="name">
      <prompt> What is your favorite color?  </prompt>
      <grammar>
        [ red green yellow blue orange ]
      </grammar>
      <filled>
        <prompt> Your favorite color is <value expr="name"/> </prompt>
         </filled>
      </field>
 </form>
</vxml>
```

Example 2 - boolean type:

```
<?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="form1">
    <field name="new_file" type="boolean">
      <prompt>
        Do you want to play the game again?
      </prompt>
      <filled>
        <prompt>
          Your answer is <value expr="new_file"/>
        </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

Example 3 - date type:

```
<?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="form1">
    <field name="today" type="date">
      <prompt>
        What date is today?  Please say or enter  month day and year.
      </prompt>
      <filled>
        <prompt>
          Your answer is <value expr="today"/>
        </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

Example 4 - digits type:

```
<?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="form1">
    <field name="card_num" type="digits">
      <prompt>
        Please say or  enter the last four  digits  of your  credit card.
      </prompt>
      <filled>
        <if cond="card_num.length != 4">
          <prompt>
            Sorry, I didn’t hear exactly four  digits.
          </prompt>
          <clear/>
          <reprompt/>
        <else/>
          <prompt>The number you entered is <value expr="card_num"/></prompt>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

Example 5 - currency type:

```
<?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="form1">
    <field name="ticket_cost" type="currency">
      <prompt>
        Please say the cost of your ticket.
      </prompt>
      <filled>
        <prompt>The cost you entered is <value expr="ticket_cost"/></prompt>
      </filled>
    </field>
  </form>
</vxml>
```

Example 6 - number type:

```
<?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="form1">
    <field name="num_count" type="number">
      <prompt>
        Please say  the number of computers  you want to order.
      </prompt>
      <filled>
        <prompt>The number you entered is <value expr="num_count"/></prompt>
      </filled>
    </field>
  </form>
</vxml>
```

Example 7 - phone type:

```
<?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="form1">
    <field name="phone_num" type="phone">
      <prompt>
        Please say or enter your phone number.
      </prompt>
      <filled>
        <prompt>The number you entered is <value expr="phone_num"/></prompt>
      </filled>
    </field>
  </form>
</vxml>
```

Example 8 - time type:

```
<?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="form1">
    <field name="meeting_time" type="time">
      <prompt>
        Please say the meeting time.
      </prompt>
      <filled>
        <prompt>The meeting time is <value expr="meeting_time"/></prompt>
      </filled>
    </field>

  </form>
</vxml>
```

Example 9 - airport type:

```
<?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="form1">
    <field name="airport1" type="airport">
      <filled>
        <prompt> The value is <value expr="airport1"/> </prompt>
        <prompt> The city is <value expr="airport1.city"/> </prompt>
        <prompt> The state is <value expr="airport1.state"/> </prompt>
        <prompt> The code is <value expr="airport1.code"/> </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

Example 10 - airline type:

```
<?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="form1">
    <field name="airline1" type="airline">
      <filled>
        <prompt> The value is <value expr="airline1"/> </prompt>
        <prompt> The name is <value expr="airline1.name"/> </prompt>
        <prompt> The code is <value expr="airline1.code"/> </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

Example 11 - equity type:

```
<?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="equityForm">
    <field name="equity1" type="equity">
      <filled>
        <prompt> The value is <value expr="equity1"/> </prompt>
        <prompt> The name is <value expr="equity1.name"/> </prompt>
        <prompt> The symbol is <value expr="equity1.symbol"/> </prompt>
        <clear/>
      </filled>
    </field>
  </form>
</vxml>
```

Example 12 - street type:

```
<?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="form1">
    <field name="street1" type="street?city=austin;state=tx">
      <filled>
        <prompt> The value is <value expr="street1"/> </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

Example 13 - streetnumber type:

```
<?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="form1">
    <field name="streetnumber1"
      type="streetnumber?city=austin;state=tx;street=heiden_lane">
      <filled>
        <prompt> The value is <value expr="streetnumber1"/> </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

Example 14 - citystate type:

```
<?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="form1">
    <field name="citystate1" type="citystate">
      <filled>
        <prompt> The value is <value expr="citystate1"/> </prompt>
        <prompt> The city is <value expr="citystate1.city"/> </prompt>
        <prompt> The state is <value expr="citystate1.state"/> </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

<filled>

Contain actions to be executed when fields are filled.

Syntax

```
<filled
    mode="any"|"all"
    namelist="variable1 ...">
  Child Elements
</filled>
```

Description

A `<filled>` element can be either the child of a field item or the child of a form.

- When used as the child of a field item, the `<filled>` element has no attributes; its action is taken when the last user input fills the field item variable of the containing element.
- When used as the child of a form, the `<filled>` element may have attributes that specify when its action is taken.

Attribute	Description
`mode`	A value of `"any"` causes execution of this element when the last user input fills any one of the specified field item variables. A value of `"all"` causes execution of this element when all specified field item variables are filled; the last user input must have filled at least one of the fields. *Optional* (default value is `"all"`).
`namelist`	Space separated list of names of the field item variables whose filling can trigger this element. *Optional* (default is all field item variables in the form).

It is an error to specify attributes in a `<filled>` element within a field item.

Usage

Parents	Children

`<form>`
`<field>`
`<record>`
`<transfer>`
`<subdialog>`
`<object>`
	`<audio>`
	`<value>`
	`<assign>`
	`<clear>`
	`<disconnect>`
	`<exit>`
	`<goto>`
	`<if>`
	`<prompt>`
	`<reprompt>`
	`<return>`
	`<script>`
	`<submit>`
	`<throw>`
	`<var>`
	`<log>`
	`<send>`

See Also

- VoiceXML 1.0 Specification:
  <filled>
- Related tag:
  "<field>" on page 95

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="foo">
    <field name="name">
      <prompt> What is your favorite color ? </prompt>
      <grammar>
        [ red green yellow blue orange ]
      </grammar>
      <filled>
        <prompt> Your favorite color is <value expr="name"/> </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

<form>

Present information and collect data.

Syntax

```
<form
   id="string"
   scope="document"|"dialog">
  Child Elements
</form>
```

Description

A dialog for collecting user input. Note that when you reach the end of a form, execution does *not* proceed to the next form on the page. If you do not explicitly transition to another dialog using `<goto>` or `<submit>`, then the application terminates when the form completes.

Attribute	**Description**
`id`	The name of the form. VoiceXML Reference
`scope`	Sets the default scope of the form's grammars. *Optional* (default is `"dialog"`). • `document` - The form's grammars are active throughout the current document. If the document is the application root document, then they are active throughout the application (application scope). • `dialog` - By default, the form's grammars have dialog scope, which means they will be active only in this form.

Usage

Parents	**Children**

`<vxml>`	`<dtmf>`
	`<grammar>`
	`<catch>`
	`<help>`
	`<noinput>`
	`<nomatch>`
	`<error>`
	`<filled>`
	`<initial>`
	`<object>`
	`<link>`
	`<property>`
	`<record>`
	`<subdialog>`
	`<transfer>`
	`<block>`
	`<field>`
	`<var>`
	`<script>`

See Also

- VoiceXML 1.0 Specification:
  <form>
- Related tags:
  “<field>” on page 95
  “<block>” on page 63
  “<record>” on page 157
  “<transfer>” on page 185
  “<object>” on page 140
  “<subdialog>” on page 177
  “<initial>” on page 124
  “<grammar>” on page 115

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="welcome">
    <block>You are in Form One.  Welcome back home! </block>
    <field name="hello">
      <grammar>[(next) (dtmf-1) start]</grammar>
      <prompt>
        Say next or press one to go to Form 2 or start to start over again.
      </prompt>
      <filled>
        <if cond="hello==’next’|| hello==’dtmf-1’">
          <goto next="#comeagain"/>
        <else/>
          <goto next="#welcome"/>
        </if>
      </filled>
    </field>
  </form>
  <form id="comeagain">
    <block>You are now in Form 2.</block>
    <field name="goback" type="boolean">
      <grammar>[(back) (dtmf-2) continue]</grammar>
      <prompt>
        Say back or press two to go to Form 1 or say continue
        to enter this form again.
      </prompt>
      <filled>
        <if cond="goback==’back’|| goback==’dtmf-2’">
          <prompt>Thanks for stopping by Form 2. Please come again.</prompt>
          <goto next="#welcome"/>
        <else/>
          <goto next="#comeagain"/>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

<goto>

Go to another location in the same or different document.

Syntax

```
<goto
   next="URL"
   expr="js_expression"
   expritem="js_expression"
   nextitem="URL"
   submit="variable1 ..."
   method="get"|"post"
   caching="safe"|"fast"
   fetchhint="prefetch"|"safe"
   fetchtimeout="time_interval"
   fetchaudio="URL"
   maxage="time_interval"
   maxstale="time_interval" />
```

Description

Transitions to another item in the same form, to another dialog in the same document, or to a different document. Except when the transition is to another item in the same form, the transition will cause all values stored in the dialog's variables to be lost. This is true even if you transition into the same dialog as you were in before.

The VoiceXML interpreter clears its prompt queue when transitioning to another form item. You will not be able to barge in during prompts played in the execution of a `<goto>` tag.

Attribute	**Description**
`next`	The URL to go to next. *Optional* (as alternative to `expr`, `expritem`, `nextitem`).
`expr`	JavaScript expression that evaluates to the URL to go to next. *Optional* (as alternative to `next`, `expritem`, `nextitem`).
`expritem`	JavaScript expression that evaluates to the name of the next item in the current form to visit next. *Optional* (as alternative to `next`, `expr`, `nextitem`).
`nextitem`	Name of the next item in the current form to visit next. *Optional* (as alternative to `next`, `expr`, `expritem`, `nextitem`).
`submit`	*Extension*. Space separated list of variables to submit. *Optional* (default is to submit no form item variables).
`method`	*Extension*. The query request method. *Optional* (default is `"get"`).
`caching`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchhint`	See Chapter 4, "Fetching Resources". *Optional.* **Note:** If this element sends variables to the server with the `submit` attribute, a `"prefetch"` value for the `fetchhint` attribute is ignored.

Attribute	Description
`fetchtimeout`	See Chapter 4, “Fetching Resources”. *Optional.*
`fetchaudio`	See Chapter 4, “Fetching Resources”. *Optional.*
`maxage`	*Extension*. See Chapter 4, “Fetching Resources”. *Optional.*
`maxstale`	*Extension*. See Chapter 4, “Fetching Resources”. *Optional.*

One and only one of the attributes `next`, `expr`, `nextitem`, and `expritem` must be specified.

Note: the `submit` and `method` attributes are extensions, included for compatibility with Nuance VBuilder output. If you are writing your own VoiceXML code, we strongly recommend that you use the standard `<submit>` tag instead.

Usage

Parents	Children

`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <goto>
- Related tag:
  “<submit>” on page 181

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">

<!--Verifies transitions between dialogs.-->

  <var name="something" expr="0"/>
  <form id="form1">
    <block>
      <assign name="something" expr="500"/>
      <prompt>
        You are in first dialog. something is
        <value expr="something"/>
        Going to second dialog.
      </prompt>
      <goto next="#form2"/>
      <prompt>goto failed</prompt>
    </block>
  </form>
  <form id="form2">
    <block>
      <prompt>
        You are now in the second dialog something is
        <value expr="something"/>
        Thank you.
      </prompt>
    </block>
  </form>
</vxml>
```

<grammar>

Specify a speech-recognition grammar.

Syntax

```
<grammar
    xml:lang="language"
    src="URL"
    expr="js_expression"
    scope="document"|"dialog"
    type="MIME_type"
    mode="voice"|"dtmf"
    root="string"
    version="version_number"
    universal="string"
    ruleset="true"|"false" >
    caching="safe"|"fast"
    fetchhint="prefetch"|"safe"
    fetchtimeout="time_interval"
    maxage="time_interval"
    maxstale="time_interval"
  Optional Inline Grammar
</grammar>
```

Description

Defines a set of valid utterances. When the speech-recognition engine detects a match with a grammar, it may cause a transition to another dialog (if the grammar is in a menu item or link) or assign a return value to assign to a field item variable (if the grammar is in a form or field item).

Attribute	Description
`xml:lang`	*Extension*. The language and optional country local identifier for the grammar. *Optional* (default is `"en-US"`) The accepted language identifiers are: • `en` - English • `en-US` - United States English

Attribute	**Description**
`src`	URL of the grammar specification, when it is contained in an external file. *Optional* (as an alternative to an inline grammar or `expr`). The URL may have any one of the following forms: • GSL independent grammar: *`gslGrammarFileURL#rootRuleName`* • GSL rule set: *`gslGrammarFileURL`* • Independent grammar in Nuance Condensed Grammar file (the file identifies its root rule): *`gscGrammarFileURL`* • Independent grammar in XML Grammar file: *`xmlGrammarFileURL#rootRuleName`* • Independent grammar in XML Grammar file; the grammar's default rule is used as the root rule: *`xmlGrammarFileURL`* • Built-in Independent grammar: `builtin:grammar/`*`type`*
`expr`	*Extension.* JavaScript expression that evaluates to grammar file URL. Optional (as alternative to `src`).
`scope`	Sets the scope of the grammar. • `document` - the grammar will be active throughout the current document. If the document is the application root document, then it will be active throughout the application (application scope). • `dialog` - the grammar is active throughout the current form. **Note:** A `<grammar>` element can include a scope attribute *only* if its parent is a `<form>` element or a `<menu>` element. *Optional* (default is `dialog`). The scope of any other `<grammar>` element is determined by its parent: • If the parent is a field item, the grammar has field scope. • If the parent is a link, the scope is the element that contains the link. • If the parent is a menu choice, the grammar scope is specified by the `scope` property of the containing `<menu>` element (or dialog scope by default).
`type`	MIME type of the grammar. *Optional* (default is `"application/x-gsl"`). The currently supported types are: • `application/x-gsl` - Nuance GSL • `application/x-gsc` - Nuance Condensed Grammar • `application/grammar+xml` - XML Speech Grammar If you specify an unsupported type, an error is thrown.
`mode`	*Extension.* The mode of the contained or referenced grammar. *Optional* (default is `"voice"`). • `voice` - Spoken input • `speech` - *Deprecated*. Use `"voice"` instead. • `dtmf` - DTMF input. (Not supported for XML grammars.)

