VoxML 1.1 Language Reference

Revision 1.1

Motorola, Inc.
Internet and Connectivity Solutions Division
55 Shuman Blvd., 6th floor
Naperville, IL 60563
USA

Contents

3. Appendices

1. An Introduction to VoxML

Overview

One reason the World Wide Web has become so popular is the relative ease with which authors can create content using HTML. For all its faults, HTML has served its purpose well as a vehicle for the presentation of rich text, images, hypertext links, and simple GUI input controls.

VoxML gives the same ease of production to voice applications. In one sense, VoxML offers the same building blocks as HTML: text (which is read via text-to-speech), recorded sound samples (analogous to images), navigational controls, and input controls.

However, writing a voice application is very different from writing a GUI application, and thus the structure of VoxML must be very different from HTML. Here are some examples of the resulting differences:

- HTML specifies a two-dimensional layout, whereas VoxML specifies a "layout" in the time dimension only.
- HTML is designed and displayed in whole-page units. VoxML is designed and delivered in whole-dialog units, with dialogs presented in much smaller units, called *steps*.
- A single HTML page often presents the user with dozens of options, which would overwhelm the user of a voice application. In general, voice applications must limit the number of options available at a given step in the dialog to ease the burden on the user's short-term memory and to improve the performance of the speech recognition.

What is VoxML?

VoxML is based on the Extensible Markup Language (XML). As such, VoxML follows all of the syntactic rules of XML, with semantics that support the creation of interactive speech applications. For more information on the exact structure of the language with respect to XML, see the VoxML 1.1 Document Type Definition (DTD) in Appendix A.

Purpose of this Document

This document is the official language reference for VoxML 1.1. It contains information on the syntax of the elements and their attributes, example usage, the structure of VoxML documents (or DIALOGs), and pointers to other reference documentation that may be helpful when developing applications using VoxML.

This document is intended for VoxML application developers.

Structure of a VoxML Document

VoxML documents have a hierarchical structure, in which every element (except the DIALOG element) is contained by another element. In contrast to HTML, there is a strict containment relationship enforced by the language, so there can be no ambiguity as to which element contains which.

DIALOGs and STEPs

The two most fundamental elements in VoxML are the DIALOG element and the STEP element. These elements provide the basic structure of a VoxML application.

The DIALOG element defines the scope of a VoxML document. All other VoxML elements are contained by the DIALOG element. In normal cases, one can equate a DIALOG to a file, in much the same way that there is one HTML element per file when developing HTML-based applications.

The STEP element defines a state within a DIALOG, or to say it another way, the STEP element defines an application state.

Together the DIALOG element and the associated STEP elements define a state machine that represents an interactive dialogue between the application and a user. When the VoxML voice browser interprets the VoxML document, it will navigate through the DIALOG to different STEPs as a result of the user's responses.

Here is a simple VoxML example, which has one DIALOG containing two STEPs.

```
1  <?xml version="1.0"?>
2  <DIALOG>
3  <STEP NAME="init">
4    <PROMPT> Please select a soft drink. </PROMPT>
5    <HELP> Your choices are coke, pepsi, 7 up,
6             or root beer.
7    </HELP>
8    <INPUT TYPE="OPTIONLIST" NAME="drink" NEXT="#confirm">
9      <OPTION VALUE="coke">      coke      </OPTION>
10     <OPTION VALUE="pepsi">     pepsi     </OPTION>
11     <OPTION VALUE="7 up">      7 up      </OPTION>
12     <OPTION VALUE="root beer"> root beer </OPTION>
13   </INPUT>
14 </STEP>
15
16 <STEP NAME="confirm">
17   <PROMPT> You ordered <VALUE NAME="drink"/>. </PROMPT>
18 </STEP>
19 </DIALOG>
```

As when any VoxML document is interpreted, the voice browser begins by executing the STEP called "init". The user will hear the text contained by the PROMPT element. If the user were to ask for "help" before making a selection, the user would hear the text contained with the HELP element. After the user makes a selection, the voice browser will execute the STEP named "confirm", which will simply read back the user's selection and then exit the application.

There are a few important points to be made after looking at this first code example. First, you will notice that line 1 of the source code contains the XML declaration string, which is required to be the first line of all VoxML documents.

Also, STEPs in a VoxML application are executed based on the user's responses not on the order of the STEPs within the source file. Although the definition of the "init" STEP appears on line 3 and the definition of the "confirm" STEP appears on line 16, the order in which they are defined has no impact on the order in which the voice browser navigates through them.

Lastly, the line numbers shown in the example listing are not part of the source code. They are shown only to make referencing the code simpler

The Basic Elements

VoxML contains a lot of elements, which provides the application developer with a lot of flexibility and power in the language. Section 2 of this document describes these elements in detail.

A few basic elements are used in almost every DIALOG, including the soft drink selection dialog we just introduced. These elements are DIALOG, STEP, PROMPT, HELP, and INPUT. If you are new to VoxML, you should familiarize yourself with these basic elements before progressing to the other, less frequently used elements.

Other Reference Documentation

The following resources provide more information that may be helpful when developing interactive speech applications using VoxML.

VoxML Application Development Guide

This document contains design guidelines and examples for VoxML application developers. It presents a description of the VoxML development environment including system requirements, installation procedures, development tools and a methodology for the development process. The main goal of the document is to provide developers with explicit guidelines for developing successful VoxML applications.

2. VoxML Element Reference

This section describes the VoxML elements, their attributes, and their syntax. Examples are provided to help show common usage of each element.

ACK Element

It is sometimes necessary to double-check some piece of information that the user has just provided. While an application developer could create an additional STEP that would prompt the user with the data just obtained and then ask if this was correct, the ACK (acknowledge) element provides an easier way to specify common sorts of acknowledgements.

The ACK element can also be used to output an acknowledgement with no user confirmation. This is useful if there is a noticeable delay in getting to the next state, as in the case where this STEP is getting information from a server.

An ACK element must be contained within a STEP or a CLASS element.

Syntax

<ACK [CONFIRM="*value*" [REPROMPT="*value*"]]>
text
</ACK>

Attributes

Attribute Name	Allowed Values
CONFIRM	YORN (default)
REPROMPT	Y N (default)

Examples

```
1  <STEP NAME="card_type">
2      <PROMPT>
3          What type of credit card do you have?
4      </PROMPT>
5      <INPUT NAME="type" TYPE="OPTIONLIST" NEXT=″#exp_date″>
6          <OPTION VALUE=″visa″>    visa         </OPTION>
7          <OPTION VALUE=″mc″>      mastercard   </OPTION>
8          <OPTION VALUE=″discover″>discover     </OPTION>
9      </INPUT>
10     <ACK CONFIRM="YORN" REPROMPT="Y">
11         I thought you said <VALUE NAME="type"/>
12         <BREAK/> Is that correct?
13     </ACK>
14 </STEP>
```

In this example, the ACK element is used to confirm the user's choice of credit card. When this code is interpreted by the VoxML voice browser, it will speak the text of

the PROMPT element using text-to-speech technology, wait until the user responds with “visa”, “mastercard”, or “discover”, and then ask the user to confirm the type of card was recognized correctly.

If the user answers “yes” to the ACK, the browser will proceed to the STEP named “exp”. If the user answers “no” to the ACK, the text of the PROMPT will be read again, and the user will be allowed to make his or her choice again — the browser re-enters the STEP as if for the first time.

The CONFIRM attribute specifies the type of input field expected from the user. The “YORN” (yes or no) type is bound to a built-in speech recognition grammar that allows all sorts of positive and negative responses in addition to a simple yes or no. Other CONFIRM types are on the drawing board (e.g., a “REPEAT” that requires the user to repeat a newly-specified password), but none are available yet.

The REPROMPT attribute specifies what to do if the confirmation fails. If REPROMPT is “Y”, the user will be prompted again. If it is “N” the user is not prompted.

AUDIO Element

The AUDIO element specifies an audio file that should be played. AUDIO elements can be used as an alternative anywhere that you would read text to the user.

An AUDIO element can be contained within a PROMPT, EMP, PROS, HELP, ERROR, CANCEL, or ACK element, that is, wherever text that is spoken to the user can go.

Syntax

<AUDIO SRC="*value*" />

Attributes

Attribute Name	Allowed Values
SRC	*audio file URL*

Examples

```
1 <PROMPT>
2     At the tone, the time will be 11:59 p m
3     <AUDIO SRC="http://localhost/sounds/beep.wav"/>
4 </PROMPT>
```

The above code is a simple example of an audio sample included in a PROMPT element. When interpreted by the VoxML Voice Browser, this code will speak the text from line 2 using text-to-speech technology, and then play the WAV file "beep.wav" as specified by the AUDIO element on line 3.