Attribute	Description
`root`	*Extension.* If this grammar is an inline XML grammar, this attribute can specify the name of the root grammar rule. *Optional* (if omitted, the grammar's default rule is used).
`version`	*Extension.* Version of the grammar. *Optional* (default is `"1.0"`). The only allowed version is `"1.0"`.
`universal`	*Extension.* Makes this grammar a "universal" grammar with the specified name so that it can be activated and deactivated using the `universals` property. This attribute does not affect the scope of the grammar; it simply assigns it to a universal category.
`ruleset`	*Extension.* If this grammar is an inline or external Nuance GSL grammar, this attribute can specify that it is a rule set that can be referred to from other active GSL grammars rather than being an active grammar in its own right. *Optional* (default is `"false"`.) • `true` - This grammar contains a rule set—that is, one or more named rules that can be used as subgrammars by other grammar rules that are in scope • `false` - This grammar contains an independent grammar rule, to be used as an active grammar
`caching`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchhint`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchtimeout`	See Chapter 4, "Fetching Resources". *Optional.*
`maxage`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*
`maxstale`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*

Since grammars are scoped, multiple grammars may be active at the same time. When the speech-recognition engine detects a match from a higher level, control is passed to that grammar's parent element.

Usage

Parents	Children

`<form>`
`<choice>`
`<field>`
`<link>`
`<record>`
`<transfer>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <grammar>
- Grammar Reference

Examples

Example 1 - external independent GSL grammar:

```
<?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="uscitystate">
    <field name="state">
      <grammar src="uscitystate.grammar#STATES"/>
      <prompt> Do you want California or Texas? </prompt>
    </field>
    <field name="city">
      <grammar expr="'uscitystate.grammar#CITIES_' + state"/>
      <prompt> Say a city in <value expr="state"/> </prompt>
    </field>
    <block>
      You said <value expr="city"/> and <value expr="state"/>
    </block>
  </form>
</vxml>
```

Example 2 - external GSL rule-set grammar:

```
<?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">
  <!-- Document-scoped rule sets -->
  <grammar ruleset="true" src="scopedRulesets.grammar"/>
  <grammar ruleset="true">
    DocRule [ doc ]
  </grammar>
  <catch event="docmatch">
    <prompt> Got doc level link match </prompt>
  </catch>
  <link event="docmatch">
    <grammar>
      [ DocRule FormDownRule ItemDownRule EXTERNAL_DOC_RULE ]
    </grammar>
  </link>

  <form id="firstform">
    <grammar ruleset="true">
      <!-- Form-scoped rule set with rule used in field grammar -->
      FormRule [ form EXTERNAL_FORM_RULE ]
      <!-- Form-scoped rule set with rule used in link grammar -->
      FormDownRule [ apples ]
    </grammar>
    <field name="fieldval">
      <prompt> try me out, i am cool </prompt>
      <!-- Field-scoped ruleset with rule used in link grammar -->
      <grammar ruleset="true">
        ItemDownRule [ oranges ]
      </grammar>
      <!-- Field-scoped independent grammar rule -->
      <grammar>
        ItemRule [ FormRule field ]
      </grammar>
      <filled>
        <prompt> you said <value expr="fieldval"/> </prompt>
        <clear/>
      </filled>
    </field>
  </form>
</vxml>
```

The external grammar file `scopedRulesets.grammar` follows:

```
EXTERNAL_DOC_RULE [
  external_doc
]

EXTERNAL_FORM_RULE [
  external_form
]
```

<help>

Catch a help event.

Syntax

```
<help
    count="integer"
    cond="js_expression">
  Executable Content
</help>
```

Description

Attribute	Description
`count`	Minimum number of times an error must have occurred during a form or menu invocation. *Optional* (default is `"1"`).
`cond`	JavaScript expression that must also evaluate to `"true"` for an event to be caught. *Optional* (default is `"true"`).

If multiple handlers for help events are defined in, or inherited by, the element in which the events occurs, one handler is chosen based on event count, scope, and document order.

A help event is thrown whenever user input matches the predefined `"help"` universal grammar.

- In VoiceXML 1.0, all universal grammars are active by default. You can deactivate the `"help"` grammar by setting the `universals` property. The following tag deactivates all universal grammars, including `"help"`:

  ```
  <property name="universals" value="none" />
  ```

  The following tag deactivates the `"help"` grammar and activates the other predefined universal grammars:

  ```
  <property name="universals" value="exit cancel goback"/>
  ```

- When the `<vxml>` tag's `version` attribute equals `2.0`, all universal grammars are deactivated by default. You can activate the `"help"` grammars by setting the `universals` property. The following tag activates all universal grammars, including `"help"`:

  ```
  <property name="universals" value="all" />
  ```

  The following tag activates the `"help"` grammar and deactivates the other predefined universal grammars:

  ```
  <property name="universals" value="help"/>
  ```

For additional information about universal grammars, see “Universal Grammars” on page 20.

Usage

Parents	Children
`<vxml>` `<form>` `<menu>` `<field>` `<initial>` `<record>` `<transfer>` `<subdialog>` `<object>`	`<audio>` `<enumerate>` `<value>` `<assign>` `<clear>` `<disconnect>` `<exit>` `<goto>` `<if>` `<prompt>` `<reprompt>` `<return>` `<script>` `<submit>` `<throw>` `<var>` `<log>` `<send>`

See Also

- VoiceXML 1.0 Specification:
  <help>
- Related tags:
  "<catch>" on page 67
  "<error>" on page 90
  "<noinput>" on page 136
  "<nomatch>" on page 138
  "<rethrow>" on page 162
  "<throw>" on page 183

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="foo">
    <field name="sports">
      <grammar> [ football basketball tennis skiing ] </grammar>
      <help>Please say one of football, basketball, tennis or skiing</help>
      <prompt> What is your favorite sport ?</prompt>
      <filled>
        <prompt>
          Looks like <value expr="sports"/> is your favorite sports.
        </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

<if>

Container for simple conditional logic.

Syntax

```
<if
   cond="js_expression">
  Executable Content
</if>
```

Description

Attribute	Description
`cond`	JavaScript expression that evaluates to a boolean value that must be `"true"` for the “if” clause to execute. *Optional* (default is `"true"`).

Usage

Parents	Children
`<block>` `<catch>` `<error>` `<help>` `<noinput>` `<nomatch>` `<if>` `<filled>`	`<audio>` `<enumerate>` `<value>` `<assign>` `<clear>` `<disconnect>` `<exit>` `<goto>` `<if>` `<prompt>` `<reprompt>` `<return>` `<script>` `<submit>` `<throw>` `<var>` `<elseif>` `<else>` `<log>` `<send>`

See Also

- VoiceXML 1.0 Specification:
  <if>
- Related tags:
  “<else>” on page 83
  “<elseif>” on page 85

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="form">
    <field name="hello">
      <grammar>[(one) (dtmf-1) goodbye]</grammar>
      <prompt>Say one or press one to continue or say goodbye to exit.</prompt>
      <nomatch>Sorry, I did not get it.<reprompt/></nomatch>
      <filled>
        <if cond="hello==’one’ || hello==’dtmf-1’">
          <prompt> Welcome to this part of the world. </prompt>
        <else/>
          <prompt> Sorry you could not have much fun. Goodbye </prompt>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

<initial>

Declare initial logic upon entry into a (mixed-initiative) form.

Syntax

```
<initial
    name="string"
    expr="js_expression"
    cond="js_expression">
  Child Elements
</initial>
```

Description

Control item that controls the initial interaction in a mixed initiative form. An `<initial>` element can request user input or perform other non-interactive initialization tasks at the beginning of a mixed initiative form. As with all form items, an `<initial>`'s form item variable must have a value of `"undefined"` in order to be visited.

If any of the form's fields are filled as a result of user input, the interpreter sets all `<initial>` form item variables to `"true"` before performing any `<filled>` actions. After that, it will request user input in a "directed mode" based on the prompts associated with the fields that are still unfilled.

Attribute	**Description**
`name`	Name of form item variable, which may not be a JavaScript reserved keyword. *Optional* (default is an unusable internal name). The form item variable has dialog (form) scope; its name must be unique among all VoiceXML and JavaScript variables within the form's scope. Generally, you use this attribute only if you want to control `<initial>` execution explicitly.
`expr`	JavaScript expression that assigns the initial value of the form item variable. *Optional* (default is `"undefined"`). If you set the form item variable to a value other than `"undefined"`, you'll need to clear it before the `<initial>` element can execute. Note that you need to give the form item variable a name if you want to clear it separately from other form item variables.
`cond`	JavaScript boolean expression that must also evaluate to `"true"` for the `<initial>` element to execute. *Optional* (default is `"true"`). If not specified, the value of the form item variable alone determines whether or not the `<initial>` element can execute.

Usage

Parents	Children

`<form>`	`<audio>`
	`<value>`
	`<catch>`
	`<help>`
	`<noinput>`
	`<nomatch>`
	`<error>`
	`<link>`
	`<prompt>`
	`<property>`

See Also

- VoiceXML 1.0 Specification:
  <initial>

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="foo">
    <grammar src="foo.grammar#Main"/>
    <nomatch> Okay , let me ask you one by one.
      <assign name="selection" expr="true"/>
    </nomatch>
    <initial name="selection">
      Please say how many apples or oranges you want.
    </initial>
    <field name="fruit">
      <grammar src="foo.grammar#Fruit"/>
      Do you want apples or oranges?
    </field>
    <field name="quantity">
      <grammar src="foo.grammar#Quantity"/>
      How many <value expr="fruit"/> do you want?
    </field>
    <block>
      <prompt>
        You said that you wanted
        <value expr="quantity"/>
        pieces of
        <value expr="fruit"/>.
      </prompt>
    </block>
  </form>
</vxml>
```

<link>

Specify a transition common to all dialogs in the link's scope.

Syntax

```
<link
    next="URL"
    expr="js_expression"
    event="event"
    caching="safe"|"fast"
    fetchhint="prefetch"|"safe"
    fetchtimeout="time_interval"
    fetchaudio="URL"
    maxage="time_interval"
    maxstale="time_interval" >
  Link Grammar
</link>
```

Description

Transitions to another dialog or throws an event when user input matches one of the link grammars. In the first case, execution jumps to the link's destination; in the second case execution resumes in the current dialog after the event is handled.

The VoiceXML interpreter clears the prompt queue when going to another form. You will not be able to barge in during prompts played in the execution of a `<link>` tag which goes to another form.

Attribute	**Description**
`next`	The URL to go to when user input matches one of the link grammars. *Optional* (as alternative to `expr`, `event`).
`expr`	JavaScript expression that evaluates to the URL to go to when user input matches one of the link grammars. *Optional* (as alternative to `next`, `event`).
`event`	The event to throw when user input matches one of the link grammars. *Optional* (as alternative to `next`, `expr`).
`caching`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchhint`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchtimeout`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchaudio`	See Chapter 4, "Fetching Resources". *Optional.*
`maxage`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*
`maxstale`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*

Tips:

- During application development, put a link like the following in your application root document, so that while you're calling your application you can say: "BeVocal reload" or press: *** to start the application again.

```
<vxml>
  <link caching="safe"
    next="<http://www.mysite.com/myapp.vxml>">
    <grammar>
    [
      (bevocal reload)
      (dtmf-star dtmf-star dtmf-star)
    ]
    </grammar>
  </link>
  ...
</vxml>
```

Usage

Parents	Children

`<vxml>`
`<form>`
`<field>`
`<initial>`
	`<dtmf>`
	`<grammar>`

See Also

- VoiceXML 1.0 Specification:
  <link>

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">

  <link caching="safe" next="target.vxml">
    <grammar>
      [
        (bevocal reload)
        (dtmf-star dtmf-star dtmf-star)
      ]
    </grammar>
  </link>

  <form id="form">
    <field name="welcome">
      This is a test for link tag.
      You can say bevocal reload or enter star star star to reload this
      application.
    </field>
  </form>
</vxml>
```

<log>

Extension. Write debugging information to a the BeVocal Café call log, which you can view on the Café website.

Syntax

```
<log
    label="string"
    expr="js_expression" >
  Debugging Text
</log>
```

Description

This tag is an extension to the BeVocal VoiceXML 1.0 platform. This tag replaces the *deprecated* `<debug>` tag.

Similar functionality is available within a JavaScript using the `bevocal.log` function.

Attribute	Description
`label`	A string that is added as a label to messages produced by this `<log>` element. *Optional* (default is no label on the messages).
`expr`	JavaScript expression that evaluates to a string to be added to the call log as a separate message. *Optional*.

A `<log>` element may write one or two messages to the call log. It writes one message corresponding to the `expr` attribute, if any, and one message corresponding to the contained debugging text, if any. If the element has a `label` attribute, the specified label precedes each message.

Usage

Parents	Children

`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	`<value>`

See Also

None

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="foo">
    <!-- Print current value of num and fruit variables -->
    <block name="dbg">
      <log>
        num: <value expr="num"/>
        fruit: <value expr="fruit"/>
      </log>
    </block>
    <field name="num" type="number">
      <prompt>Say a number.</prompt>
    </field>
    <field name="fruit">
      <grammar> [ apples oranges ] </grammar>
      <prompt>Do you want apples or oranges?</prompt>
    </field>
    <filled mode="any" namelist="num fruit">
      <clear namelist="dbg"/>
    </filled>
    <block>
      <log>
        End of form foo reached
      </log>
    </block>
  </form>
</vxml>
```

<menu>

Allow user to choose between alternative destinations.

Syntax

```
<menu
    id="string"
    scope="document"|"dialog"
    dtmf="dtmf_sequence"
    accept="exact"|"approximate"
  Child Elements
</menu>
```

Description

Dialog that transitions to `<choice>` destinations based on user input.

Attribute	**Description**
`id`	Menu identifier. *Optional*. Lets you specify this menu as the target for a `<goto>` or `<submit>`.
`scope`	Sets the scope of the menu's grammar. *Optional* (default is `"dialog"`). • `document` - The menu's grammar is active throughout the current document. If the document is the application root document, then it is active throughout the application (application scope). • `dialog` - The menu's grammar is active only in the current menu.
`dtmf`	Enables DTMF selection for all choices in this menu. *Optional* (default is `"false"`). • `true` - For `<choice>` elements that do not explicitly specify a DTMF (touch tone) sequence using the `dtmf` attribute, the interpreter assigns DTMF selectors of `"1"`, `"2"`, `...` to those choices, in document order. • `false` - The interpreter does not make implicit DTMF assignments to menu choices with no DTMF sequences.
`accept`	*Extension.* Specifies whether the default grammars generated for `<choice>` elements require all words or accept a subset of the words. • `exact` - Requires the user to say the exact phrase that appears in the <choice> element. • `approximate` - Allows the user to say a subset of the words in the <choice> element. **Note:** For backward compatibility, the default is `"approximate"` if the `version` attribute of the containing `<vxml>` element is less than `2.0` or unspecified. The default is `"exact"` if the `version` attribute is `2.0` or greater.

Usage

Parents	Children

`<vxml>`	`<audio>`
	`<enumerate>`
	`<value>`
	`<choice>`
	`<catch>`
	`<help>`
	`<noinput>`
	`<nomatch>`
	`<error>`
	`<prompt>`
	`<property>`
	`<script>`

See Also

- VoiceXML 1.0 Specification:
  <menu>
- Related tags:
  "<choice>" on page 70
  "<enumerate>" on page 88

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">
  <menu id="mainMenu" >
    <prompt>
      This is main menu. Please choose from
      <enumerate/> to complete your order.
    </prompt>
    <choice next="#dateForm"> date </choice>
    <choice next="#quantityForm"> quantity </choice>
    <choice next="#phoneForm"> phone </choice>
  </menu>
  <form id="dateForm">
    <field name="date" type="date">
      <prompt> What date would you like to pick up your order?</prompt>
      <filled>
        <prompt>
          Just to confirm, I heard you saying <value expr="date"/>
          to pick up.
        </prompt>
      </filled>
    </field>
    <block>
      <goto next="#quantityForm" />
    </block>
  </form>
  <form id="quantityForm">
    <field name="quantity" type="number">
      <prompt> How many bags of candies would you like to order?</prompt>
      <filled>
        <prompt>
          Okay, <value expr="quantity"/>
          bags of candies would be ordered.
        </prompt>
      </filled>
    </field>
    <block>
      <goto next="#phoneForm" />
    </block>
  </form>
  <form id="phoneForm">
    <field name="phone" type="phone">
      <prompt>
        What number should I call to reach you when the order is ready?
      </prompt>
      <filled>
        <prompt> I guess <value expr="phone"/> is your home number.</prompt>
      </filled>
    </field>
  </form>
</vxml>
```

<meta>

Define a meta data item as a name/value pair.

Syntax

```
<meta
   name="string"
   content="string"
   http-equiv="string" />
```

Description

Attribute	**Description**
`name`	Name of the meta-data property. For general document information such as: author, copyright, description, keywords, etc. Set the `name` attribute to `"maintainer"` to have the trace log emailed to the user specified in the `content` attribute after the call.
`content`	Value of the meta-data property.
`http-equiv`	Name of an HTTP response header. *Optional* (as an alternative to `name`). For HTTP response headers such as `Expires` or `Date`.

Usage

Parents	**Children**
`<vxml>`	None.

See Also

- VoiceXML 1.0 Specification:
  <meta>

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">

  <!--Replace myName@myCompany.com  with your emailAddress in the meta tag line.-->

  <meta name="author" content="bevocal"/>
  <meta name="maintainer" content="myName@myCompany.com" />
  <form>
    <block>
      <prompt>
        Welcome to the BeVocal Cafe.  The client log is E mailed to the maintainer.
        Please check your email.
      </prompt>
    </block>
  </form>
</vxml>
```

<noinput>

Catch a no-input event.

Syntax

```
<noinput
    count="integer"
    cond="js_expression">
  Executable Content
</noinput>
```

Description

Attribute	Description
`count`	Minimum number of times an error must have occurred during a form or menu invocation. *Optional* (default is `"1"`).
`cond`	JavaScript expression that must also evaluate to `"true"` for an event to be caught. *Optional* (default is `"true"`).

If multiple handlers for no-input events are defined in, or inherited by, the element in which the events occurs, one handler is chosen based on event count, scope, and document order. See Chapter 3, "Event Handling".

Usage

Parents	Children

`<vxml>`
`<form>`
`<menu>`
`<field>`
`<initial>`
`<record>`
`<transfer>`
`<subdialog>`
`<object>`
	`<audio>`
	`<enumerate>`
	`<value>`
	`<assign>`
	`<clear>`
	`<disconnect>`
	`<exit>`
	`<goto>`
	`<if>`
	`<prompt>`
	`<reprompt>`
	`<return>`
	`<script>`
	`<submit>`
	`<throw>`
	`<var>`
	`<log>`
	`<send>`

See Also

- VoiceXML 1.0 Specification:
  <noinput>
- Related tags:
  "<catch>" on page 67
  "<error>" on page 90
  "<help>" on page 120

“<nomatch>” on page 138
“<rethrow>” on page 162
“<throw>” on page 183

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">
  <noinput>
    Sorry I did not hear what you said
    <reprompt/>
  </noinput>
  <form id="foo">
    <field name="name">
      <prompt> What is your favorite color ?  </prompt>
      <grammar> [ red green yellow blue orange ] </grammar>
      <filled>
        <prompt> Your favorite color is <value expr="name"/> </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

<nomatch>

Catch a no-match event.

Syntax

```
<nomatch
    count="integer"
    cond="js_expression">
  Executable Content
</nomatch>
```

Description

Attribute	Description
`count`	Minimum number of times an error must have occurred during a form or menu invocation. *Optional* (default is `"1"`).
`cond`	JavaScript expression that must also evaluate to `"true"` for an event to be caught. *Optional* (default is `"true"`).

If multiple handlers for no-match events are defined in, or inherited by, the element in which the events occurs, one handler is chosen based on event count, scope, and document order. See Chapter 3, “Event Handling”.

Usage

Parents	Children

`<vxml>`
`<form>`
`<menu>`
`<field>`
`<initial>`
`<record>`
`<transfer>`
`<subdialog>`
`<object>`
	`<audio>`
	`<enumerate>`
	`<value>`
	`<assign>`
	`<clear>`
	`<disconnect>`
	`<exit>`
	`<goto>`
	`<if>`
	`<prompt>`
	`<reprompt>`
	`<return>`
	`<script>`
	`<submit>`
	`<throw>`
	`<var>`
	`<log>`
	`<send>`

See Also

- VoiceXML 1.0 Specification:
  <nomatch>
- Related tags:
  “<catch>” on page 67
  “<error>” on page 90
  “<help>” on page 120

“<noinput>” on page 136
“<rethrow>” on page 162
“<throw>” on page 183

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">
  <nomatch>
    Please say one of red green yellow blue or orange
  </nomatch>
  <form id="foo">
    <field name="name">
      <prompt> What is your favorite color ? </prompt>
      <grammar> [ red green yellow blue orange ] </grammar>
      <filled>
        <prompt> Your favorite color is <value expr="name"/> </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

<object>

Interact with a custom extension.

Syntax

```
<object
    name="string"
    expr="js_expression"
    cond="js_expression"
    classid="speechobject//name.source"
    modal="true"|"false"
    data="URL"
    codebase="URL"
    type - not implemented
    codetype - not implemented
    archive="URL"
    caching="safe"|"fast"
    fetchhint="prefetch"|"safe"
    fetchtimeout="time_interval"
    fetchaudio="URL"
    maxage="time_interval"
    maxstale="time_interval" >
  Child Elements
</object>
```

Description

Field item that invokes reusable Speech Object components (supports Nuance SpeechObjects).

The field item variable for `<object>` contains a property corresponding to the specific Speech Object that was executed. The property name corresponds to the key in which that value is stored in the SpeechObject result class `KVSet`. See the SpeechObjects Reference for more details.

Attribute	**Description**
`name`	Field item variable used to store the results of the `<object>` invocation. The variable name may not be a JavaScript reserved keyword. The field item variable has dialog (form) scope; its name must be unique among all VoiceXML and JavaScript variables within the form's scope. The result is returned as a JavaScript object. You can access the return value elements using the following notation: *`name`*`.`*`outParam1`*, *`name`*`.`*`outParam2`*
`expr`	JavaScript expression that assigns the initial value of the field item variable. *Optional* (default is `"undefined"`). If you set the field item variable to a value other than `"undefined"`, you'll need to clear it before the `<object>` can execute.

Attribute	Description
`cond`	JavaScript boolean expression that also must evaluate to `"true"` for the `<object>` to be invoked. *Optional* (default is `"true"`). If not specified, the value of the field item variable alone determines whether or not the `<object>` can execute.
`classid`	URL of the object's implementation. Currently, the only supported objects are Nuance SpeechObjects. As a consequence, the URL must consist of `"speechobject://"` followed by the fully-qualified class name of the SpeechObject. For example: `"speechobject://nuance.so.SODate"` See SpeechObjects Reference for a list of BeVocal speech objects. If the object is not found, the interpreter throws an `error.unsupported.object` event.
`modal`	*Extension*. Boolean value that controls whether higher level grammars are active during execution of the Speech Object. • `"true"` - Interaction is modal; only the Speech Object's grammary are active while the object executes; higher level grammars are not active. • `"false"` - Higher level grammars are active while the object executes. *Optional* (default is `"true"`).
`data`	URL of the Java Archive (JAR) or Zip archive containing prompts used by this SpeechObject. The URL can be specify an absolute location or a location relative to the `codebase` path. *Optional*. Each prompt file in the archive will be unpacked into a temporary directory that is placed at the beginning of the SpeechObject's prompt path. Relative paths of files in the archive are preserved, but the SpeechObject must refer to the prompts using the complete paths. For example, if the archive file includes a file named `prompts/myprompts/hello.wav`, then the SpeechObject should refer to the prompt by exactly that path rather than just by `hello.wav`.
`codebase`	The base path used to resolve relative URLs specified by the `data` and `archive` attributes. *Optional (*defaults is the base URL of the current document).
`type`	*Not implemented.*
`codetype`	*Not implemented.*
`archive`	URL of the object's data; that is, the Java Archive (JAR) file containing the object's Java class files. The URL can be specify an absolute location or a location relative to the `codebase` path. The specified file is downloaded and cached according to the normal caching rules; then, the object specified by the `classid` attribute is loaded from the file.

Attribute	Description
`caching`	See Chapter 4, “Fetching Resources”. *Optional.*
`fetchhint`	See Chapter 4, “Fetching Resources”. *Optional.*
`fetchtimeout`	See Chapter 4, “Fetching Resources”. *Optional.*
`fetchaudio`	See Chapter 4, “Fetching Resources”. *Optional.*
`maxage`	*Extension*. See Chapter 4, “Fetching Resources”. *Optional.*
`maxstale`	*Extension*. See Chapter 4, “Fetching Resources”. *Optional.*

Usage

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

See Also

- VoiceXML 1.0 Specification:
  <object>
- SpeechObjects Reference
- Related tag:
  “<subdialog>” on page 177

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="form1">
    <block name="welcome">
      This is a test for nuance speech object s o date.
    </block>
    <object classid="speechobject://nuance.so.SODate" name="date">
      <filled>
        <prompt> The month is <value expr="date.Month"/> </prompt>
        <prompt> The day of month is <value expr="date.DayOfMonth"/> </prompt>
        <prompt> The day of week is <value expr="date.DayOfWeek"/> </prompt>
        <clear/>
      </filled>
    </object>
  </form>
</vxml>
```

<option>

Specify an option in a `<field>`.

Syntax

```
<option
    value="string"
    dtmf="dtmf_sequence">
  Option Text
</object>
```

Description

Provide one of a simple set of alternatives within a field without specifying a grammar. A grammar for the field is generated automatically, based on the option list. You can use `<enumerate>` to generate prompts automatically based on option lists as well.

Attribute	Description
`value`	String to assign to the field item variable when this item is selected. *Optional* (default is the value of the `dtmf` attribute, if any, otherwise, the option text itself with leading and trailing white space removed).
`dtmf`	DTMF sequence to assign to this option. *Optional.*

Usage

Parents	Children
`<field>`	None.

See Also

- VoiceXML 1.0 Specification:
  <option>
- Related tags:
  “<field>” on page 95
  “<choice>” on page 70
  “<enumerate>” on page 88

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="mainform">
    <block name="welcome">
      This is a test for option tag.
    </block>
    <catch event="nomatch noinput">
      <prompt> Sorry, I didn't understand. </prompt>
      <reprompt/>
    </catch>
    <field name="mainmenu">
      <prompt>
        Please select an application from the list of
        Driving Directions, Stock Quotes, Business Finder.
      </prompt>
      <option value="Directions">  Driving Directions </option>
      <option value="Portfolio">  Stock Quotes </option>
      <option value="Stores">  Business Finder </option>
      <filled>
        <prompt>You chose <value expr="mainmenu"/> </prompt>
      </filled>
    </field>
  </form>
</vxml>
```

<param>

Specify a parameter in `<object>` or `<subdialog>`.

Syntax

```
<param
   name="string"
   expr="js_expression"
   value="string"
   valuetype - not implemented
   type="MIME_type"
   index="integer"
  Nested Param Element
</param>
```

Description

Passes values to objects or subdialogs.

When a `<param>` element is used to pass a parameter to a subdialog, the subdialog must contain a `<var>` declaration for the parameter. If the `<var>` contains an initializing `expr` attribute, that initializing value is ignored and the value passed with the `<param>` element is used instead.

Attribute	**Description**
`name`	Parameter name.
`expr`	JavaScript expression that evaluates to the value of this parameter. *Optional* (as alternative to `value`).
`value`	String to assign as the value of this parameter. *Optional* (as alternative to `expr`).
`valuetype`	*Not implemented.*
`type`	Type of the `value` attribute (following Nuance precedent). *Optional*. If not specified and parent is a Speech Object, the type of the Speech Object parameter is used when assigning the value.
`index`	*Extension*. Index of current `<param>` element. *Optional*. Use when parent `<param>` element is a vector (`java.lang.Vector`) or an array.

Usage

Parents	**Children**

`<param>`
`<subdialog>`
`<object>`
	`<param>`

See Also

- VoiceXML 1.0 Specification:
  <param>

- Related tags:
  “<object>” on page 140
  “<subdialog>” on page 177

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="form1">
    <object classid="speechobject://nuance.so.SOBrowsableList" name="SOBrowsable1">
      <param name="browsable" type="bevocal.cafe.BVBrowsableVector">
        <param name="playableVector" type="java.util.Vector">
          <param index="0" type="vcommerce.util.prompt.TTSPrompt">
            <param name="text" expr="'apples'" />
          </param>
          <param index="1" type="vcommerce.util.prompt.TTSPrompt">
            <param name="text" expr="'pears'" />
          </param>
          <param index="2" type="vcommerce.util.prompt.TTSPrompt">
            <param name="text" expr="'oranges'" />
          </param>
        </param>
      </param>
    </object>
  </form>
</vxml>
```

<prompt>

Queue TTS and audio output to the user.