For a description of the audio formats currently supported by the voice browser, see the release notes for the VoxML SDK.

BREAK Element

The BREAK element is used to insert a pause into content to be presented to the user. BREAK elements can be used anywhere that you would read text to or play audio samples for the user.

The BREAK element can be contained within a PROMPT, EMP, PROS, HELP, ERROR, CANCEL, or ACK element.

Syntax

<BREAK [MSECS="*value*" | SIZE="*value*"] />

Attributes

Attribute Name	Allowed Values
MSECS	*milliseconds (integer)*
SIZE	NONE
	SMALL
	MEDIUM (default)
	LARGE

Examples

```
1 <PROMPT>
2     Welcome to Earth. <BREAK MSECS="250"/>
3     How may I help you?
4 </PROMPT>
```

This first example illustrates the use of the BREAK element with the MSECS attribute, inside a PROMPT. When interpreted by the VoxML voice browser, this code would speak the text "Welcome to Earth.", pause for 250 milliseconds, and then speak the text "How may I help you?".

As an alternative to specifying an exact number of milliseconds, an application developer can use the SIZE attribute of the BREAK element to control the duration of the pause:

```
1 <PROMPT>
2     Welcome to Earth. <BREAK SIZE="MEDIUM"/>
3     How may I help you?
4 </PROMPT>
```

The actual duration of "SMALL", "MEDIUM", and "LARGE" are system defined, and may change. Use the MSECS attribute if a specific duration is required.

CANCEL Element

The CANCEL element enables the application developer to define the behavior of the VoxML application in response to a user's request to cancel the current PROMPT. If the application developer does not define the behavior of CANCEL for a given STEP, the system default behavior will be used.

The default behavior for the CANCEL element is to stop the PROMPT, and then process any interactive INPUT.

The CANCEL element, like the HELP element, can be invoked through a variety of phrases. The user may say only the word "cancel", or the user may say "I would like to cancel, please." In either case, the CANCEL element will be interpreted.

The CANCEL element can be contained within a STEP or a CLASS element.

Syntax

<CANCEL NEXT="*value*" [NEXTMETHOD="*value*"] />
or
<CANCEL NEXT="*value*" [NEXTMETHOD="*value*"] > *text* </CANCEL>

Attributes

Attribute Name	Allowed Values
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST

Examples

```
1  <STEP NAME="report">
2     <CANCEL NEXT="#traffic_menu"/>
3     <PROMPT> Traffic conditions for Chicago,
4        Illinois, Monday, May 18.  Heavy
5        congestion on ...
6     </PROMPT>
7     <INPUT TYPE="OPTIONLIST">
8        <OPTION NEXT="#report"> repeat </OPTION>
9        <OPTION NEXT="#choose"> new city </OPTION>
10    </INPUT>
11 </STEP>
```

The code on line 2 illustrates the use of the CANCEL element to specify that when the user says "cancel", the browser should proceed to the STEP named "traffic_menu", instead of the default behavior, which would be to simply stop the PROMPT from playing and wait for a user response. Users can also interrupt the PROMPT by speaking a valid OPTION. In this example, the user could interrupt the PROMPT and get the traffic conditions for a different city by saying "new city".

CASE Element

The CASE element is used to define the flow of control of the application, based on the values of internal VoxML variables.

The CASE element can be contained by a SWITCH element, or by an INPUT element, when using an INPUT type that collects a single value (DATE, DIGITS, MONEY, PHONE, TIME, and YORN).

Syntax

<CASE VALUE="*value*" NEXT="*value*" [NEXTMETHOD="*value*"] />

Attributes

Attribute Name	Allowed Values
VALUE	*literal value*
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST

Examples

```
1 <SWITCH FIELD="pizza">
2     <CASE VALUE="pepperoni" NEXT="#p_pizza"/>
3     <CASE VALUE="sausage"   NEXT="#s_pizza"/>
4     <CASE VALUE="veggie"    NEXT="#v_pizza"/>
5 </SWITCH>
```

The code on lines 2-4 shows the use of the CASE element within the SWITCH element. In this example, the CASE elements are used to direct the browser to different STEPs based on the value of the VoxML variable “pizza”.

Many VoxML elements have the NEXT and NEXTMETHOD attributes. These specify the STEP to go to next, using standard HTTP conventions. If you want to go to a STEP within the current VoxML document, use the “#step_name” notation. If you want to go to an entirely new VoxML document and start at its “init” STEP, set NEXT to the URL of that document and optionally provide the HTTP method used in obtaining that document.

CLASS Element

The CLASS element defines a set of elements that are to be reused within the context of a DIALOG. STEPs or other CLASSs may inherit the definitions of a given CLASS. The CLASS element allows the application developer to define a set of elements once, and then use them several times. The CLASS element is often used to define the default behavior of ERROR, HELP, and CANCEL, within a given DIALOG.

The CLASS element can be contained by a DIALOG element only.

Syntax

<CLASS NAME="*value*" [PARENT="*value*"] [BARGEIN="*value*"] [COST="*value*"] >
VoxML
</CLASS>

Attributes

Attribute Name	Allowed Values
NAME	*The name of the class (identifier).*
PARENT	*The name of the superclass (identifier).*
BARGEIN	Y (default) – *allows user "barge-in" to stop prompts.* N
COST	*The "cost" of executing each STEP belonging to this class.*[1]

Examples

```
1  <CLASS NAME="simple">
2      <HELP> Your choices are <OPTIONS/> </HELP>
3      <ERROR> I did not understand what you said.
4          Valid responses are <OPTIONS/> </ERROR>
5  </CLASS>
6
7  <STEP NAME="beverage" PARENT="simple">
8      <PROMPT> Please choose a drink. </PROMPT>
9      <INPUT NAME="drink" TYPE="OPTIONLIST">
10         <OPTION NEXT="#food"> coke </OPTION>
11         <OPTION NEXT="#food"> pepsi </OPTION>
12     </INPUT>
13 </STEP>
14
15 <STEP NAME="food" PARENT="simple">
```

[1] The COST attribute is a platform-dependent feature. A given browser implementation may or may not support each platform-dependent feature.

```
16     <PROMPT> Please choose a meal. </PROMPT>
17     <INPUT NAME="meal" TYPE="OPTIONLIST">
18        <OPTION NEXT="#deliver"> pizza </OPTION>
19        <OPTION NEXT="#deliver"> tacos </OPTION>
20     </INPUT>
21 </STEP>
```

The code shown on lines 1-5 illustrates the use of the CLASS element to define a HELP element and an ERROR element that will be used in several STEPs within this DIALOG.

The code on lines 7 and 15 illustrates the use of the PARENT attribute on the STEP element to refer to the CLASS element, and therefore inherit the behaviors defined within it.

When interpreted by the VoxML voice browser, the STEPs defined on lines 7-13 and lines 15-21 will behave as if the HELP and ERROR elements that are defined in the CLASS were defined explicitly in the STEPs themselves.

Inheritance

A CLASS's sub-elements and attributes are inherited by its subclasses and steps unless they are overridden somewhere down the hierarchy. However, inheritance does not make sense for some of the sub-elements. Here are the rules used.

COST. The "cost" of a step has meaning only to a particular platform running VoxML. The VoxML voice browser will calculate the summary costs of completing each STEP in a user session, and will write out a billing record for that user. The cost of a STEP is the value of its COST attribute. If there is no explicit COST attribute, the cost is inherited from the parent CLASS, if any, and if the parent CLASS doesn't have a COST associated with it, the search continues on up to the grandparent, if there is one, and so on. If neither the STEP nor any of its superclasses have a COST attribute, the cost of completing that STEP is zero.

BARGEIN. This governs whether or not the user can stop the TTS engine by speaking or not. This attribute is inherited just like COST, so if a STEP doesn't define BARGEIN, its parent CLASS is examined, and so on.

CANCEL. This element is inherited in a process just like the COST and BARGEIN attributes are inherited. If a STEP has no CANCEL element, it inherits its parent class's one, if any, and so on up the chain.

HELP. HELP elements are interesting because the inheritance of HELP information is sensitive to their ORDINAL attributes. This means that if a STEP has a HELP with ORDINAL="7", it will not inherit a HELP ORDINAL="7" from its class, though it will inherit HELPs with other ORDINAL values.

ERROR. These elements are inherited based on both ORDINAL value *and* TYPE value.

DIALOG Element

The DIALOG element is the fundamental element of VoxML. If one were to imagine a VoxML document as a tree, the DIALOG element would be the root of the tree. The DIALOG element defines the basic unit of context within a VoxML application, and in the common case, there is one DIALOG element per URL.

Each VoxML DIALOG must contain exactly one STEP element named “init”. The execution of the VoxML application begins with the STEP named “init”.

A DIALOG element cannot be contained by any VoxML element.

Syntax

<DIALOG [BARGEIN="*value*"] > *VoxML* </DIALOG>

Attributes

Attribute Name	**Allowed Values**
BARGEIN	Y (default)
	N

Examples

```
1 <DIALOG>
2     <STEP NAME="init">
3         <PROMPT> Welcome to VoxML. </PROMPT>
4     </STEP>
5 </DIALOG>
```

The above code shows a simple, yet complete VoxML DIALOG. The DIALOG element is specified on lines 1 and 5 and contains a single STEP element named “init”. The STEP has a single PROMPT that will be read via text-to-speech. Since there is no INPUT defined in this STEP, the VoxML application will terminate immediately after the PROMPT is read.

The BARGEIN attribute controls whether or not the user can "barge into" or interrupt the reading of the prompt by saying something. If BARGEIN is "Y", and the user says something, the reading of the prompt will terminate prematurely. If it is "N", the prompt will be read to completion.

EMP Element

The EMP ("emphasis") element is used to mark a span of text whose emphasis level should be different when that text is read to the user. The EMP element can be used anywhere that text is read to the user, and it can be nested.

The EMP element can be contained within a PROMPT, EMP, PROS, HELP, ERROR, CANCEL, or ACK element.

Syntax

<EMP [LEVEL="*value*"] > *text* </EMP>

Attributes

Attribute Name	Allowed Values
LEVEL	NONE
	REDUCED
	MODERATE (default)
	STRONG

Examples

```
1 <PROMPT>
2     The weather today is going to be
3     <EMP LEVEL="strong"> really </EMP>
4     bad.  Up to 36 inches of snow will fall …
5 </PROMPT>
```

The above code illustrates the use of the EMP element to apply "strong" emphasis to the word "really" in a prompt.

The actual effect on the speech output is determined by the text-to-speech (TTS) software used by the voice browser. Since the desktop simulator uses different TTS software than the voice browser in the highly-scalable network system, the output may vary. To achieve a specific emphatic effect, use the PROS element instead of the EMP element.

ERROR Element

The ERROR element enables the application developer to define the behavior of the VoxML application in response to an error. If the application developer does not define the behavior of ERROR for a given STEP, the default behavior will be used.

The default behavior for the ERROR element is to speak the phrase "An error has occurred", remain in the current STEP, replay the PROMPT, and wait for the user to respond.

The ERROR element can be contained within a STEP or a CLASS element.

A given ERROR element applies to all types of errors if no type is specified, or just to one specific type of error. By default, all types of errors are addressed by the element. The individual error types occur in the following circumstances:

- NOMATCH: The speech recognizer detected an utterance, but could not match that utterance to any of the possible phrases in the grammar with an acceptable level of confidence.
- NOSPEECH: The speech recognizer did not detect an utterance from the user within the specified time.
- TOOLITTLE: When collecting an INPUT of type "DIGITS", fewer than the minimum number of digits were collected.
- TOOMUCH: When collecting an INPUT of type "DIGITS", more than the maximum number of digits were collected.
- NOAUTH: An attempt to collect user profile information failed because the user opted not to transmit the information.
- NSF: An attempt to charge for listening to a STEP has failed due to lack of funds in the user's account.
- BADNEXT: A failure occurred while trying to fetch and execute the URL given in a NEXT attribute.

Syntax

```
<ERROR
        [TYPE="value"]
        [ORDINAL="value"]
        [REPROMPT="value"]
        [NEXT="value" [NEXTMETHOD="value"] ] >
         text
</ERROR>
```

Attributes

Attribute Name	Allowed Values
TYPE	ALL (default)
	NOMATCH
	NOSPEECH
	TOOLITTLE

	TOOMUCH
	NOAUTH
	NSF
	BADNEXT
ORDINAL	*integer*
REPROMPT	Y
	N (default)
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default)
	POST

Examples

```
1  <STEP NAME="errors">
2     <ERROR TYPE="NOMATCH"> First error message.
3        I did not understand what you said. </ERROR>
4     <ERROR TYPE="NOMATCH" ORDINAL="2">
5        Second error message.
6        I did not understand what you said. </ERROR>
7     <PROMPT> This step tests error messages.
8        Say 'oops' twice. Then say 'done' to
9        choose another test. </PROMPT>
10    <INPUT TYPE="OPTIONLIST">
11       <OPTION NEXT="#end"> done </OPTION>
12    </INPUT>
13 </STEP>
```

The code shown above illustrates the use of the ERROR element to define the application's behavior in response to an error. On line 2, we define the error message to be used the first time an error of type "NOMATCH" occurs in this STEP. On line 4, we define the error message to be used the second and all subsequent times an error of type "NOMATCH" occurs in this STEP.

The ORDINAL attribute determines which message will be used in the case of repeated errors within the same STEP. The VoxML voice browser will choose an error message based on this simple algorithm: If the error has occurred 3 times, the browser will look for an ERROR element with ORDINAL of "3". If no such ERROR element has been defined, the voice browser will look for an ERROR with ORDINAL of "2", and then "1", and then an ERROR with no ORDINAL defined.

So, if we had defined an ERROR element with ORDINAL of "6" in the STEP shown above, and the same error occurred 6 times in a row, the user would hear the first error message one time, then the second error message 4 times, and finally the error message with ORDINAL of "6".

HELP Element

The HELP element enables the application developer to define the behavior of the VoxML application when the user asks for help. If the application developer does not define the behavior of HELP for a given STEP, the system default behavior will be used.

The HELP element, like CANCEL the element, can be invoked through a variety of phrases. The user may say only the word "help", or the user may say "I would like help, please." In either case, the HELP element will be interpreted.

The default behavior for the HELP element is to stop the PROMPT (if one is playing), speak the phrase "No help is available.", remain in the current STEP, and process any interactive INPUT.

The HELP element can be contained within a STEP or a CLASS element.

Syntax

```
<HELP
        [ORDINAL="value"]
        [REPROMPT="value"]
       [NEXT="value" [NEXTMETHOD="value"] ] >
        text
</HELP>
```

Attributes

Attribute Name	Allowed Values
ORDINAL	*integer*
REPROMPT	Y N (default)
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST

Examples

```
1  <STEP NAME="helps">
2     <HELP REPROMPT="Y"> First help message.
3        You should hear the prompt again. </HELP>
4     <HELP ORDINAL="2"> Second help message.
5        You should not hear the prompt now. </HELP>
6     <PROMPT> This step tests help prompts.
7        Say 'help' twice. Then say 'done' to
8        choose another test. </PROMPT>
9     <INPUT TYPE="OPTIONLIST">
10       <OPTION NEXT="#end"> done </OPTION>
```

```
11    </INPUT>
12 </STEP>
```

The code above illustrates the use of the HELP element to define the application's behavior in response to the user input “help”. On line 2, we define the help message to be used the first time the user says “help”. On line 4, we define the help message to be used the second and all subsequent times the user says “help”. It should also be noted that through the use of the REPROMPT attribute, the prompt will be repeated after the first help message, but it will not be repeated after the second help message.

The ORDINAL attribute determines which message will be used in the case of repeated utterances of “help” within the same STEP element. The VoxML voice browser will choose a help message based on this simple algorithm: If the user has said “help” 3 times, the browser will look for a HELP element with ORDINAL of “3”. If no such HELP element has been defined, the voice browser will look for a HELP with ORDINAL of “2”, and then “1”, and then a HELP with no ORDINAL defined.

So, if we had defined a HELP element with ORDINAL of “6” in the STEP shown above, and the user said “help” 6 times in a row, the user would hear the first help message one time, then the second help message 4 times, and finally the help message with ORDINAL of “6”.

INPUT Element

The INPUT element is used to define the valid user input in a STEP. The application developer can define the type of input as well as specific values that are to be recognized.

The INPUT element can exist only within a STEP element.

Syntax

Because the syntax of the different types of INPUTs varies widely, each of the types of INPUT elements will be described in its own section.

Attributes

Attribute Name	Allowed Values
TYPE	DATE
	DIGITS
	GRAMMAR
	HIDDEN
	MONEY
	NONE
	NUMBER
	OPTIONLIST
	PHONE
	PROFILE
	RECORD
	TIME
	YORN

Examples

For examples, please reference the section that describes the type of INPUT that you would like to use.