Syntax

```
<prompt
    bargein="true"|"false"
    cond="js_expression"
    count="integer"
    timeout="time_interval">
  Prompt Text
</prompt>
```

Description

Uses TTS to convey information to the user. If the prompt consists of simple text only, you can omit the prompt tags and the text will be interpreted as if they were present.

Attribute	Description
`bargein`	Determines whether user input will be recognized during the prompt. *Optional* (default is `"true"`).
`hotword`	*Extension.* Determines whether only speech that matches a grammar can interrupt the prompt. *Optional* (default is `"false"`). By default, any user utterance will interrupt the prompt. Only applies when `bargein` is `"true"`.
`cond`	JavaScript boolean expression that must evaluate to `"true"` for the prompt to be spoken.
`count`	Minimum number of times the user must have visited the form item containing the prompt for the prompt to be spoken. *Optional* (default is `"1"`). Lets you vary prompts if the user is having problems and revisits the form same item several times. Form item prompt counters are reset with each invocation of the form.
`timeout`	Time to wait before throwing a no-input event. *Optional* (default value is `"5s"` and default unit is seconds). Express time interval as an unsigned number followed by `"s"` for time in seconds; `"ms"` for time in milliseconds.

Tips:

- Set the `hotword` attribute to `"true"` if you have a long audio segment that you only want interrupted by in-grammar utterances.
- You can use the `hotword` property to enable hotword behavior by default. See "<property>" on page 151. As with `bargein`, the value of the `hotword` attribute

takes precedence over scoped occurrences of the `hotword` property. For example:

```
<form>
  <property name="hotword" value="true"/>
  <field name="first">
    <prompt> This prompt will have hotword
      enabled.
    </prompt>
  </field>
  <field name="second">
    <property name="hotword" value="false"/>
    <prompt> This prompt will not have hotword
      enabled.
    </prompt>
    <prompt hotword="true"/> But this one will.
    </prompt>
  </field>
</form>
```

Usage

Parents	Children

`<menu>`
`<field>`
`<initial>`
`<record>`
`<transfer>`
`<subdialog>`
`<object>`
`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	`<audio>`
	`<enumerate>`
	`<value>`
	`<break>`
	`<div>`
	`<emp>`
	`<pros>`
	`<sayas>`
	`<say-as>`

See Also

- VoiceXML 1.0 Specification:
  <prompt>
- Related tags:
  "<reprompt>" on page 160
  "<audio>" on page 58

Examples

Example 1:

```
<?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="tapered">
    <block>
      <prompt bargein="false">
        This is question # 1.
      </prompt>
    </block>
    <field name="color">
      <noinput> <reprompt/> </noinput>
      <nomatch> <reprompt/> </nomatch>
      <grammar>[blue red green yellow]</grammar>
      <prompt count="1">What is the color of the Sky?</prompt>
      <prompt count="2">Choose a color.</prompt>
      <prompt count="3">Choose from red blue green or yellow.</prompt>
      <help>
        The color of the sky is usually blue except
        during sunset and sunrise.
      </help>
      <filled>
        <if cond="color==’blue’">
          <prompt>Thats correct. The sky is <value expr="color"/> </prompt>
        <else/>
          <prompt> thats not correct. The sky is blue in color. </prompt>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

Example 2:

```
<?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="hear_again">
    <var name="r" expr="5"/>
    <field name="more" type="boolean">
      <prompt>The value of r is <value expr="r"/></prompt>
      <prompt cond="r &lt; 6">
        Do you want me to play another music?
      </prompt>
      <prompt cond="r >= .50">
        For another music say yes.  To exit say no.
      </prompt>
      <filled>
        <prompt>The value of r is <value expr="r"/></prompt>
        <if cond="more">
          <audio src="chimes.wav"/>
        </if>
        <clear/>
      </filled>
    </field>
  </form>
</vxml>
```

<property>

Control implementation platform settings.

Syntax

```
<param
    name="string"
    value="string"/>
```

Description

Sets values that affect platform behavior and/or represent default attribute values.

Attribute	Description
`name`	Property name. The name can be any of the supported properties described in Chapter 7, “Property Reference”.
`value`	Property value. The allowable values depend on the property specified in the `name` attribute.

The property settings apply to the parent element and all descendents. However, property values set at lower levels take precedence.

The value of time-related properties must be an unsigned number, optionally followed by `"s"` for seconds or `"ms"` for milliseconds. If there is no suffix, seconds are assumed.

Usage

Parents	Children

`<vxml>`
`<form>`
`<menu>`
`<field>`
`<initial>`
`<record>`
`<transfer>`
`<subdialog>`
`<object>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <property>

Examples

Example 1 - bargein property:

```
<?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">
  <property name="bargein" value="true"/>
  <form id="form_property">
    <property name="bargein" value="false"/>
    <block>
      <prompt>
        This is a prompt where you can not barge in
      </prompt>
      <goto next="#form2"/>
    </block>
  </form>
  <form id="form2">
    <field name="option" type="boolean">
      <prompt>
        Try bargein in by saying one of yes or no as I speak because
        this is a prompt where you can barge in.
      </prompt>
      <filled>
        <if cond="option">
          <disconnect/>
        <else/>
          <prompt> Continuing till this prompt </prompt>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

Example 2 - timing properties:

```
<?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>
    <!-- If you say nothing, it should pause for 10s then land here -->

    <property name="timeout" value="10s"/>
    <noinput>
      <prompt>no input</prompt>
      <reprompt/>
    </noinput>

    <!-- If you say "one" then stop, it should pause for 5s then land here -->

    <property name="incompletetimeout" value="5s"/>
    <nomatch>
      <prompt>no match.</prompt>
      <reprompt/>
    </nomatch>

    <!-- If you say "one two three four", it should land here immediately -->

    <property name="completetimeout" value="0.1s"/>
    <filled>
      <prompt>Filled.</prompt>
      <exit/>
    </filled>

    <field name="one">
      Say one two three four
      <grammar>(one two three four)</grammar>
    </field>

    <block>
      Finished
    </block>
  </form>
</vxml>
```

Example 3 - maximum error properties:

```
<?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">

<!--this tests the maxerrors and maxdialogerrors properties. -->
```

```
<property name="bevocal.maxerrors" value="9"/>
<property name="bevocal.maxdialogerrors" value="4" />
<property name="timeout" value="1.5s" />
<noinput> i did not hear anything </noinput>

<form id="firstform">
  <catch event="error.bevocal.maxdialogerrors_exceeded">
    <prompt>
      max dialog errors event detected.  going to next form.
    </prompt>
    <goto next="#secondform"/>
  </catch>
  <catch event="error.bevocal.maxerrors_exceeded">
    <prompt>
      total max errors event detected.  this should not happen.  exiting test.
    </prompt>
    <exit/>
  </catch>
  <field name="first" type="boolean">
    please do not say anything.  you should hear the no input message 3 times before
    moving to the next field.
  </field>
 </form>

<!-- When you reach the second form, 4 errors have occurred. -->
<form id="secondform">
  <property name="bevocal.maxdialogerrors" value="3" />
  <catch event="error.bevocal.maxdialogerrors_exceeded">
    <prompt>
      max dialog errors event detected.  going to next form.
    </prompt>
    <goto next="#thirdform"/>
  </catch>
  <catch event="error.bevocal.maxerrors_exceeded">
    <prompt>
      total max errors event detected.  this should not happen.  exiting test.
    </prompt>
    <exit/>
  </catch>
  <field name="first" type="boolean">
    please do not say anything.  you should hear the no input message 2 times before
    moving to the next field.
  </field>
</form>

<!-- When you reach the third form, 7 errors have occurred;
  After two more errors, you will exceed the maximum for the call. -->
<form id="thirdform">
  <catch event="error.bevocal.maxdialogerrors_exceeded">
    <prompt>
      max dialog errors event detected.  this should not happen.
    </prompt>
    <exit/>
  </catch>
  <catch event="error.bevocal.maxerrors_exceeded">
```

```
      <prompt>
        total max errors event detected.  the test was successful.  now exiting.
      </prompt>
      <exit/>
    </catch>
    <field name="first" type="boolean">
      please do not say anything.  you should hear the no input message 2 times before
      you get a max total errors event.
    </field>
  </form>
</vxml>
```

<pros>

Java Speech Markup Language (JSML) element to change the prosody of speech output.

Syntax

```
<pros
    rate="string"
    vol="string"
    pitch="string"
    range="string">
  Text
</pros>
```

Description

Controls how the enclosed text is spoken.

Currently all attributes are ignored and text is spoken normally.

Usage

Parents	Children

`<choice>`
`<prompt>`
`<enumerate>`
`<audio>`
`<div>`
`<emp>`
`<pros>`
	`<audio>`
	`<enumerate>`
	`<value>`
	`<break>`
	`<div>`
	`<emp>`
	`<pros>`
	`<sayas>`
	`<say-as>`

See Also

- VoiceXML 1.0 Specification:
  <pros>
- Related Elements:
  “<break>” on page 65
  “<div>” on page 80
  “<emp>” on page 87
  “<sayas>” on page 168

<record>

Record an audio sample.

Syntax

```
<record
   name="string"
   expr="js_expression"
   cond="js_expression"
   maxtime="time_interval"
   finalsilence="time_interval"
   type - not implemented
   beep - not implemented
   modal - not implemented
   dtmfterm - not implemented >
  Child Elements
</record>
```

Description

Field item that collects a recording from the user and stores the result in the field item variable.

Attribute	Description
`name`	Field item variable used to store the recording. The variable name may not be a JavaScript reserved keyword. The field item variable has dialog (form) scope; its name must be unique among all VoiceXML and JavaScript variables within the form's scope.
`expr`	JavaScript expression that assigns the initial value of the field item variable. *Optional* (default is `"undefined"`). If you set the field item variable to a value other than `"undefined",` you'll need to clear it before the `<record>` can execute.
`cond`	JavaScript boolean expression that also must evaluate to `"true"` for the `<record>` to execute. *Optional* (default is `"true"`). If not specified, the value of the field item variable alone determines whether or not the `<record>` can execute.
`maxtime`	Maximum duration of the recording. *Optional* (default is `"10s"`). Express time interval as an unsigned number followed by `"s"` for time in seconds; `"ms"` for time in milliseconds (the default).
`finalsilence`	Duration of silence that will terminate the recording. *Optional* (default is `"1.5s"`). Express time interval as unsigned number followed by `"s"` for time in seconds; `"ms"` for time in milliseconds (the default).
`type`	*Not implemented (*default is 8 KHz Wav).
`beep`	*Not implemented.*

Attribute	Description
`modal`	*Not implemented.*
`dtmfterm`	*Not implemented.*

Corresponding to the field item variable *`name`* is a “shadow variable” called *`name$`*. After the recording is made, additional information is available in the following properties of this shadow variable:

- `duration` - The duration of the recording in milliseconds.
- `size` - The size of the recording in bytes.
- `termchar` - *Not implemented.* If the `dtmfterm` attribute is `"true"`, and the user terminates the recording by pressing a DTMF key, then this property is the key pressed (for example, `"#"`). Otherwise, the variable is null.

For a `<record>` element whose name is *`name`*, you access the property *`propName`* of the shadow variable with the syntax:

```
name$.property
```

For example, you access the `duration` property for the `<record>` named `greeting` as:

```
greeting$.duration
```

Usage

Parents	Children

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

See Also

- VoiceXML 1.0 Specification:
  <record>

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="form-record">
    <record name="greeting" beep="true" maxtime="10s" finalsilence="2000ms">
      <prompt>
        Please record your greeting after the tone.
      </prompt>
      <noinput>
        i did not hear anything.  <reprompt/>
      </noinput>
    </record>
    <field name="confirm" type="boolean">
      <prompt>
        your greeting is <value expr="greeting"/>
        to keep it say yes, to discard it say no.
      </prompt>
      <filled>
        <if cond="confirm">
          <prompt> ok, i will save your greeting </prompt>
          <submit method="post" namelist="greeting"
            next="greetingstore.jsp"/>
        <else/>
          <prompt> ok, lets try again </prompt>
          <clear namelist="greeting confirm"/>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

<reprompt>

Play a field prompt when a field is re-visited after an event.

Syntax

```
<reprompt/>
```

Description

Normally the interpreter suppresses prompts when it selects the next form item after executing a `<catch>`. By placing a `<reprompt>` in the `<catch>` you can cause normal prompting to occur and prompt counters to increment when the next form item is executed.

Usage

Parents	Children

`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <reprompt>
- Related tag:
  "<prompt>" on page 147

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">

  <!--Do not give any input. The reprompt is played for the noinput-->

  <form id="form-record">
    <record name="message" beep="true" maxtime="10s"
      finalsilence="2000ms" dtmfterm="true" type="audio/wav">
      <prompt>
        Please record your message after the tone and press any
        key after you are done.
      </prompt>
      <noinput>
        Error: i did not quite get your message.  <reprompt/>
      </noinput>
    </record>
    <block>
      <prompt>
        Your message is <value expr="message"/>
      </prompt>
    </block>
  </form>
</vxml>
```

<rethrow>

When executed within an event handlers, causes the event currently being handled to be rethrown. The execution environment searches for a new handler for the event starting in the scope above the one containing the current handler.

Syntax

```
<rethrow />
```

Description

Extension: Used within an event handler, causes the event currently being handled to be thrown again, in the scope above that handler. Rethrowing an event allows you to handle the same event at different levels. For example, an event handler in a form could perform a certain amount of cleanup and then rethrow the event so that a document-level event handler could perform further cleanup. For more information, see Chapter 3, “Event Handling”.

Usage

Parents	Children
`<block>` `<catch>` `<error>` `<help>` `<noinput>` `<nomatch>` `<if>` `<filled>`	None.

See Also

- Related tags:
  “<catch>” on page 67
  “<error>” on page 90
  “<help>” on page 120
  “<noinput>” on page 136
  “<nomatch>” on page 138
  “<throw>” on page 183

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">
  <catch event="myEvent">
    <prompt>Catch of my event at document scope.</prompt>
    <reprompt/>
  </catch>
  <form id="numbers">
    <catch event="myEvent">
      <prompt>Catch of my event at form scope.</prompt>
      <rethrow/>
    </catch>
    <block name="numbergame"> This is a test for throw tag. </block>
    <field name="mynumber" type="number">
      <prompt>
        Tell me a number greater than ten.
      </prompt>
      <filled>
        <prompt>
          The number you said is <value expr="mynumber"/>
        </prompt>
        <if cond="mynumber &lt; 10">
          <throw event="myEvent"/>
        </if>
      </filled>
    </field>
  </form>
 </vxml>
```

<return>

Return from a subdialog.