Note that the different types of INPUT element expect *one* real spoken input from the user, *except for* INPUT HIDDEN, which takes an input from a literal value in the VoxML document; INPUT NONE, which expects no input at all and just does branching; INPUT PROFILE, which sets a user profile variable obtained from some external data source; and INPUT GRAMMAR, which can return multiple inputs.

If the input from the user is to be kept around between STEPs, use the NAME attribute to specify the VoxML variable to store it in. You can also set the TIMEOUT value for close control over the interval after which the user will get a "NOSPEECH" error message.

INPUT Element : Type DATE

The DATE input is used to collect a calendar date from the user.

Syntax

<INPUT TYPE="DATE"
NAME="*value*"
[NEXT="*value*" [NEXTMETHOD="*value*"]]
[TIMEOUT="*value*"] />

Attributes

Attribute Name	Allowed Values
NAME	*identifier*
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST
TIMEOUT	*milliseconds (integer)*

Examples

```
1 <STEP NAME="init">
2    <PROMPT> What is your date of birth? </PROMPT>
3    <INPUT TYPE="DATE" NAME="dob" NEXT="#soc"/>
4 </STEP>
```

The code on line 3 uses DATE INPUT to gather the user's birthday, store it in the VoxML variable "dob", and then go to the STEP named "soc".

Data Format

The DATE input makes use of an input grammar to interpret the user's response, and stores that response into a VoxML variable in a standard format. The DATE input grammar can interpret dates expressed in several different formats.

A fully defined date like "next Friday, July 10th, 1998" is stored as "`07101998|July|10|1998|Friday|next`". If the full date cannot be determined from the user's response, the ambiguous parts of the response are omitted from the data.

The response "July 4th", is stored as "`????????|July|4||`", "Tomorrow" becomes "`????????||||tomorrow`", "The 15th" is stored as "`????????||15||`", and "Monday" becomes "`????????|||Monday|`".

INPUT Element : Type DIGITS

The DIGITS input is used to collect a series of digits from the user. These may be entered either verbally, or by keying them in on the telephone touch pad.[2]

Syntax

<INPUT TYPE="DIGITS"
NAME="*value*"
[NEXT="*value*" [NEXTMETHOD="*value*"]]
[TIMEOUT="*value*"]
[MIN="*value*"] [MAX="*value*"] />

Attributes

Attribute Name	Allowed Values
NAME	*identifier*
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST
TIMEOUT	*milliseconds (integer)*
MIN	*minimum number of digits (integer)*
MAX	*maximum number of digits (integer)*

Examples

```
1 <STEP NAME="init">
2     <PROMPT> Please say your pin now. </PROMPT>
3     <INPUT TYPE="DIGITS" NAME="pin" NEXT="#doit"/>
4 </STEP>
```

The code on line 3 illustrates the use of the DIGITS INPUT to collect digits from the user, store the number in the VoxML variable named “pin”, and then go to the STEP named “doit”.

If the user were to say, “four five six”, in response to the PROMPT shown on line 2, the value “456” would be stored in the VoxML variable “pin”. The DIGITS input type will collect the digits 0 (i.e., zero) through 9 (i.e., nine), but will not understand when given input like "twenty-nine" or "four hundred". The NUMBER input type is used in those situations.

[2] Some implementations may not allow you to intermix both voice and DTMF entry of digits.

INPUT Element : Type GRAMMAR

While many things can be done with other INPUT elements, sometimes the developer will want to develop application-specific speech recognition grammars rather than rely on the built-in ones for recognizing dates, times, numbers, etc. The INPUT GRAMMAR element supports this, allowing the developer to specify an input grammar that is to be used when interpreting the user's responses.

The grammar can be one that is built into a customized voice browser, or it can be one loaded dynamically. In both cases they are specified via URLs. Dynamic grammars live on the web and are accessed with a normal "http" URL. Built-in grammars are accessed via URLs using the special "gram" protocol.[3]

There are three forms of the INPUT GRAMMAR element. In the first one, each named speech recognition slot is copied into a VoxML variable having the same name. In the second form, the RENAME sub-elements allow you to bind the recognition slots to variables with different names. The third form has RESPONSE sub-elements, which determine what actions to take when the speech recognition results in only some of the slots being filled. See the entries for RENAME and RESPONSE for more information.

Syntax

```
<INPUT TYPE="GRAMMAR"
        SRC="value"
        NEXT="value" [NEXTMETHOD="value"]
        [TIMEOUT="value"] />
        …or…
<INPUT TYPE="GRAMMAR"
        SRC="value"
        NEXT="value" [NEXTMETHOD="value"]
        [TIMEOUT="value"] >
        RENAME elements
</INPUT>
        …or…
<INPUT TYPE="GRAMMAR"
        SRC="value"
        [TIMEOUT="value"]
        [NEXT="value" [NEXTMETHOD="value"] ] >
        RESPONSE elements
</INPUT>
```

Attributes

Attribute Name	Allowed Values
SRC	*grammar URL*

[3] Unfortunately there is as yet no common standard grammar syntax understood by all speech recognizers, so care must be taken to ensure that external grammar's syntax is consistent with the speech recognizer used by your particular voice browser implementation(s).

NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST
TIMEOUT	*milliseconds (integer)*

Examples

```
1 <STEP NAME="init">
2     <PROMPT> Say the month and year in which the
3         credit card expires. </PROMPT>
4     <INPUT TYPE="GRAMMAR"
5         SRC="gram://.SomeGrammar/month/year"
6         NEXT="#stepNineteen"/>
7 </STEP>
```

The code on lines 4, 5, and 6 illustrates the use of the GRAMMAR INPUT to collect a month and year from the user, store the interpreted values in variables named "month" and "year", and then go to the step named "stepNineteen".

The "gram" URL indicates that the voice browser has been customized with a built-in grammar called ".SomeGrammar", which returns recognition slots called "month" and "year". Because there are no RENAMEs in this element, the VoxML variables "month" and "year" are assigned the slot values.

INPUT Element : Type HIDDEN

The HIDDEN input is used to assign a value to a VoxML variable without user participation. Since this value is not obtained interactively from the user, a STEP may contain any number of INPUT HIDDENs in addition to the one interactive INPUT it is allowed to have.

Syntax

<INPUT TYPE="HIDDEN" NAME="*value*" VALUE="*value*"/>

Attributes

Attribute Name	Allowed Values
NAME	*identifier*
VALUE	*literal value*

Examples

```
1  <STEP NAME="init">
2      <PROMPT> Login sequence complete.
3          Are you ready to place your order?
4          </PROMPT>
5      <INPUT TYPE="HIDDEN" NAME="firstname"
6          VALUE="Bill"/>
7      <INPUT TYPE="HIDDEN" NAME="lastname"
8          VALUE="Clinton"/>
9      <INPUT TYPE="HIDDEN" NAME="favorite"
10         VALUE="fries"/>
11     <INPUT TYPE="OPTIONLIST">
12         <OPTION NEXT="#order"> yes </OPTION>
13         <OPTION NEXT="#wait"> not yet </OPTION>
14     </INPUT>
15 </STEP>
```

In the example code shown above, the HIDDEN INPUT type is used to create VoxML variables and assign values to those variables. In this particular example, the user has completed the login sequence and the application designer chose to save certain information in VoxML variables as soon as the user's identity has been established. This information could then be used later in the application without requiring another access into the database.

Notice that when using the HIDDEN INPUT that it is permissible to have more than one INPUT element in the same STEP. This is because the HIDDEN INPUT is not an interactive INPUT. Each STEP can contain only one INPUT that accepts a response from the user.

INPUT Element : Type MONEY

The MONEY input is used to collect a monetary amount from the user.

Syntax

<INPUT TYPE="MONEY" NAME="*value*" NEXT="*value*"
[NEXTMETHOD="*value*"] [TIMEOUT="*value*"] />

Attributes

Attribute Name	Allowed Values
NAME	*identifier*
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST
TIMEOUT	*milliseconds (integer)*

Examples

```
1 <STEP NAME="init">
2     <PROMPT> How much would you like to deposit?
3         </PROMPT>
4     <INPUT TYPE="MONEY" NAME="dep" NEXT="#deposit"/>
5 </STEP>
```

The above example illustrates the use of the MONEY input type to collect the amount of money that the user would like to deposit in his account, store that amount in a VoxML variable named “dep”, and then go to the STEP named “deposit”.

Data Format

The MONEY input makes use of an input grammar to interpret the user’s response and store that response in a standard format. The input grammar is able to interpret several ways of expressing monetary amounts.[4]

The data is stored in integer format, in terms of cents, so that “five cents” is stored as “5”, “five dollars” is stored as “500”, and “a thousand” is stored as “100000”. Note that no punctuation is added to the digits. Also note that in the case where the units are ambiguous, the grammar assumes dollars, as in the example above in which “a thousand” was stored as if the user had said “a thousand dollars”.

[4] But note that the exact capabilities of the built-in grammars depend on the specific implementation of the VoxML voice browser being used. In addition, the grammar must be localized for different currencies.

INPUT Element : Type NONE

Input type NONE is used to specify the next location for the voice browser to go to continue execution, in STEPs where no response is collected from the user.

Syntax

<INPUT TYPE="NONE" NEXT="*value*" [NEXTMETHOD="*value*"] />

Attributes

Attribute Name	Allowed Values
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST

Examples

```
1 <STEP NAME="init">
2     <PROMPT> Welcome to the system. </PROMPT>
3     <INPUT TYPE="NONE" NEXT="#mainmenu"/>
4 </STEP>
```

The code shown above illustrates the use of the NONE input type to jump to another STEP in this dialog without waiting for any user response. In this example, the user would here the phrase "Welcome to the system" followed immediately by the prompt of the main menu.

INPUT Element : Type NUMBER

The NUMBER input is used to collect numbers from the user.[5]

Syntax

<INPUT TYPE="NUMBER" NAME="*value*" NEXT="*value*"
[NEXTMETHOD="*value*"] [TIMEOUT="*value*"] />

Attributes

Attribute Name	Allowed Values
NAME	*identifier*
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST
TIMEOUT	*milliseconds (integer)*

Examples

```
1 <STEP NAME="init">
2     <PROMPT> Please say your age now. </PROMPT>
3     <INPUT TYPE="NUMBER" NAME="age" NEXT="#doit"/>
4 </STEP>
```

The code on line 3 illustrates the use of the NUMBER INPUT to collect numbers from the user, store the number in the VoxML variable named “age”, and then go to the STEP named “doit”.

If the user were to say, “eighteen”, in response to the PROMPT shown on line 2, the value “18” would be stored in the VoxML variable “age”. The NUMBER input type will collect numbers like 20 (i.e. twenty), only one number per input. To collect a series of digits like “four five six” (i.e. “456”), use the DIGITS input type.

[5] Again, specific implementations may vary in what is meant by "number". The current VoxML voice browser defines them to be non-negative integers from one to 24 digits.

INPUT Element : Type OPTIONLIST

The OPTIONLIST input is used to specify a list of options from which the user can select. This input type is used in conjunction with the OPTION element, which defines the specific user responses and the behavior associated with each.

Syntax

<INPUT TYPE="OPTIONLIST"
[NAME="*value*"]
[TIMEOUT="*value*"]
[NEXT="*value*" [NEXTMETHOD="*value*"]] >
OPTION elements
</INPUT>

Attributes

Attribute Name	Allowed Values
NAME	*identifier*
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST
TIMEOUT	milliseconds (integer)

Examples

```
1 <STEP NAME="init">
2    <PROMPT> What would you like to drink? </PROMPT>
3    <INPUT TYPE="OPTIONLIST">
4       <OPTION NEXT="#coke">  coke      </OPTION>
5       <OPTION NEXT="#coke">  coca-cola </OPTION>
6       <OPTION NEXT="#pepsi"> pepsi     </OPTION>
7       <OPTION NEXT="#rc">    r c       </OPTION>
8    </INPUT>
9 </STEP>
```

In this simple example, the VoxML voice browser will go to a different STEP depending on which cola the user selects. As defined on lines 4 and 5, if the user said “coke” or “coca-cola”, the voice browser would go to the STEP named “coke”.

If you want to store the user's selection in a VoxML variable, use the INPUT OPTIONLIST element's NAME attribute, and the OPTION element's VALUE attribute, like this:

```
1 <STEP NAME="init">
2    <PROMPT> What would you like to drink? </PROMPT>
3    <INPUT TYPE="OPTIONLIST" NAME="favorite_drink">
4      <OPTION VALUE="coke"  NEXT="#coke">  coke      </OPTION>
5      <OPTION VALUE="coke"  NEXT="#coke">  coca-cola </OPTION>
6      <OPTION VALUE="pepsi" NEXT="#pepsi"> pepsi     </OPTION>
7      <OPTION VALUE="royal" NEXT="#rc">    r c       </OPTION>
8    </INPUT>
9 </STEP>
```

INPUT Element : Type PHONE

The PHONE input is used to collect a telephone number from the user.[6]

Syntax

<INPUT TYPE="PHONE" NAME="*value*" NEXT="*value*"
[NEXTMETHOD="*value*"] [TIMEOUT="*value*"] />

Attributes

Attribute Name	Allowed Values
NAME	*identifier*
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST
TIMEOUT	*milliseconds (integer)*

Examples

```
1 <STEP NAME="phone">
2     <PROMPT> What is your phone number? </PROMPT>
3     <INPUT TYPE="PHONE" NAME="ph" NEXT="#fax"/>
4 </STEP>
```

In this simple example, the code on line 3 illustrates the use of the PHONE input type to collect a telephone number from the user, store the number in the VoxML variable named “ph”, and go to the STEP named “fax”.

Data Format

The PHONE input makes use of an input grammar to interpret the user’s response and store that response in a standard format. The phone number is interpreted as a string of digits, and stored in the VoxML variable as such.

If a user said “One, eight zero zero, seven five nine, eight eight eight eight”, the response would be stored as “18007598888”. Note that there is no punctuation added to the digits.

[6] The notion of “phone number” in the built-in input grammar for PHONE can vary from locale to locale.

INPUT Element : Type PROFILE

The INPUT PROFILE element is used to collect a piece of the user's profile information (e.g., name, home address, or phone number) from the platform's subscriber information database. This information is placed into a VoxML variable. This saves the user a lot of tedious and error-prone voice input.

Profile information is necessarily very platform-dependent, so to limit differences between VoxML implementations, we strongly recommend the use of the "Vcard" standard for personal information interchange. See Appendix B for details on "Vcard", and a list of its valid profile variable names and subtypes.

Because the INPUT PROFILE does not obtain its input from the interactive user, it is like INPUT HIDDEN, and can be combined with many INPUT HIDDENs and one interactive INPUT in the same STEP.

To protect personal information, any VoxML voice browser supporting INPUT PROFILE should institute some form of authorization checking. One possibility is to ask the user each session for permission to release this data. If the user does not give such permission, the INPUT PROFILE statement always generates a "noauth" error so that the VoxML dialog can take alternate action.

Syntax

<INPUT TYPE="PROFILE" NAME="*value*" PROFNAME="*value*" [SUBTYPE="*value*"] />

Attributes

Attribute Name	Allowed Values
NAME	*identifier*
PROFNAME	*profile element name (string)*
SUBTYPE	*profile element subtype (string)*

Examples

```
1 <STEP NAME="getinfo">
2     <INPUT TYPE="PROFILE" NAME="firstname"
3         PROFNAME="N" SUBTYPE="first"/>
4     <PROMPT> Hello, <VALUE NAME="firstname"/>.
5         Please say your pin. </PROMPT>
6     <INPUT TYPE="DIGITS" NAME="pin" NEXT="#verify"/>
7 </STEP>
```

On lines 2 and 3, the PROFILE input is used to retrieve the user's first name and store the string in the VoxML variable named "firstname". The string containing the name is then inserted into the PROMPT on line 4 using the VALUE element.

INPUT Element : Type RECORD

The RECORD input type is used to record an audio sample and to store that sample in a location specified by the application developer. The recording begins when the user starts talking, and ends when the user stops talking. The TIMEOUT attribute may be used to limit the length of the recording.