Syntax

```
<return
   namelist="variable1 ...">
   event="event"/>
```

Description

Causes subdialog's execution context to terminate. You can use the `<return>` to pass back variable values from the subdialog's execution context and to propagate an event back to the calling dialog (for example, a no-match event).

To propagate an event back to the calling dialog, use the `event` attribute. For example:

```
<return event="notmatch"/>
```

It is also possible to return a value *and* propagate an event. For example:

```
<return namelist="a" event="notmatch"/>.
```

This will store the return value in the subdialog's field item variable and then propagate the event.

If `<return>` is encountered in a dialog that is not executing as a subdialog, an `error.semantic` event is thrown.

Attribute	**Description**
`namelist`	Space separated list of variables to return to the calling dialog. *Optional* (default is to return no variables). This attribute can specify both VoiceXML variables and JavaScript variables, including variables that have not been explicitly declared.
`event`	Return to calling dialog and throw this event.

The values returned with the `namelist` attribute are available to the calling dialog as properties of the `<subdialog>` field item variable. They can be accessed using the following notation:

```
subdialogName.namelistVarible
```

For example, if a subdialog is invoked with:

```
<subdialog name="sub" .../>
```

and exited with:

```
<return namelist="a"/>
```

then the return value can be accessed as:

```
"sub.a"
```

Usage

Parents	Children

`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <return>
- Related tag:
  “<subdialog>” on page 177

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="main">
    <subdialog name="result" src="#subbie">
      <param name="goodbye" value="goodbye"/>
    </subdialog>

    <block>
      We’re back from the sub dialog.
      Result dot hello equals <value expr="result.hello"/>
      Result dot goodbye equals <value expr="result.goodbye"/>
    </block>
  </form>
  <form id="subbie">  <!-- This is the subdialog -->

    <var name="hello" expr="’hello’"/>  <!-- Already has a value,
                                             so can’t be initialized by caller -->
    <var name="goodbye"/>               <!-- Is initialized by caller -->
    <block>
      This is the sub dialog.
      <return namelist="hello goodbye"/>
    </block>
  </form>
</vxml>
```

<say-as>

W3C Speech Synthesis Markup Language element to modify how the enclosed word or phrase is spoken.

Syntax

```
<say-as
    sub="string"
    type="typeName"|"typeName:format" >
  Text
</say-as>
```

Description

This tag is an extension to the BeVocal VoiceXML 1.0 platform. This tag replaces the VoiceXML 1.0 tag `<sayas>`.

Attribute	Description
`sub`	Substitute text to be spoken instead of enclosed text. *Optional*.
`type`	Speak enclosed text in the given style. *Optional*. Possible type names and formats are: • *Not implemented:* `acronym` • `number` - For number, such as `10.5` or `10` • `number:digits` - For sequence of digits, such as: `123456` • `date` - For dates, such as `20001210` • *Not implemented:* `time` • *Not implemented:* `duration` • `currency` - For currency in dollars and cents, such as `$123.45` • *Not implemented:* `measure` • `telephone` - For telephone number adhering to the North American Dialing Plan, such as `408-907-3200` • *Not implemented:* `name` • *Not implemented:* `net` • `address` - For street, city, state, and zip code separated by commas, such as `1380 bordeaux drive, sunnyvale, california, 94089`

Usage

Parents	Children

`<choice>`
`<prompt>`
`<enumerate>`
`<audio>`
`<div>`
`<emp>`
`<pros>`
	None.

See Also

- Related tags:
  “<break>” on page 65
  “<div>” on page 80

“<emp>” on page 87
“<pros>” on page 156

<sayas>

Java Speech Markup Language (JSML) element to modify how a word or phrase is spoken.

Syntax

```
<sayas
   sub="string"
   class="phone"|"date"|"digits"|"literal"|
        "currency"|"number"|"airport"|"airline"|
        "equity"|"street"|"city"|"state"|
        "citystate"|"address"
   type="phone"|"date"|"digits"|"literal"|
        "currency"|"number"|"airport"|"airline"|
        "equity"|"street"|"city"|"state"|
        "citystate"|"address"
   phon - not implemented>
  Text
</sayas>
```

Description

The `<say-as>` tag replaces the `<sayas>` tag. The `<say-as>` tag is from the Synchronized Speech Markup Language.

Attribute	Description
`sub`	Substitute text to be spoken instead of enclosed text. *Optional*.

Attribute	Description
`class` `type`	Speak enclosed text in the given style. *Optional*. Possible values with enclosed text formats are: • `phone` - For telephone number adhering to the North American Dialing Plan, such as `408-907-3200` • `date` - For dates, such as `20001210` • `digits` - For digits, such as: `123456` • `literal` - For literals • `currency` - For currency in dollars and cents, such as `$123.45` • `number` - For numbers, such as `10.5` or `10` • *Not implemented:* `time` *Extensions:* • `airport` - For airport codes, such as `DFW` • `airline` - For airline codes, such as `AA` • `equity` - For company symbol or full name, such as `ibm` or `cisco systems` • `street` - For street name (with or without street number), such as `bordeaux drive` or `1380 bordeaux drive` • `city` - For city name, such as `sunnyvale` • `state` - For state name, such as `california` • `citystate` - For city name and state name separated by a comma, such as `sunnyvale, california` • `address` - For street, city, state, and zip code separated by commas, such as `1380 bordeaux drive, sunnyvale, california, 94089` **Note:** The `class` attribute is the VoiceXML 1.0 standard; the `type` attribute is an extension. These two attributes are identical; only one of them should be used.
`phon`	*Not implemented*. Representation of the Unicode International Phonetic Alphabet (IPA) characters to be spoken instead of enclosed text. *Optional*.

Usage

Parents	Children

`<choice>`
`<prompt>`
`<enumerate>`
`<audio>`
`<div>`
`<emp>`
`<pros>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <sayas>
- Related tags:
  "<break>" on page 65
  "<div>" on page 80
  "<emp>" on page 87
  "<pros>" on page 156

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>
    <block>
      <prompt>
        Here is a currency amount <sayas class="currency"> $123.45 </sayas>

        Here is a company <sayas class="equity"> ibm </sayas>

        Here is a company with a full name
        <sayas class="equity"> cisco systems</sayas>

        Here is a number <sayas class="number"> 10.05 </sayas>

        Here is an integer number.  You should not hear any decimal content
        <sayas class="number"> 10 </sayas>

        Here is an airport <sayas class="airport"> DFW </sayas>

        Here is an airline <sayas class="airline"> AA </sayas>

        Here is a phone number : <sayas class="phone"> 512-301-0691 </sayas>

        Here is a date : <sayas class="date"> 20001210 </sayas>

        Here is a street : <sayas class="street"> heiden lane </sayas>

        Here is a city : <sayas class="city"> austin </sayas>

        Here is a state: <sayas class="state"> texas </sayas>

        Here is a citystate <sayas class="citystate"> austin, texas </sayas>

        Here is an address
        <sayas class="address"> 9008 heiden lane, austin, texas, 78749 </sayas>

        Here is a digit string <sayas class="digits"> 123456 </sayas>
      </prompt>
    </block>
  </form>
</vxml>
```

<script>

Specify a block of JavaScript client-side scripting logic.

Syntax

```
<script
    src="URL"
    charset="string"
    caching="safe"|"fast"
    fetchhint="prefetch"|"safe"
    fetchtimeout="time_interval"
    maxage="time_interval"
    maxstale="time_interval" >
  Script Text
</script>
```

Description

Scripts do not have their own scope, but are executed in the scope of the containing element.

Attribute	Description
`src`	URL of the script. *Optional* (as alternative to inline).
`charset`	Character encoding of the script if `src` is used.
`caching`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchhint`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchtimeout`	See Chapter 4, "Fetching Resources". *Optional.*
`maxage`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*
`maxstale`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*

Tips:

- Put CDATA escapes around your scripts, so you don't have to follow all the XML rules for <, >, &, etc. For example,

```
<script>
<![CDATA[
  function factorial(n)
  {
    return (n <= 1) ? 1 : n * factorial(n-1);
  }
]]>
</script>
```

- Remember that any variables or functions declared in a script are valid only within the scope that contains the `<script>` element. Define the script in document scope (in the `<vxml>` element) if it defines items that you want to use in several dialogs or blocks.

Any items defined by a script inside a block are accessible only within that block. In the following example, the first block contains a script that defines the function `foo`. That function can be used in a JavaScript expression later in the same block. However, it is illegal to use the function in a different block. The `<value>` tag in the second block will fail because the function `foo` goes out of scope when the interpreter leaves the first block.

```
<block>
  <script>
    <!{CDATA[
      function foo() { return 1; }
    ]]>
  </script>
  <!-- This use of foo() is legal -->
  Foo is <value expr="foo()"/>
</block>
<block>
  <!-- ERROR!!!! This use of foo() is illegal -->
  Foo is <value expr="foo()"/>
</block>
```

Usage

Parents	Children

`<vxml>`
`<form>`
`<menu>`
`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <script>
- JavaScript Quick Reference

Examples

Example 1 - script to get the current time:

```
<?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="form">
    <block name="time">  <!-- This block uses a java script to tell you the time-->

      <var name="hours"/>
      <var name="minutes"/>
      <var name="seconds"/>

      <script>
        var d=new Date();
        hours=d.getHours();
        minutes=d.getMinutes();
        seconds=d.getSeconds();
      </script>

      <prompt>
        Now the Time is  <value expr="hours"/> hours
        <value expr="minutes"/> minutes, and
        <value expr="seconds"/> Seconds.
      </prompt>
    </block>
  </form>
</vxml>
```

Example 2 script defining a factorial function:

```
<?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">

  <script>
    <![CDATA[
      function factorial(n) { return (n <= 1) ? 1 : n * factorial(n-1); }
         ]]>
  </script>
  <var name="max" expr="10"/>
  <form id="form-factorial">
    <field name="fact" slot="num">
      <prompt>
        Tell me a number and I’ll tell you its factorial.
      </prompt>
      <prompt count="2">
        Your number, please.
      </prompt>
      <prompt count="3">
        Number?
      </prompt>
      <grammar src="number.grammar#MyTop"/>
      <catch event="help">
        <prompt>
           Please say a number more then zero and less than <value expr="max"/>
        </prompt>
      </catch>
      <filled>
        <if cond="fact &lt; max">
          <prompt>
            <value expr="fact" mode = "tts"/> factorial is
            <value expr="factorial(fact)" mode="tts"/>
          </prompt>
        <else/>
          <prompt>
            Please choose a number less than <value expr="max"/>.
          </prompt>
          <clear namelist="fact" />
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

<send>

Experimental Extension. Submit values to a web server without transitioning to a new VoiceXML document.

Syntax

```
<send
   url="URL"
   expr="js_expression"
   method="get"|"post"
   enctype=MIME_type
   namelist="variable1 ..."
   fetchtimeout="time_interval"
   fetchaudio="URL"/>
```

Description

This tag submits the specified data to a web server; when the web server returns a successful HTTP result code, execution of the current document continues with the tag following the `<send>`. This tag is useful when you need to save data (for example, the result of a `<record>`) in a database but do not need to transfer to a new document.

The servlet or CGI script document to which `<send>` submits data must return a valid HTTP reply, including headers and at least one blank line indicating the end of the headers.

If the returned HTTP result code does not indicate success, an `error.badfetch` is thrown to signal a server error to the VoiceXML application. That application can catch the error and play an error message or take some other action to alert the user.

Note: BeVocal is providing the current implementation to give our developers the opportunity to use the tag and provide feedback, which we can pass on to the W3C. If `<send>` is standardized, the BeVocal implementation will change as necessary to match the VoiceXML standard. If such changes occur, we will attempt to maintain backwards compatibility with the current implementation.

Attribute	**Description**
`url`	URL to which to submit the values. *Optional* (as alternative to `expr`).
`expr`	JavaScript expression that evaluates to the URL to which to submit the values. *Optional* (as alternative to `next`).
`method`	The query request method. *Optional* (default is `"get"`).
`enctype`	MIME encoding of the submitted document. *Optional* (default is `application/x-www-form-urlencoded`). The supported types are: • `application/x-www-form-urlencoded` • `multipart/form-data` The type `multipart/form-data` is more efficient when submitting large amounts of binary data.

Attribute	Description
`namelist`	Space separated list of variables to submit. *Optional* (default is to submit all field item variables that have been given explicit names with the name attribute of `<field>`, `<object>`, `<record>`, `<transfer>`, or `<subdialog>`). This attribute can specify both VoiceXML variables and JavaScript variables, including variables that have not been explicitly declared.
`fetchtimeout`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchaudio`	See Chapter 4, "Fetching Resources". *Optional.*

Usage

Parents	Children

`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

- Related tags:
  "<submit>" on page 181

Example

```
<?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="form-record">
    <field name="name" type="boolean">
      <prompt>
        Say yes to send your data to Snoop Servlet or no to exit
      </prompt>
      <filled>
        <if cond="name">
          <send method="post" url="http://www.yoursite.com/SomeServlet"/>
          <prompt>Data sent successfully!</prompt>
        </if>
        <prompt> Good bye! </prompt>
        <exit/>
      </filled>
    </field>
  </form>
</vxml>
```

<subdialog>

Invoke another dialog as a subdialog of the current one.

Syntax

```
<subdialog
    name="string"
    expr="js_expression"
    cond="js_expression"
    src="URL"
    method="get"|"post"
    enctype=MIME_type
    namelist="variable1 ..."
    modal="true"|"false"
    caching="safe"|"fast"
    fetchhint="prefetch"|"safe"
    fetchtimeout="time_interval"
    fetchaudio="URL"
    maxage="time_interval"
    maxstale="time_interval" >
  Child Elements
</subdialog>
```

Description

Field item that invokes a subdialog, which is a reusable dialog that is specially coded to pass back values with a `<return>` element. When control returns from the subdialog, the returned values are available as properties of the `<subdialog>` field item variable.

You pass values into the subdialog by including `<param>` child elements. The subdialog must contain a `<var>` declaration for each parameter passed to it; if the `<var>` contains an initializing `expr` attribute, that initializing value is ignored and the value passed with the `<param>` element is used instead.

When a subdialog is invoked, it runs in a new execution context. This means that all variables and state information in the subdialog's document are reinitialized (including the application root document, if one is used). Any changes that the subdialog makes to application-scoped variables apply only in the subdialog context. When the subdialog returns, its context is destroyed and the context of the calling dialog is in the same state it was in before the subdialog call. The *only* way for the subdialog to return information to its calling dialog is with a `<return>` element.

If the subdialog invokes other dialogs, those dialogs are also run in the new execution context. The new context terminates and the old context resumes only when the subdialog or another dialog it has invoked calls `<return>` to pass back the results. After the subdialog returns, execution proceeds to any applicable `<filled>` elements in the calling context.

Attribute	**Description**
`name`	Field item variable used to store the results of the subdialog. The field item variable has dialog (form) scope; its name must be unique among all VoiceXML and JavaScript variables within the form's scope. You can access the return values as properties of the field item variable using the syntax: *`name.returnVariableName`*
`expr`	JavaScript expression that assigns the initial value of the field item variable. *Optional* (default is `"undefined"`). If you set the field item variable to a value other than `"undefined",` you'll need to clear it before the `<subdialog>` can execute.
`cond`	JavaScript boolean expression that also must evaluate to `"true"` for the `<subdialog>` to execute. *Optional* (default is `"true"`). If not specified, the value of the field item variable alone determines whether or not the `<subdialog>` can execute.
`src`	URL of the subdialog. You can use the "*`#DialogName`*" syntax to refer to another dialog in the current document. Even in this case, the subdialog is run in a new context. Note that the `method`, `enctype`, `namelist`, `caching`, `fetchhint`, `fetchtimeout`, and `fetchaudio` parameters apply only if `src` points to a different document (as opposed to starting with "#" to invoke a subdialog in the current document).
`method`	The query request method, either `"get"` or `"post"`. *Optional* (default is `"get"`).
`enctype`	MIME encoding of the submitted document. *Optional* (default is `application/x-www-form-urlencoded`). The supported types are: • `application/x-www-form-urlencode` • `multipart/form-data` The type `multipart/form-data` is more efficient when submitting large amounts of binary data.
`namelist`	Space separated list of variables to submit. *Optional* (default is to submit nothing). This attribute can specify both VoiceXML variables and JavaScript variables, including variables that have not been explicitly declared.

Attribute	Description
`modal`	Boolean value that must be `"false"` to enable `<link>` grammars from the calling context. *Optional* (default is `"true"`). Lets you alter default behavior so that `<link>` grammars from the calling context can be active while the subdialog executes. **Note:** in the next version of the VoiceXMLstandard, the `modal` attribute will be removed and all subdialogs will be modal. To ensure portability, your applications should avoid using this attribute.
`caching`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchhint`	See Chapter 4, "Fetching Resources". *Optional.* **Note:** If this element sends variables to the server with the `namelist` attribute, a `"prefetch"` value for the `fetchhint` attribute is ignored.
`fetchtimeout`	See Chapter 4, "Fetching Resources". *Optional.*
`fetchaudio`	See Chapter 4, "Fetching Resources". *Optional.*
`maxage`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*
`maxstale`	*Extension*. See Chapter 4, "Fetching Resources". *Optional.*

By default, no grammars from the calling dialog's context are active, except any default grammars defined by the VoiceXML interpreter. However, if the `modal` attribute is `"false"`, `<link>` elements in the calling dialog's context are active. When a grammar from the calling context is triggered during execution of a subdialog, the subdialog context terminates and control returns to the calling context.

If an event is thrown during execution of a subdialog and no event handler for the event is found in the subdialog context, the interpreter's response depends on whether the subdialog is modal.

- If the subdialog is modal, a fatal error occurs, causing the interpreter to exit.
- If the subdialog is non-modal, the interpreter causes the subdialog's context to return. It then rethrows the event in the calling context and starts its search for the event handler in that context.

Usage

Parents	Children

`<form>`	`<audio>`
	`<value>`
	`<catch>`
	`<help>`
	`<noinput>`
	`<nomatch>`
	`<error>`
	`<filled>`
	`<param>`
	`<prompt>`
	`<property>`

See Also

- VoiceXML 1.0 Specification:
  <subdialog>
- Related tags:
  "<param>" on page 145
  "<return>" on page 164
  "<object>" on page 140

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="main">
    <block>We're about to call the sub dialog</block>
    <subdialog name="result" src="#subbie">
      <param name="hello" value="goodbye"/>
      <param name="goodbye" value="goodbye"/>
    </subdialog>
    <block>
      We're back from the sub dialog.
      Result dot hello equals <value expr="result.hello"/>
      Result dot goodbye equals <value expr="result.goodbye"/>
    </block>
  </form>

  <form id="subbie">  <!-- This is the subdialog -->
    <!-- Variables are given values by parameters to the subdialog -->
    <var name="hello" expr="'hello'"/>  <!-- Initial value will be ignored -->
    <var name="goodbye"/>
    <block>
      This is the sub dialog.
      <return namelist="hello goodbye"/>
    </block>
  </form>
</vxml>
```

<submit>

Submit values to a document server.

Syntax

```
<submit
   next="URL"
   expr="js_expression"
   method="get"|"post"
   enctype=MIME_type
   namelist="variable1 ..."
   caching="safe"|"fast"
   fetchtimeout="time_interval"
   fetchaudio="URL"
   maxage="time_interval"
   maxstale="time_interval" />
```

Description

Attribute	**Description**
`next`	URL to which to submit the values. *Optional* (as alternative to `expr`).
`expr`	JavaScript expression that evaluates to the URL to which to submit the values. *Optional* (as alternative to `next`).
`method`	The query request method. *Optional* (default is `"get"`).

`enctype`	MIME encoding of the submitted document. *Optional* (default is `application/x-www-form-urlencoded`).
	The supported types are:
	`application/x-www-form-urlencoded`
	`multipart/form-data`
	The type `multipart/form-data` is more efficient when submitting large amounts of binary data.

`namelist`	Space separated list of variables to submit. *Optional* (default is to submit all field item variables that have been given explicit names with the name attribute of `<field>`, `<object>`, `<record>`, `<transfer>`, or `<subdialog>`). This attribute can specify both VoiceXML variables and JavaScript variables, including variables that have not been explicitly declared.
`caching`	See Chapter 4, “Fetching Resources”. *Optional.*
`fetchtimeout`	See Chapter 4, “Fetching Resources”. *Optional.*
`fetchaudio`	See Chapter 4, “Fetching Resources”. *Optional.*
`maxage`	*Extension*. See Chapter 4, “Fetching Resources”. *Optional.*
`maxstale`	*Extension*. See Chapter 4, “Fetching Resources”. *Optional.*

Usage

Parents	Children
`<block>` `<catch>` `<error>` `<help>` `<noinput>` `<nomatch>` `<if>` `<filled>`	None.

See Also

- VoiceXML 1.0 Specification:
  <submit>
- Related tags:
  “<goto>” on page 112
  “<send>” on page 175

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="form-record">
    <field name="name" type="boolean">
      <prompt>
        Say yes or no to submit a request to Snoop Servlet
      </prompt>
      <filled>
        <if cond="name">
          <submit caching="safe" method="post"
            next="http://www.yoursite.com:8080/docroot/SomeServlet"/>
        <else/>
          <prompt> Good bye! </prompt>
          <exit/>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

<throw>

Throw an event.

Syntax

```
<throw
    event="event"
    eventexpr="js_expression"
    message="string"
    messageexpr="js_expression" />
```

Description

You can throw either a predefined event or an application-specific event. For more information, see Chapter 3, "Event Handling".

Attribute	Description
`event`	Event to throw. *Optional* (as alternative to `eventexpr`).
`eventexpr`	JavaScript expression that evaluates to the event to throw. *Optional* (as alternative to `event`).
`message`	Message string providing additional context about the event being thrown. *Optional*.
`eventexpr`	JavaScript expression that evaluates to the message string. *Optional*.

One and only one of the attributes `event` or `eventexpr` must be specified. At most one of the attributes `message` or `messageexpr` may be specified.

Within an event handler that catches the thrown event, the variable

Usage

Parents	Children

`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <throw>
- Related tags:
  "<catch>" on page 67
  "<error>" on page 90
  "<help>" on page 120
  "<noinput>" on page 136
  "<nomatch>" on page 138
  "<rethrow>" on page 162

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">
  <catch event="myEvent">
    <prompt>
      Catch at document scope in  myEvent.
      <value expr="_message">
    </prompt>
    <reprompt/>
  </catch>

  <catch event="nomatch noinput" >
    <prompt>
      Catch at document scope
    </prompt>
    <reprompt/>
  </catch>

  <help>
    <prompt>
      Catch for help at document scope
    </prompt>
    <reprompt/>
  </help>

  <form id="numbers">
    <block name="numbergame"> This is a test for throw tag. </block>
    <field name="mynumber" type="number">
      <prompt>
        Tell me a number greater than ten.
      </prompt>
      <filled>
        <if cond="mynumber &lt; 10">
          <throw
            event="myEvent"
            messageexpr="mynumber + ’ is less than ten’"/>
        <elseif cond="mynumber &gt; 10" />
          <prompt>
            The number you said is <value expr="mynumber"/>
          </prompt>
        <else/>
          <throw event="myEvent" message="The number was ten"/>
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

<transfer>

Transfer the caller to another destination.

Syntax

```
<transfer
    name="string"
    expr="js_expression"
    cond="js_expression"
    dest="URL"
    destexpr="js_expression"
    connecttimeout="time_interval"
    maxtime="time_interval"
    bridge - not implemented >
  Child Elements
</transfer>
```

Description

Field item for transferring the caller to another phone number. The transfer may be done so that the current session with the interpreter resumes after the call with the third party completes (bridging) or so that the current session terminates when the transfer is made (blind transfer).

Attribute	**Description**
`name`	Field item variable used to store the result of the transfer. The variable name may not be a JavaScript reserved keyword. The field item variable has dialog (form) scope; its name must be unique among all VoiceXML and JavaScript variables within the form's scope. Possible values are: • `busy` - The call was refused by the endpoint. • `noanswer` - There was no answer within the duration specified by `connecttimeout`. • `network_busy` - The call was refused by an intermediate network. • `near_end_disconnect` - The call completed because the caller hung up. • `far_end_disconnect` - The call completed because the callee hung up. • `network_disconnect` - The call completed and was terminated by the network.
`expr`	JavaScript expression that assigns the initial value of the field item variable. *Optional* (default is `"undefined"`). If you set the field item variable to a value other than `"undefined"`, you'll need to clear it before the `<transfer>` can execute.

Attribute	**Description**
`cond`	JavaScript boolean expression that also must evaluate to `"true"` for the `<transfer>` to execute. *Optional* (default is `"true"`). If not specified, the value of the field item variable alone determines whether or not the `<transfer>` can execute.
`dest`	URL of the destination (for example, phone, IP telephony address). *Optional* (as alternative to `destexpr`). You can specify the URL using any of the following formats: • `phone://8005551212` • `800-555-1212` • `phone://800-555-1212` A leading "1" on the phone number is optional and will be ignored.
`destexpr`	JavaScript expression that evaluates to the URL of the destination. *Optional* (as alternative to `dest`).
`connecttimeout`	Time to wait before returning a value of `"noanswer"`. Express time interval as an unsigned number followed by `"s"` for time in seconds; `"ms"` for time in milliseconds (the default).
`maxtime`	How long the call is allowed to last. *Optional* (default is `"0"`, signifying no limit). This only pertains to bridged calls. Express time interval as an unsigned number followed by `"s"` for time in seconds; `"ms"` for time in milliseconds (the default).
`bridge`	*Not implemented* (currently always `"true"`). Determines whether the current session will resume after the transferred call completed. *Optional* (default is `"true"`). If you set this to `"false"`, the current session terminates by throwing a `telephone.disconnect.transfer` event when the transfer is made.

Corresponding to the field item variable *`name`* is a "shadow variable" whose name is *`name$`*. After the transfer is complete, the duration of the call in seconds is stored in the floating-point `duration` property of this shadow variable. For a `<transfer>` element whose name is *`name`*, you access the duration with the syntax:

```
name$.duration
```

The following events may be thrown during the execution of a `<transfer>` element:

- `telephone.disconnect.hangup` - The caller hung up.
- `telephone.disconnect.transfer` - The caller was transferred unconditionally to another line and will not return.