Syntax

<INPUT TYPE="RECORD" TIMEOUT="*value*" STORAGE="*value*" [FORMAT="*value*"] [NAME="*value*"] NEXT="*value*" [NEXTMETHOD="*value*"] />

Attributes

Attribute Name	Allowed Values
TIMEOUT	*maximum time to allow user to record, in milliseconds (integer)*
FORMAT	*MIME type for recorded audio (defaults to audio/wav). The exact format (i.e. sample rate, encoding) depends on the voice browser implementation.*
NAME	*identifier*
STORAGE	FILE REQUEST
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST

Examples

```
1 <STEP NAME="init">
2     <PROMPT> Please say your first and last name.
3         </PROMPT>
4     <INPUT TYPE="RECORD" TIMEOUT="7000"
5         NAME="samp" STORAGE="REQUEST"
6         NEXT="http://wavhost/acceptwav.asp"
7         NEXTMETHOD="POST"/>
8 </STEP>
```

In this first example, the RECORD input type is used to record a 7 second (maximum) audio sample and store it in the variable “samp”. The (encoded) binary contents of the recording are assigned to the variable, so it is important to use a NEXTMETHOD of “POST” whenever audio samples appear as the values of VoxML variables.

```
1 <STEP NAME="init">
2     <PROMPT> Please say your first and last name.
3         </PROMPT>
4     <INPUT TYPE="RECORD" TIMEOUT="7000"
5         NAME="theName" STORAGE="FILE"
6         NEXT="#reccomplete" NEXTMETHOD="GET"/>
7 </STEP>
```

In the second example, the RECORD input type is used to record a another 7 second audio sample, but this time the sample is stored in a file, instead of sent in the HTTP request as it was in the earlier example. The name of the file is chosen by the voice browser automatically and is stored in the VoxML variable named “theName”. After storing the audio sample in the file, the voice browser will continue execution at the URL specified by the NEXT attribute.

Note that in contrast to the earlier example, the value of the VoxML variable “theName” will be the name of the audio file. In the earlier example (where the audio sample was transmitted via the HTTP request), the value of the VoxML variable “theName” would be a many-kilobyte-long array of binary values.

INPUT Element : Type TIME

The TIME input type is used to collect a time of day from the user.

Syntax

<INPUT TYPE="TIME" NAME="*value*" NEXT="*value*" [NEXTMETHOD="*value*"] [TIMEOUT="*value*"] />

Attributes

Attribute Name	Allowed Values
NAME	*identifier*
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST
TIMEOUT	*milliseconds (integer)*

Examples

```
1 <STEP NAME="init">
2     <PROMPT> What time would you like your wakeup
3         call? </PROMPT>
4     <INPUT TYPE="TIME" NAME="wakeup" NEXT="#record"/>
5 </STEP>
```

This example makes use of the TIME input type to collect a time of day from the user, store that data in the VoxML variable named “wakeup”, and then go to the STEP named “record”.

Data Format

The TIME input makes use of an input grammar to interpret the user’s response and store that response in a standard format. This grammar interprets responses of various forms, including both 12-hour and 24-hour conventions.

“Four o’clock” is stored as “400”. Since the user was not specific as to the morning or evening, no indication is stored in the VoxML variable.

“Four oh three PM” becomes “403P”. Note the “P” appended to the time. Likewise, “Ten fifteen in the morning” becomes “1015A”. Note the “A”. “Noon” is stored as “1200P”, and “Midnight” is stored as “1200A”. Military time, such as, “Thirteen hundred hours” becomes “100P”.

INPUT Element : Type YORN

The YORN input is used to collect "yes or no" responses from the user. This input maps a variety of affirmative and negative responses to the values "Y" and "N", simplifying the work of the application developer in interpreting this type of user response.

Syntax

<INPUT TYPE="YORN" NAME="*value*" [TIMEOUT="*value*"]
NEXT="*value*" [NEXTMETHOD="*value*"] />
or
<INPUT TYPE="YORN" [NAME="*value*"] [TIMEOUT="*value*"]
[NEXT="*value*" [NEXTMETHOD="*value*"]] >
CASE elements
</INPUT>

Attributes

Attribute Name	Allowed Values
NAME	*identifier*
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST
TIMEOUT	*milliseconds (integer)*

Examples

```
1 <STEP NAME="ask">
2     <PROMPT> Fire the missles now? </PROMPT>
3     <INPUT TYPE="YORN" NAME="fire" NEXT="#confirm"/>
4 </STEP>
```

In this example, we use the YORN input type to collect a "yes or no" response from the user, store that response into the VoxML variable named "fire", and then go to the STEP named "confirm".

The YORN input type stores the value "Y" for affirmative responses and the value "N" for negative responses. Affirmative and negative responses are determined using an input grammar that maps various user responses to the appropriate result.

OPTION Element

The OPTION element is used to define the application behavior associated with a specific user response.

The OPTION element can exist only within the INPUT element, and then only when using the OPTIONLIST input type.

Syntax

<OPTION [NEXT="*value*" [NEXTMETHOD="*value*"]] [VALUE="*value*"] >
text
</OPTION>

Attributes

Attribute Name	Allowed Values
VALUE	*literal value*
NEXT	*URL of next step or dialog.*
NEXTMETHOD	GET (default) POST

Examples

```
1 <INPUT NAME="choice" TYPE="OPTIONLIST">
2     <OPTION NEXT="#doit" VALUE="1"> one </OPTION>
3     <OPTION NEXT="#doit" VALUE="2"> two </OPTION>
4 </INPUT>
```

The code on lines 2 and 3 illustrate the use of the OPTION element within the INPUT element. In this example, the OPTION on line 2 would be executed when the user responded with “one”, and the OPTION on line 3 would be executed when the user responded with “two”. If the user said “one” the result would be that the value of the variable named “choice” would be “1”, because of the use of the VALUE attribute. Because the NEXT attributes for both of the OPTIONs in this OPTIONLIST are the same, the VoxML voice browser would proceed to the STEP named “doit” when either “one” or “two” was recognized.

```
1 <INPUT TYPE="OPTIONLIST">
2     <OPTION NEXT="http://localhost/vml/weather.asp">
3         weather </OPTION>
4     <OPTION NEXT="http://localhost/vml/news.asp">
5         news </OPTION>
6     <OPTION NEXT="http://localhost/vml/traffic.asp">
7         traffic </OPTION>
8 </INPUT>
```

The code shown above illustrates the use of the OPTION element to select one of three VoxML applications. Note that the URLs used in the NEXT attributes are full HTTP URLs, and that unlike the previous example, each OPTION has a unique NEXT attribute.

OPTIONS Element

The OPTIONS element describes the type of interactive input expected within a given STEP element. The OPTIONS element is typically used in HELP elements to present the user with a complete list of valid responses. The OPTIONS element can be used anywhere that text is read to the user.

The OPTIONS element can be contained by a PROMPT, EMP, PROS, HELP, ERROR, or ACK element.

Syntax

<OPTIONS/>

Attributes

The OPTIONS element has no attributes.

Examples

```
1 <CLASS NAME="helpful">
2     <HELP> Your choices are: <OPTIONS/> </HELP>
3 </CLASS>
```

This example illustrates how the OPTIONS element can be used to construct a "helpful" CLASS. Any STEPs that directly or indirectly name "helpful" as a PARENT respond to "help" by speaking the message, in which the OPTIONS element expands to a description of what can be said at this point in the dialog.

OR Element

The OR element is used to define alternate recognition results in an OPTION element. The OR element is interpreted as a logical "or", and is used to associate multiple recognition results with a single NEXT attribute.

The OR element can exist only within the OPTION element.

Syntax

<OR/>

Attributes

The OR element has no attributes.

Examples

```
<INPUT TYPE="OPTIONLIST">
    <OPTION NEXT="#coke_chosen">
        coke <OR/> coca-cola
    </OPTION>
    <OPTION NEXT="#pepsi_chosen"> pepsi </OPTION>
</INPUT>
```

The code shown above illustrates the use of the OR element within an OPTION element. As you can see on line 3, the user may respond with either "coke" or "coca-cola", and the effect is the same — the voice browser will proceed to the STEP named "coke_chosen".

PROMPT Element

The PROMPT element is used to define the content that is to be presented to a user in a STEP. This content consists of intermixed text and audio elements. The text is read via text-to-speech technology, and you can control this reading using BREAK elements to interject periods of silence, EMP elements to change the vocal emphasis, and PROS elements to vary several dimensions of prosody.[7] The AUDIO elements specify sound files to play to the user; these are used in applications that require concatenated speech, sound effects, and samples recorded from users. (Note that this full generality of marked up text and audio clips is available inside *any* VoxML element that outputs information to the user.)

The PROMPT element can be contained within a STEP or a CLASS element.

Syntax

<PROMPT> *text* </PROMPT>

Attributes

The PROMPT element has no attributes.

Examples

```
1 <STEP NAME="init">
2  <PROMPT> How old are you? </PROMPT>
3  <INPUT TYPE="NUMBER" NAME="age" NEXT="#weight"/>
4 </STEP>
```

In this simple example, the text "How old are you?" will be spoken via text-to-speech technology, and then the application will wait for the user to say his age. The PROMPT element on line 2 defines what will be presented to the user.

[7] Different TTS engines adhere to EMP and PROS markups with different degrees of fidelity.

PROS Element

The PROS (prosody) element controls the prosody of the content presented to the user via PROMPT, HELP, ERROR, CANCEL, and ACK elements. Prosody affects certain qualities of the text-to-speech presentation, including rate of speech, pitch, range, and volume. The precise effect depends on the capabilities of the TTS engine used.

The PROS element can be contained within a PROMPT, EMP, PROS, HELP, ERROR, CANCEL, or ACK element.

Many styles of speech markup exist. The VoxML style is based closely on the Java Speech Markup Language (JSML) being considered a standard by the Sable Consortium. This is an XML-style markup, and includes BREAK, EMP, and PROS elements.

See http://java.sun.com/products/java-media/speech for details on JSML.

Syntax

<PROS [RATE="*value*"] [VOL="*value*"] [PITCH="*value*"] [RANGE="*value*"] >
    text
</PROS>

Attributes

Attribute Name	Allowed Values
RATE	*Speaking rate in words per minute (integer). A value of 135 is normal, and 300 very fast.*
VOL	*Volume of speech (integer). 1.0 is maximum, 0.0 is silence.*
PITCH	*Baseline pitch of speaker in Hz (integer). A female voice usually has a baseline pitch of 140-280 Hz. Male voices are 70-140 Hz.*
RANGE	*Pitch range of speaker in Hz (integer). Females have a range of 80 Hz or more, males 40-80 Hz. Setting the range to 0 Hz gives you a complete monotone.*

Examples

```
1 <PROMPT> Let me tell you a secret:
2      <PROS VOL="0.5"> I ate the apple. </PROS>
3 </PROMPT>
```

In this example, “I ate the apple.” is spoken at one half the normal volume.

Setting Prosody Values

The various PROS attribute values can be specified in four ways:

- As an absolute value, e.g., RATE="100" to speak at a somewhat slower rate than normal.
- As a relative change in value, e.g., RATE="-50" speaks 50 words less per minute than the current speed, and VOL="+.25" doubles the current volume if it is currently .25.
- As a relative percentage change in value. RATE="+50%" increases the speed fifty percent from the current speed, while VOL="+100%" doubles the current volume.
- As a reset to default value. RATE="RESET" resets the speed to the default as defined by the particular VoxML voice browser implementation.

Note that although VoxML voice browser implementations must attempt to support as much of JSML speech markups as possible, they determine what the default values are and may also impose limits on the range of allowable values. Moreover, different text-to-speech systems have varying capabilities, and rarely support the full set of markups.

For instance, the VoxML voice browser used in the VoxML 1.1 Software Development Kit has these default values, and imposes these safety restrictions on *relative* (but not absolute) changes:

Attribute	Default (RESET)	Minimum (relative changes)	Maximum (relative changes)
RATE	135	50	250
VOL	1.00	0.00	1.00
PITCH	50	50	400
RANGE	50	none	none

RENAME Element

If you have a custom grammar, the results of each recognition are returned in named slots. By default, the INPUT GRAMMAR puts the slot values into VoxML variables having the same name. But this means you could not use the same grammar to gather several chunks of information. So, in VoxML this problem is avoided with the RENAME element. This specifies the mapping between the recognition slot names and the variables to store the results in.

Another use of RENAME is when the recognition slot names are uninformative or misleading, and you are unable to change the grammar.

The RENAME element can exist only within the INPUT element, and then only when using the GRAMMAR input type.

Syntax

<RENAME RECNAME="*value*" VARNAME="*value*" />

Attributes

Attribute Name	Allowed Values
VARNAME	*identifier*
RECNAME	*identifier*

Examples

```
1 <INPUT TYPE="GRAMMAR"
2    SRC="http://www.foo.com/mygram.grm"
3    NEXT="http://www.fancyquotes.com/vmlstocks.asp">
4    <RENAME VARNAME="sym"    RECNAME="symbol">
5    <RENAME VARNAME="detail" RECNAME="quotetype">
6 </INPUT>
```

In this example, the RENAME element is used to account for differences in the variable names collected from a grammar and those expected by another script. In particular, the case imagined here is one where a grammar from foo.com is used to provide input to a VoxML application hosted by fancyquotes.com. Because the grammar and script are imagined to have been developed independently, the RENAME element is used to help connect the grammar and the stock-quoting application.

RESPONSE Element

The RESPONSE element is used to define the behavior of the VoxML application in response to different combinations of recognition slots. The RESPONSE element enables the application developer to define a different NEXT attribute depending on which of the grammar's slots were filled.

The RESPONSE element can exist only within an INPUT element, and then only when using an input type of GRAMMAR.

Syntax

<RESPONSE FIELDS="*value*" [NEXT="*value*" [NEXTMETHOD="*value*"]] />
… or …
<RESPONSE FIELDS="*value*" [NEXT="*value*" [NEXTMETHOD="*value*"]] >
SWITCH elements
</RESPONSE>

Attributes

Attribute Name	Allowed Values
FIELDS	*A comma separated list of recognition slot names.*
NEXT	*The next step's URL.*
NEXTMETHOD	GET (default) POST

Examples

```
1 <INPUT TYPE="GRAMMAR"
2    SRC="gram://.Banking/action/amt/fromacct/toacct"
3    NEXT="#notenoughfields">
4    <RESPONSE FIELDS="action,amt,fromacct,toacct"
5       NEXT="#doit"/>
6    <RESPONSE FIELDS="action,amt,fromacct"
7       NEXT="#asktoacct"/>
8    <RESPONSE FIELDS="action,amt,toacct"
9       NEXT="#askfromacct"/>
10   <RESPONSE FIELDS="action,amt" NEXT="#askaccts"/>
11   <RESPONSE FIELDS="action" NEXT="#askamtaccts"/>
12 </INPUT>
```

This example illustrates how the RESPONSE element can be used to deal with situations where the user specifies less than all the possible variables available in the grammar. Using the RESPONSE element, the application can arrange to collect only the information not already filled in by prior steps.

In particular this example transfers to the "askaccts" STEP if neither the source nor destination account is specified (for example, the user said "transfer 500 dollars"), but it transfers to the "askfromacct" STEP if the user said what account to transfer to,

but did not specify a source account (for example, if the user had said “transfer 100 dollars to savings”).

The NEXT URL on the INPUT element is used when the user’s response does not match any of the defined RESPONSEs.

STEP Element

The STEP element defines a state in a VoxML application. A STEP element typically has an associated PROMPT element and INPUT element that minimally define the application state.

Each VoxML DIALOG must contain exactly one STEP element named “init”. The execution of the VoxML application begins with the STEP named “init”.

To exit a dialog, don’t specify a NEXT attribute in the last STEP, or specify one with the dummy “#end” STEP. The latter is preferred for clarity.

The STEP element can be contained by a DIALOG element only.

Syntax

<STEP NAME="*value*" [PARENT="*value*"] [BARGEIN="*value*"] [COST="*value*"] >
VoxML
</STEP>

Attributes

Attribute Name	Allowed Values
NAME	*identifier*
PARENT	*identifier*
BARGEIN	Y (default) N
COST	integer[8]

Examples

```
1 <STEP NAME="askpython" PARENT="tvrating">
2    <PROMPT> Please rate Monty Python's Flying Circus
3         on a scale of 1 to 10. </PROMPT>
4    <INPUT NAME="python" TYPE=“NUMBER” NEXT="#drwho" />
5 </STEP>
```

This example illustrates a simple STEP that collects the user’s opinion on one of several public television shows. The step uses the PARENT attribute to share a common set of help and error elements with other TV-show-rating STEPs. For example, the parent class might contain a help element explaining what a rating of 1, 5, and 10 would mean; a common error message might remind the user that a numeric rating is expected.

[8] COST is an implementation-dependent amount to charge for entering (or completing) this step. VoxML attaches no semantics to this attribute.

SWITCH Element

The SWITCH element is used to define the application behavior dependant on the value of a specified recognition slot. The SWITCH element is used only in conjunction with the CASE element.

The SWITCH element can exist only within the INPUT element, and then only when using the GRAMMAR input type.