Usage

Parents	Children

`<form>`	`<audio>`
	`<value>`
	`<catch>`
	`<help>`
	`<noinput>`
	`<nomatch>`
	`<error>`
	`<dtmf>`
	`<filled>`
	`<grammar>`
	`<prompt>`

See Also

- VoiceXML 1.0 Specification:
  <transfer>
- Related tag:
  “<disconnect>” on page 78

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="foo">
    <block>
      <prompt>
        Welcome to Be Vocal Cafe! Now taking you to
        BeVocal Consumer Services.
      </prompt>
      <var name="transname"/>
      <assign name="transname" expr="’phone://18004286225’"/>
    </block>
    <transfer name="services" bridge="true"
      connecttimeout="300" dest="phone://18004286225" />
  </form>
</vxml>
```

<value>

Insert the value of a expression in a prompt.

Syntax

```
<value
   class="phone"|"date"|"digits"|"literal"|
        "currency"|"number"|"airport"|"airline"|
        "equity"|"street"|"city"|"state"|
        "citystate"|"address"
   type="phone"|"date"|"digits"|"literal"|
        "currency"|"number"|"airport"|"airline"|
        "equity"|"street"|"city"|"state"|
        "citystate"|"address"
   expr="js_expression"
   mode - not implemented
   recsrc - not implemented />
```

Description

Attribute	**Description**
`class` `type`	The `<sayas>` type of the variable for interpretive purposes. *Optional*. See the `<sayas>` tag. **Note:** The `class` attribute is the VoiceXML 1.0 standard; the `type` attribute is an extension. These two attributes are identical; only one of them should be used.
`expr`	JavaScript expression to evaluate and insert in the prompt.
`mode`	*Not implemented.*
`recsrc`	*Not implemented.*

Usage

Parents	Children

`<menu>`
`<choice>`
`<prompt>`
`<enumerate>`
`<field>`
`<initial>`
`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<audio>`
`<div>`
`<emp>`
`<pros>`
`<record>`
`<transfer>`
`<if>`
`<filled>`
`<subdialog>`
`<object>`
`<log>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <value>
- Related tags:
  “<assign>” on page 56
  “<var>” on page 193
  “<field>” on page 95
  “<record>” on page 157
  “<object>” on page 140
  “<subdialog>” on page 177
  “<transfer>” on page 185
  “<block>” on page 63
  “<initial>” on page 124

Examples

Example 1 - with field item variable:

```
<?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>
    <field name="state">
      <prompt>
        Do you want texas or california?
      </prompt>
      <grammar> [ texas california ] </grammar>
    </field>
    <block>
      <prompt>
        Your answer was <value expr="state" class="state"/>
      </prompt>
    </block>
  </form>
</vxml>
```

Example 2 - using assignment:

```
<?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="myCalculator">
    <var name="result"/>
    <field name="op">   <!-- OPERATION    -->
      <prompt>
        BeVocal calculator.
        Choose add, subtract, multiply, or divide.
      </prompt>
      <grammar>
        [add subtract multiply divide]
      </grammar>
      <help>Say add, subtract, multiply, or divide.</help>
      <filled>
        <prompt>Okay, let’s <value expr="op"/> two numbers.</prompt>
      </filled>
    </field>

    <field name="a" type="number">   <!-- FIRST OPERAND -->
      <prompt>Whats the first number?</prompt>
      <help>
        Please say a number. This number will be used as the first operand.
      </help>
      <filled>
        <prompt><value expr="a"/></prompt>
      </filled>
    </field>

    <field name="b" type="number">   <!-- SECOND OPERAND -->
      <prompt>and the second number?</prompt>
      <help>
        Please say a number. This number will be used as the second operand.
      </help>
      <filled>
        <prompt><value expr="b"/> Okay.</prompt>

        <!-- NOW SAY THE RESULT -->

        <if cond="op==’add’">
          <!-- NOTE: Use Number() here in order to avoid
               interpreting + as string concatenation! -->
          <assign name="result" expr="Number(a) + Number(b)"/>
          <prompt>
            <value expr="a"/> plus <value expr="b"/> equals
            <value expr="result"/>
          </prompt>
        <elseif cond="op==’subtract’"/>
          <assign name="result" expr="a - b"/>
          <prompt>
            <value expr="a"/> minus <value expr="b"/> equals
            <value expr="result"/>
          </prompt>
        <elseif cond="op==’multiply’"/>
          <assign name="result" expr="a * b"/>
          <prompt>
            <value expr="a"/> times <value expr="b"/> equals
            <value expr="result"/>
          </prompt>
        <elseif cond="op==’divide’"/>
          <assign name="result" expr="a / b"/>
          <prompt>
            <value expr="a"/> divided by <value expr="b"/> equals
            <value expr="result"/>
          </prompt>
        </if>
        <clear/>
      </filled>
    </field>
  </form>
</vxml>
```

<var>

Declare a variable.

Syntax

```
<var
   name="string"
   expr="js_expression"/>
```

Description

Declaration of a variable in the scope of the enclosing element. If this variable is already declared in the current scope, further declarations are treated as value assignments.

Although variables do not need to be declared, explicit `<var>` declarations improve the readability and maintainability of an application, and aid application flow analysis.

Attribute	**Description**
`name`	Variable name, which must be a valid JavaScript identifier and may not be a reserved keyword in either JavaScript or Java.
`expr`	JavaScript expression that assigns the initial value of the variable. *Optional* (default is either `"undefined"` or the current value of the variable, if already declared in this scope).

If a `<var>` element names a variable that is already in scope, it does not declare a new variable with the same name, but simply assigns a value to the existing variable. If the `<var>` element has an `expr` attribute, the variable is assigned the specified value; otherwise, the variable is assigned the value `"undefined"`.

Usage

Parents	**Children**

`<form>`
`<vxml>`
`<block>`
`<catch>`
`<error>`
`<help>`
`<noinput>`
`<nomatch>`
`<if>`
`<filled>`
	None.

See Also

- VoiceXML 1.0 Specification:
  <var>
- Related tags:
  "<assign>" on page 56
  "<value>" on page 188

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="foo">
    <var name="a"/>
    <var name="b"/>
    <var name="result"/>
    <block>
      <assign name="a" expr="10"/>
      <assign name="b" expr="35"/>
      <assign name="result" expr="a * b"/>
    </block>
    <block>
      <prompt>
        <value expr="a"/> multiplied by <value expr="b"/> equals
        <value expr="result"/>
      </prompt>
    </block>
  </form>
</vxml>
```

<vxml>

Contain VoiceXML code.

Syntax

```
<vxml
    application="URL"
    base="URL"
    lang="language"
    version="version_number">
  Child Elements
</vxml>
```

Description

Top-level element in each VoiceXML document.

Attribute	Description
`application`	URL of this application's root document. *Optional* (default is not to have an application root document). If the specified document also has an `application` attribute, an `error.semantic` event is thrown.
`base`	Base URL. *Optional*.
`lang`	Language and locale for this document. *Optional*.
`version`	VoiceXML version used in this document. *Optional* (default is `"1.0")`.

Usage

Parents	Children

None.	`<catch>`
	`<help>`
	`<noinput>`
	`<nomatch>`
	`<error>`
	`<form>`
	`<link>`
	`<menu>`
	`<meta>`
	`<property>`
	`<script>`
	`<var>`

See Also

- VoiceXML 1.0 Specification:
  <vxml>

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="foo">
    <block>
      <prompt>
        Hello Developers! You are the VXML Gurus.
        Please keep using our services.
      </prompt>
    </block>
  </form>
</vxml>
```

7 Property Reference

This chapter describes the VoiceXML properties that can be set with the `<property>` tag.

Each property description explains what the property does, what possible values the property may have, and what default value is used if the application does not set the property.

In the cases where the BeVocal interpreter deviates from the VoiceXML 1.0 Specification, the difference is clearly marked below in the following ways:

- *Not Implemented* - Functionality not currently available.
- *Extension* - Added functionality.
- *Deprecated* - Non-standard or superseded feature that was supported by an earlier version but has been replaced by a new feature.

The property descriptions are organized according to the facet of the interpreter that the properties control. The following table list the properties in alphabetical order.

Property	**Controls**
`audiofetchhint`	Fetching
`audiomaxage` *Extension*	Fetching
`audiomaxstale` *Extension*	Fetching
`bargein`	Prompts
`bevocal.maxdialogerrors` *Extension*	Speech errors
`bevocal.maxerrors` *Extension*	Speech errors
`bevocal.goback` *Extension*	Go-back facility
`bevocal.mingoback` *Extension*	Go-back facility
`caching`	Fetching
`completetimeout`	Speech recognition
`confidencelevel`	Speech recognition
`datafetchhint` *Extension*	Fetching
`datamaxage` *Extension*	Fetching
`datamaxstale` *Extension*	Fetching
`documentfetchhint`	Fetching
`documentmaxage` *Extension*	Fetching
`documentmaxstale` *Extension*	Fetching
`fetchaudio`	Background Audio
`fetchaudiodelay` *Extension*	Background audio
`fetchaudiominimum` *Extension*	Background audio
`fetchtimeout`	Fetching

Property	Controls
`grammarfetchhint`	Fetching
`grammarmaxage` *Extension*	Fetching
`grammarmaxstale` *Extension*	Fetching
`hotword` *Extension*	Prompts
`incompletetimeout`	Speech recognition
`inputmodes`	Input modes
`interdigittimeout`	DTMF recognition
`objectfetchhint`	Fetching
`objectmaxage` *Extension*	Fetching
`objectmaxstale` *Extension*	Fetching
`scriptfetchhint`	Fetching
`scriptmaxage` *Extension*	Fetching
`scriptmaxstale` *Extension*	Fetching
`sensitivity`	Speech recognition
`speedvsaccuracy`	Speech recognition
`termchar`	DTMF recognition
`termtimeout` *Not Implemented*	DTMF recognition
`timeout`	Prompts
`universals` *Extension*	Universal grammars

Prompts

The following properties control the behavior of prompts:

- `bargein`
- `hotword`
- `timeout`

bargein

The `bargein` property indicates whether the user can “barge in” during prompts. Set this property to `"true"` to allow barge-in; set it to `"false"` to disallow barge-in.

The default value is `"true"`.

hotword

Extension. The `hotword` property indicates whether speech can interrupt the prompt even if it does not match a grammar. This property is relevant only when `bargein` is `"true"`.

Set this property to `"true"` so that input that doesn’t match the grammar is ignored and only speech that matches a grammar can interrupt the prompt. Set it to `"false"` to allow any user utterance to interrupt the prompt.

The default value is `"false"`.

timeout

The value of the `timeout` property is the time after which a no-input event is thrown by the interpreter.

The default value is 5 seconds.

See Appendix F of the VoiceXML 1.0 specification for a thorough description of the way this property interacts with the `completetimeout`, and `incompletetimeout` properties.

Input Modes

The `inputmodes` property controls allowable user input modes.

inputmodes

The `inputmodes` property specifies which input modes to enable: DTMF and/or voice.

The defaults value is `"dtmf voice"`. To disable speech recognition, set `inputmodes` to `"dtmf"`. To disable DTMF, set it to `"voice"`.

You can use this property:

- To turn off speech recognition in noisy environments.
- To conserve speech recognition resources by turning them off where the all input is expected to be DTMF.

Speech Recognition

The following properties control behavior of the speech-recognition engine:

- `completetimeout`
- `confidencelevel`
- `incompletetimeout`
- `sensitivity`
- `speedvsaccuracy`

See Appendix F of the VoiceXML 1.0 specification for a thorough description of the way the `timeout` property interacts with the `completetimeout` and `incompletetimeout` properties.

completetimeout

The value of the `completetimeout` property is the amount of time to wait for additional input after the engine has recognized speech matching one of the input grammars.

The default value is 0.5 seconds

confidencelevel

The value of the `confidencelevel` property is the confidence threshold required for the speech-recognition engine to decide whether the input speech matches a grammar.

The default value of 0.5 maps to 35% confidence in our Nuance speech-recognition engine. Setting this property higher will result in fewer false recognitions at the cost of many more no-match events. Setting it lower will reduce the number of no-match events at the risk of more false recognitions.

In practice, most applications use a `confidencelevel` very near the default value of 0.5. The minimum is 0.0 and the maximum is 1.0.

incompletetimeout

The value of the `incompletetimeout` property is amount of time to wait for additional speech input when the user has begun speaking but the input does not yet match a complete grammar.

The default is 1.5 seconds.

sensitivity

The value of the `sensitivity` property is the sensitivity of the speech recognition. In effect, this is the “gain” or “input volume” used for the speech input. Higher values will allow the speech-recognition engine to recognize softer speech but will also pick up more background noise. You may need to tune this property downward if you expect your application to be used in noisy environments, or upward if you expect it to be used in very quiet environments.

The default value is 0.5. The minimum is 0.0 and the maximum is 1.0.

speedvsaccuracy

The `speedvsaccuracy` property controls the trade-off between speed and accuracy in the speech-recognition engine. Lower values cause potential results with a low probability to be pruned early in the search process, resulting in faster recognition speed. Higher values retain potential results longer, resulting in slower but more accurate recognitions.

The default value of 0.5 maps to a fairly accurate setting (1200 on a scale of 0 to 1400 in our Nuance recognition engine). The minimum is 0.0 and the maximum is 1.0.

If you have extremely large grammars that are causing speech recognition to take too long, you might experiment with lower values for this property. Otherwise, you probably will not need to adjust this property.

DTMF Recognition

The following properties control DTMF recognition:

- `interdigittimeout`
- `termchar`
- `termtimeout`

interdigittimeout

The value of the `interdigittimeout` property is the amount of time that the recognition engine waits for another DTMF digit before it decides that the user has finished entering digits and returns a recognition or a no-match event.

The default is 3 seconds.

Note: Currently, this property also controls the amount of time the engine waits for a termination character (if `termchar` is set) after the last DTMF digit in a grammar is entered. When you have a fixed-length input field with a DTMF grammar, you may wish to set this property to 0 seconds to avoid a pause after the last digit is entered.

termchar

The value of the `termchar` property is the terminating character for DTMF recognition. If this property is set, the engine will wait interdigittimeout seconds for a the termination character before returning recognized DTMF input.

Note: The BeVocal VoiceXML interpreter currently has a bug that causes the terminating character to be returned as part of the result, even though the VoiceXML specification says that it should not be.

termtimeout

Not implemented. Currently, the `interdigittimeout` property is used in place of `termtimeout`.

Background Audio

The following properties control background audio:

- `fetchaudio`
- `fetchaudiodelay`
- `fetchaudiominimum`

fetchaudio

The value of the `fetchaudio` property is the URL of the audio clip to play while waiting for a VoiceXML document or object file to be fetched.

The default is not to play any audio.

Note: This property is relevant only when the interpreter fetches VoiceXML documents and object files. Background audio is never played while the interpreter fetches grammar, audio, or script files.

fetchaudiodelay

Extension. The value of the `fetchaudiodelay` property is the amount of time to wait after a VoiceXML download is started before its `fetchaudio` source is played in the background. This property is useful if you want a short period of silence before the audio starts playing. If the download completes during the period of silence, the audio will not be played at all.

The default value is 0 seconds.

fetchaudiominimum

Extension. The value of the `fetchaudiominimum` property is the minimum time interval to play the `fetchaudio` source, once started, even if the fetch result arrives in the mean time.

The default value is 0 seconds.

Fetching

The following properties control resource and document fetching:

- `audiofetchhint`
- `audiomaxage`
- `audiomaxstale`
- `caching`
- `datafetchhint`
- `datamaxage`
- `datamaxstale`
- `documentfetchhint`
- `documentmaxage`
- `documentmaxstale`
- `fetchtimeout`
- `grammarfetchhint`
- `grammarmaxage`
- `grammarmaxstale`
- `objectfetchhint`
- `objectmaxage`
- `objectmaxstale`
- `scriptfetchhint`
- `scriptmaxage`
- `scriptmaxstale`

The following additional properties are deprecated;

- `audioexpirationtime`
- `documentexpirationtime`
- `grammarexpirationtime`
- `objectexpirationtime`
- `scriptexpirationtime`

For more information about how resources are fetched, see Chapter 4, “Fetching Resources”.

audiofetchhint

The `audiofetchhint` property tells the interpreter whether it can attempt to optimize dialog interpretation by prefetching or streaming audio files. The value is one of:

- `"safe"` - Fetch audio files only when they are needed, never before.
- `"prefetch"` - Permit, but do not require, the interpreter to prefetch audio files.
- `"stream"` - Allow the interpreter to stream the fetched audio files.

  Note: The ability to stream audio files in a beta feature.

The default value is `"prefetch"`.

audiomaxage

Extension. The `audiomaxage` property specifies the maximum acceptable age, in seconds, of cached audio resources. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again.

When no value is set for this property:

- A VoiceXML 1.0 application uses the `caching` property to controls whether unexpired cached audio files are used.
- When the `vxml` tag's `version` attribute equals `2.0`, the `caching` property is ignored and unexpired cached audio files are used.

By default, no value is set for this property.

audiomaxstale

Extension. The `audiomaxstale` property specifies the maximum acceptable time, in seconds, during which expired cached audio resources can still be used. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

A cached file that has been expired for more that the maximum stale time will be refetched; one that has been stale for less than or equal to the maximum stale time will be used.

The default is 300 seconds (5 minutes). This default results in much faster performance for most applications, since it greatly reduces the number of times the interpreter must send “get if modified” requests to the HTTP server. If you want the behavior defined in the VoiceXML specification (always do a “get if modified” if there was no `Expires` header), set `audiomaxstale` to 0.

caching

In a VoiceXML 1.0 application, the `caching` property controls the use of unexpired cache files when the relevant maximum-age property is not set. The `caching` property can be set to either `"safe"` to ignore any cached file when fetching, or `"fast"` to use any cached file instead of fetching.

The default value is `"fast"`.

Note: The `caching` property is ignored when the `vxml` tag's `version` attribute equals `2.0`, and the application will use an unexpired cached copy of a file if the relevant maximum-age property is not set.

datafetchhint

Extension. The `datafetchhint` property tells the interpreter whether XML data may be prefetched. The value is one of:

- `"safe"` - Fetch data files only when they are needed, never before.
- `"prefetch"` - Permit, but do not require, the interpreter to prefetch data files.

The default value is `"safe"`.

datamaxage

Extension. The `datamaxage` property specifies the maximum acceptable age, in seconds, of cached XML data files. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again.

When no value is set for this property:

- A VoiceXML 1.0 application uses the `caching` property to controls whether unexpired cached data files are used.
- When the `vxml` tag's `version` attribute equals `2.0`, the `caching` property is ignored and unexpired cached audio files are used.

By default, no value is set for this property.

datamaxstale

Extension. The `datamaxstale` property specifies the maximum acceptable time, in seconds, during which expired cached XML data files can still be used. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

A cached file that has been expired for more that the maximum stale time will be refetched; one that has been stale for less than or equal to the maximum stale time will be used.

The default is 0 seconds.

documentfetchhint

The `documentfetchhint` property tells the interpreter whether VoiceXML documents may be prefetched. The value is one of:

- `"safe"` - Fetch document files only when they are needed, never before.
- `"prefetch"` - Permit, but do not require, the interpreter to prefetch document files.

The default value is `"safe"`.

documentmaxage

Extension. The `documentmaxage` property specifies the maximum acceptable age, in seconds, of cached VoiceXML document files. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again.

When no value is set for this property:

- A VoiceXML 1.0 application uses the `caching` property to controls whether unexpired cached VoiceXML document files are used.
- When the `vxml` tag's `version` attribute equals `2.0`, the `caching` property is ignored and unexpired cached document files are used.

By default, no value is set for this property.

documentmaxstale

Extension. The `documentmaxstale` property specifies the maximum acceptable time, in seconds, during which expired cached VoiceXML document files can still be used. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

A cached file that has been expired for more that the maximum stale time will be refetched; one that has been stale for less than or equal to the maximum stale time will be used.

The default is 0 seconds.

fetchtimeout

The value of the `fetchtimeout` property is the time-out period for fetches. The interpreter will wait this long for the resource to be returned before throwing an `error.badfetch` event. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds; `"ms"` for time in milliseconds (the default).

The default value is 1 minute.

If you set this property to 0, the interpreter waits indefinitely.

grammarfetchhint

The `grammarfetchhint` property tells the interpreter whether grammars may be prefetched. The value is one of:

- `"safe"` - Fetch grammar files only when they are needed, never before.

- `"prefetch"` - Permit, but do not require, the interpreter to prefetch grammar files.

The default value is `"prefetch"`.

grammarmaxage

Extension. The `grammarmaxage` property specifies the maximum acceptable age, in seconds, of cached grammar resources. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again.

When no value is set for this property:

- A VoiceXML 1.0 application uses the `caching` property to controls whether unexpired cached grammar files are used.
- When the `vxml` tag's `version` attribute equals `2.0`, the `caching` property is ignored and unexpired cached grammar files are used.

By default, no value is set for this property.

grammarmaxstale

Extension. The `grammarmaxstale` property specifies the maximum acceptable time, in seconds, during which expired cached grammar resources can still be used. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

A cached file that has been expired for more that the maximum stale time will be refetched; one that has been stale for less than or equal to the maximum stale time will be used.

The default is 0 seconds.

objectfetchhint

The `objectfetchhint` property tells the interpreter whether the URL contents for an `<object>` may be prefetched. The value is one of:

- `"safe"` - Fetch object files only when they are needed, never before.
- `"prefetch"` - Permit, but do not require, the interpreter to prefetch object files.

The default value is `"prefetch"`.

objectmaxage

Extension. The `objectmaxage` property specifies the maximum acceptable age, in seconds, of cached object resources. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again.

When no value is set for this property:

- A VoiceXML 1.0 application uses the `caching` property to controls whether unexpired cached object files are used.
- When the `vxml` tag's `version` attribute equals `2.0`, the `caching` property is ignored and unexpired cached object files are used.

By default, no value is set for this property.

objectmaxstale

Extension. The `objectmaxstale` property specifies the maximum acceptable time, in seconds, during which expired cached object resources can still be used. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

A cached file that has been expired for more that the maximum stale time will be refetched; one that has been stale for less than or equal to the maximum stale time will be used.

The default is 0 seconds.

scriptfetchhint

The `scriptfetchhint` property tells the interpreter whether scripts may be prefetched or not. The value is one of:.

- `"safe"` - Fetch script files only when they are needed, never before
- `"prefetch"` - Permit, but do not require, the interpreter to prefetch script files.

The default value is `"prefetch"`.

scriptmaxage

Extension. The `scriptmaxage` property specifies the maximum acceptable age, in seconds, of cached script resources. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

An unexpired cached file that does not exceed the maximum age will be used; a cached file that exceeds the maximum will be fetched again.

When no value is set for this property:

- A VoiceXML 1.0 application uses the `caching` property to controls whether unexpired cached script files are used.
- When the `vxml` tag's `version` attribute equals `2.0`, the `caching` property is ignored and unexpired cached script files are used.

By default, no value is set for this property.

scriptmaxstale

Extension. The `scriptmaxstale` property specifies the maximum acceptable time, in seconds, during which expired cached script resources can still be used. The value is a time interval expressed as an unsigned number followed by `"s"` for time in seconds (the default); `"ms"` for time in milliseconds.

A cached file that has been expired for more that the maximum stale time will be refetched; one that has been stale for less than or equal to the maximum stale time will be used.

The default is 0 seconds.

File Expiration Time

Deprecated. An earlier version of BeVocal VoiceXML used the following properties to control expiration time for audio files of various kinds:

- `audioexpirationtime` - audio files
- `documentexpirationtime` - document files
- `grammarexpirationtime` - grammar files
- `objectexpirationtime` - object files
- `scriptexpirationtime` - script files

These properties have been deprecated. You should now use the *`type`*`maxstale` property in stead of the *`type`*`expirationtime` property; for example, use `documentmaxstale` instead of `documentexpirationtime`.

For backward compatibility, if the interpreter encounters an expiration-time property, it will map the specified value to a setting for the corresponding maximum-stale-time property and emit a warning message.

Universal Grammars

Extension. The `universals` property controls the use of universal grammars. See "Universal Grammars" on page 20.

universals

Extension. The `universals` property specifies which universal grammars should be active; it deactivates all other universal grammars. The value is one of:

- `"all"` - make all universal grammars active
- `"none"` - deactivate all universal grammars
- Space-separated list of the universal grammar names - make the specified grammars active; deactivate all other universal grammars.

  The predefined universal grammars are `"help"`, `"exit"`, `"cancel"` and `"goback"`. You can define additional universal grammars by setting the `universals` property in `<grammar>` elements.

The default value for VoiceXML 1.0 applications is `"all"`. When the `vxml` tag's `version` attribute equals `2.0`, the default value is `"none"`.

The following `<property>` element deactivates the `"help"` grammar and activates the other predefined universal grammars:

```
<property
  name="universals"
  value="exit cancel goback"/>
```

Speech Errors

Extension. The following properties are BeVocal extensions for limiting the number of speech errors that can occur:

- `bevocal.maxdialogerrors`
- `bevocal.maxerrors`

A *speech error* is either a recognition errors, which normally results in a no-match event, or a time-out while waiting for user input, which normally results in a no-input event.

bevocal.maxdialogerrors

Extension. The value of the `bevocal.maxdialogerrors` property is the maximum number of speech errors that can occur within a particular execution of a dialog.

The default value is 0, which means no limit on the number of errors.

If you set the `bevocal.maxdialogerrors` property to 5, then on the 5th error in a particular form, an `error.bevocal.maxdialogerrors_exceeded` event is thrown (instead of a no-match or no-input event).

bevocal.maxerrors

Extension. The value of the `bevocal.maxerrors` property is the maximum number of speech errors that can occur during the entire call.

The default value is 0, which means no limit on the number of errors.

If you set the `bevocal.maxerrors` property to 10, then on the 10th error in the call, an `error.bevocal.maxerrors_exceeded` event is thrown (instead of a no-match or no-input event).

Go Back Facility

Experimental Extension. The following property controls the use of the go-back facility:

- `bevocal.goback`
- `bevocal.mingoback`

Note: The go-back facility is an experimental extension to the BeVocal VoiceXML 1.0 platform.See “Go-Back Facility” on page 43.

bevocal.goback

Extension. The `bevocal.goback` property specifies whether the parent element is a legal go-back destination. Set this property to `"true"` to make the parent element a legal go-back destination; set it to `"false"` to disallow going back to the parent element.

The default value is `"true"`.

You can set this property to `"false"` in a form to disable the go-back facility in that form. You can set it to `"false"` in a document to disable the go-back facility in that entire document.

bevocal.mingoback

Extension. The value of the `bevocal.mingoback` property is the minimum size of the go-back stack. The interpreter keeps at least this many entries on the stack, except at the beginning of the call when fewer steps have been executed, and after the user has said “go back” so many consecutive times that the stack has been depleted.

The default value is 0, meaning that the go-back stack is always empty and the go-back facility is effectively disabled.

8 Variable Reference

This chapter describes variables defined by VoiceXML. Session variables are available throughout the application; event-related variables are available in event handlers only.

Session Variables

The execution environment sets some session variables that you can access from within executable content:

- `session.bevocal.timeincall`
- `session.bevocal.version`
- `session.iidigits`
- `session.telephone.ani`
- `session.telephone.dnis`
- `session.uui`

`session.bevocal.timeincall`

Extension. The number of milliseconds since the beginning of this call.

`session.bevocal.version`

Extension. The version number of the VoiceXML interpreter (for example, "1.2.3").

`session.iidigits`

Information Indicator Digits

The `session.iidigits` variable is set to information about the caller's location (pay phone, etc.), when available. Complete list available in "Local Exchange Routing Guide" published by Telecordia.

`session.telephone.ani`

Automatic Number Identification

The `session.telephone.ani` variable is set to the caller's telephone number, when available.

`session.telephone.dnis`

Dialed Number Identification Service

The `session.telephone.dnis` variable is set to the number the caller dialed, when available.

`session.uui`

User-to-User Information

The `session.uui` variable is set to information from ISDN call setup, when available.

Event-Related Variables

The following variables are set whenever an event is thrown:

- _event
- _message

Both these variables are extensions to the BeVocal VoiceXML 1.0 platform.

_event

Extension. Within the anonymous scope of an event handler, the JavaScript variable `_event` is set to the name of the event that was thrown. For example:

```
<error>
  <prompt>event is <value expr="_event"/></prompt>
</error>
```

If the event is `"error.badfetch.http.500"`, this handler will say, “event is error.badfetch.http.500.”

_message

Extension. Within the anonymous scope of an event handler, the JavaScript variable `_message` is set to the message string that provides additional context about the event that was thrown. If no message was supplied when the event was thrown, this variable has the value `"undefined"`.

9 Function Reference

This chapter describes built-in JavaScript functions that are available to BeVocal VoiceXML applications. All such functions are extensions to the VoiceXML specification.

bevocal.getProperty

Extension. Gets the value of a VoiceXML property.

Syntax

`bevocal.getProperty(String *name*)`

Parameters

`*name*` - The name of the property.

Returns

The value of the specified property in the current scope.

bevocal.log

Extension. Writes a message to the BeVocal Café website.

Syntax

`bevocal.log(String *message*)`

Parameters

`*message*` - The message to be written to the website.

Description

The log function writes the specified message to the BeVocal Café call log, which you can view on the Café website. This function performs the same operation as a `<log>` VoiceXML element with no `label` or `expr` attribute.