Syntax

<SWITCH FIELD="*value*">
VoxML
</SWITCH>

Attributes

Attribute Name	Allowed Values
FIELD	*identifier*

Examples

```
1  <INPUT TYPE="GRAMMAR"
2     SRC="gram://.Banking/action/amount/fromacct/toacct">
3     <SWITCH FIELD="action">
4        <CASE VALUE="transfer" NEXT="#transfer" />
5        <CASE VALUE="balance" NEXT="#balance" />
6        <CASE VALUE="activity">
7           <SWITCH FIELD="fromacct">
8              <CASE VALUE="checking" NEXT="#chxact" />
9              <CASE VALUE="savings" NEXT="#savact" />
10          </SWITCH>
11       </CASE>
12    </SWITCH>
13 </INPUT>
```

This example shows how a SWITCH element might be used to determine the next step to execute in response to a banking request. In this example, the grammar may fill in some or all of the variables “action”, “amount”, “fromacct”, and “toacct”. If the user asks for a transfer or balance action, the next step to execute is the transfer or balance step. If the user asks for a report of account activity, a second SWITCH element determines the next step based on the account type for which a report is being requested (assumed to be available in the “fromacct” variable).

VALUE Element

The VALUE element is used to present the value of a VoxML variable to the user via text-to-speech. The VALUE element can be used anywhere that text is read to the user.

The VALUE element can be contained by a PROMPT, EMP, PROS, HELP, ERROR, CANCEL, or ACK element.

Syntax

<VALUE NAME="*value*" />

Attributes

Attribute Name	Allowed Values
NAME	*identifier*

Examples

```
1 <STEP NAME="thanks">
2     <PROMPT> Thanks for your responses. I'll record
3         that <VALUE NAME="first"/> is your favorite
4         and that <VALUE NAME="second"/> is your
5         second choice.
6     </PROMPT>
7     <INPUT TYPE="NONE" NEXT="/recordresults.asp" />
8 </STEP>
```

The VoxML code shown above illustrates the use of the VALUE element to read the user's selections back to the user. On line 3, the value of the VoxML variable named "first" would be inserted into the PROMPT, and on line 4 the value of the VoxML variable named "second" would be inserted into the PROMPT.

3. Appendices

Appendix A - VoxML Document Type Definition

This appendix describes the VoxML Document Type Definition. A DTD is used to define the syntax and grammar of a language in a way that can be read and understood by humans as well as machines (i.e. parsers).

```
<!ELEMENT DIALOG (STEP|CLASS)*>
<!ATTLIST DIALOG BARGEIN (Y|N) "Y">

<!ELEMENT STEP (PROMPT|INPUT|HELP|ERROR|CANCEL|ACK)*>
<!ATTLIST STEP NAME ID #REQUIRED
               PARENT IDREF #IMPLIED
               COST CDATA #IMPLIED
               BARGEIN (Y|N) "Y”>

<!ELEMENT CLASS (PROMPT|HELP|ERROR|CANCEL|ACK)*>
<!ATTLIST CLASS NAME ID #REQUIRED
                PARENT IDREF #IMPLIED
                COST CDATA #IMPLIED
                BARGEIN (Y|N) "Y”>

<!ELEMENT PROMPT (#PCDATA|OPTIONS|VALUE|EMP|BREAK|PROS|AUDIO)*>

<!ELEMENT EMP (#PCDATA|OPTIONS|VALUE|EMP|BREAK|PROS|AUDIO)*>
<!ATTLIST EMP LEVEL (STRONG|MODERATE|NONE|REDUCED) "MODERATE">

<!ELEMENT PROS (#PCDATA|OPTIONS|VALUE|EMP|BREAK|PROS|AUDIO)*>
<!ATTLIST PROS RATE CDATA #IMPLIED
               VOL CDATA #IMPLIED
               PITCH CDATA #IMPLIED
               RANGE CDATA #IMPLIED>

<!ELEMENT HELP (#PCDATA|OPTIONS|VALUE|EMP|BREAK|PROS|AUDIO)*>
<!ATTLIST HELP ORDINAL CDATA #IMPLIED
               REPROMPT (Y|N) "N"
               NEXT CDATA #IMPLIED
               NEXTMETHOD (GET|POST) "GET">

<!ELEMENT ERROR (#PCDATA|OPTIONS|VALUE|EMP|BREAK|PROS|AUDIO)*>
<!ATTLIST ERROR TYPE NMTOKEN "ALL"
                ORDINAL CDATA #IMPLIED
                REPROMPT (Y|N) "N"
                NEXT CDATA #IMPLIED
                NEXTMETHOD (GET|POST) "GET">

<!ELEMENT CANCEL (#PCDATA|VALUE|EMP|BREAK|PROS|AUDIO)*>
<!ATTLIST CANCEL NEXT CDATA #REQUIRED
                 NEXTMETHOD (GET|POST) "GET">
```

```
<!ELEMENT AUDIO EMPTY>
<!ATTLIST AUDIO SRC CDATA #REQUIRED>

<!ELEMENT ACK (#PCDATA|OPTIONS|VALUE|EMP|BREAK|PROS|AUDIO)*>
<!ATTLIST ACK CONFIRM NMTOKEN "YORN"
              BACKGROUND (Y|N) "N"
              REPROMPT (Y|N) "N">

<!ELEMENT INPUT (OPTION|RESPONSE|RENAME|SWITCH|CASE)*>
<!ATTLIST INPUT TYPE (NONE|OPTIONLIST|RECORD|GRAMMAR
                      |PROFILE|HIDDEN|YORN|DIGITS
                      |NUMBER|TIME|DATE
                      |MONEY|PHONE) #REQUIRED
                NAME NMTOKEN #IMPLIED
                NEXT CDATA #IMPLIED
                NEXTMETHOD (GET|POST) "GET"
                TIMEOUT CDATA #IMPLIED
                MIN CDATA #IMPLIED
                MAX CDATA #IMPLIED
                PROFNAME NMTOKEN #IMPLIED
                SUBTYPE NMTOKEN #IMPLIED
                SRC CDATA #IMPLIED
                VALUE CDATA #IMPLIED
                MSECS CDATA #IMPLIED
                STORAGE (FILE|REQUEST) #IMPLIED
                FORMAT CDATA #IMPLIED>

<!ELEMENT SWITCH (CASE|SWITCH)*>
<!ATTLIST SWITCH FIELD NMTOKEN #REQUIRED>

<!ELEMENT RESPONSE (SWITCH)*>
<!ATTLIST RESPONSE NEXT CDATA #IMPLIED
                   NEXTMETHOD (GET|POST) "GET"
                   FIELDS CDATA #REQUIRED>

<!ELEMENT RENAME EMPTY>
<!ATTLIST RENAME VARNAME NMTOKEN #REQUIRED
                 RECNAME NMTOKEN #REQUIRED>

<!ELEMENT CASE EMPTY>
<!ATTLIST CASE VALUE CDATA #REQUIRED
               NEXT CDATA #REQUIRED
               NEXTMETHOD (GET|POST) "GET">

<!ELEMENT VALUE EMPTY>
<!ATTLIST VALUE NAME NMTOKEN #REQUIRED>

<!ELEMENT BREAK EMPTY>
<!ATTLIST BREAK MSECS CDATA #IMPLIED
                SIZE (NONE|SMALL|MEDIUM|LARGE) "MEDIUM">

<!ELEMENT OPTIONS EMPTY>

<!ELEMENT OR EMPTY>

<!ELEMENT OPTION (#PCDATA|VALUE|OR)*>
<!ATTLIST OPTION VALUE CDATA #IMPLIED
                 NEXT CDATA #IMPLIED
                 NEXTMETHOD (GET|POST) "GET">
```

Appendix B - Vcard Profile Names and Subtypes

VoxML voice browsers should implement INPUT PROFILE in terms of the Vcard specification whenever possible. This appendix describes the valid combinations of Vcard profile names and their associated subtypes. For more information regarding the Vcard specification, see "http://www.imc.org/rfc2426".

All information described below is provided based on the profile information of the VoxML voice browser's current interactive user. Although not all browser implementations will store the user's profile information, this schema represents a convenient way to access the data when it is available.

Profile Name	Subtype	Description
ADR	HOME WORK	Home address. Work address.
BDAY	none	Birthday.
EMAIL	none NOTIFICATION	Primary email address. Notification email address.
FN	none	Formatted name.
GEO	none	Geographic location (longitude;latitude).
KEY	none	Public encryption key.
LABEL	HOME WORK	Home mailing label. Work mailing label.
MAILER	none	Email program used.
N	FIRST LAST MIDDLE PREFIX SUFFIX	First name. Last name. Middle name. Prefix (e.g., Mr., Mrs., Dr.). Suffix (e.g., Jr., Ph.D., M.D.).
ORG	none	Organization name.
ROLE	none	Job role or position.
TEL	HOME WORK FAX CELL PREF	Home telephone number. Work telephone number. Fax call telephone number. Cellular telephone number. Preferred telephone number.
TITLE	none	Job title.
TZ	none	Time zone.
UID	none	Globally unique id.
URL	none	URL of home page.