



"Offering **integrated**
Voice,
Web
& Mobile
Solutions **&**
Services
which
make
communication
more **effective**
& more
efficient"

SpeechFrame[©] VoiceXML manual



This document is the copyright of
Comsys Telecom & Media B.V.
All rights reserved.

© Copyright

Proprietary information of Comsys Telecom & Media B.V. or its affiliates is contained herein. Any reproduction, use, appropriation, or disclosure of this information, in whole or in part, without the specific prior written authorisation of the owners thereof is strictly prohibited. Failure to observe this notice may result in legal proceedings or liability for resulting damage or loss.

Management Summary

Document management

Version	Date	Description
0.1	05-09-2008	First concept

Author

Luuk Engelen, Comsys Telecom & Media B.V.

Table of contents

1.	Introduction.....	7
1.1.	Aim and scope of this document	7
1.2.	Structure of this document	7
1.3.	VoiceXML	7
1.4.	Voice User Interface	7
1.5.	SpeechFrame VoiceXML Applications	8
2.	System Architecture.....	9
2.1.	VoiceXML architecture in general	9
2.2.	SpeechFrame architecture	10
3.	VoiceXML Application	11
3.1.	Basic principles and elements	11
3.2.	Structure of a VoiceXML application	11
3.3.	Root document	11
3.3.1.	Scripts	12
3.3.2.	Default platform properties	12
3.3.3.	Scoping rules	13
3.3.4.	Global variables	13
3.3.5.	Default error handling	14
3.3.6.	Finishing the dialogue	14
4.	Overview of VoiceXML elements	15
4.1.	Assign	15
4.1.1.	attributes	15
4.2.	Audio	16
4.2.1.	Comsys enhancements to the <audio> tag	16
4.2.2.	Attributes	16
4.3.	Block	17
4.3.1.	Attributes	17
4.4.	Catch	18
4.4.1.	Attributes	18
4.5.	Choice	19
4.5.1.	Attributes	19
4.6.	Clear	20
4.6.1.	Attributes	20
4.7.	Disconnect	21
4.7.1.	Attributes	21
4.8.	Else, elseif	22
4.8.1.	Attributes	22
4.9.	Error	23
4.9.1.	Attributes	23
4.10.	Exit	24
4.10.1.	Attributes	24
4.11.	Field	25

4.11.1.	Attributes	25
4.12.	Filled	26
4.12.1.	Attributes	26
4.13.	Form	27
4.13.1.	Attributes	27
4.14.	Goto	28
4.14.1.	Attributes	29
4.15.	Grammar	30
4.15.1.	Attributes	31
4.16.	If	32
4.17.	Log	33
4.17.1.	Attributes	33
4.18.	Menu	34
4.18.1.	Attributes	35
4.19.	Noinput	36
4.19.1.	Attributes	36
4.20.	Nomatch	37
4.20.1.	Attributes	37
4.21.	Option	38
4.21.1.	Attributes	38
4.22.	Param	39
4.22.1.	Attributes	40
4.23.	Prompt	41
4.23.1.	Attributes	41
4.24.	Property	42
4.24.1.	Attributes	42
4.25.	Record	43
4.25.1.	Attributes	44
4.26.	Reprompt	45
4.26.1.	Attributes	45
4.27.	Return	46
4.27.1.	Attributes	46
4.28.	Script	47
4.28.1.	Attributes	47
4.29.	Subdialog	49
4.29.1.	Attributes	50
4.30.	Submit	51
4.30.1.	Attributes	51
4.31.	Throw	52
4.31.1.	Attributes	52
4.32.	Transfer	53
4.32.1.	Comsys enhancements on the transfer	53
4.32.1.1.	Enhanced caller- and called party info	53
4.32.1.2.	Play a prompt to the called party	54
4.32.1.3.	conferencing	54
4.32.2.	Attributes	54
4.33.	Value	56

4.33.1.	Attributes	56
4.34.	Var	57
4.34.1.	Attributes	57
4.35.	VXML	58
4.35.1.	Attributes	58
5.	Conferencing	59
5.1.	conference address	59
5.2.	A simple conference	59
5.3.	A complex conference	59
5.4.	Example	61
5.4.1.	Application listing	61
5.4.2.	Root document listing	62
6.	Pre-call Announcements.....	63
6.1.	Sending a specific clearing cause	64
7.	Audio prompting	65
7.1.	Audio prompting	65
7.1.1.	Concatenation	65
7.1.2.	Structure of the prompts directory	66

1. Introduction

1.1. Aim and scope of this document

This document is a manual for developers of voice applications on the SpeechFrame[®] platform. It serves as a guide for writing, building, changing and testing of VoiceXML applications on SpeechFrame.

SpeechFrame applications use VoiceXML, style sheets, JAVA, Java Server Pages and other programming languages and tools. In this manual, the referenced terms and methods from these languages will be presented solely from the point of view of their deployment in SpeechFrame applications. With the exception of our treatment of all relevant VoiceXML elements in Chapter 5 of this document, no detailed explanation will be given here; we assume knowledge of these elements.

1.2. Structure of this document

Following this introductory chapter, Chapter 2 presents the architecture of VoiceXML applications in general and the architecture of the SpeechFrame platform in particular. Chapter 3 describes how to develop new SpeechFrame applications on the basis of SpeechFrame's Dialogue Template. Chapters 4 and 5 deal with the relevant elements of VoiceXML and their use in SpeechFrame applications. In Chapter 6 audio prompting in SpeechFrame applications is explained, as well as scripting which is used in combination with SpeechFrame's systematic prompt directory structure for automatic concatenation of compound prompts such as numerals, phone numbers etc. Finally in Chapter 7 dynamic generation of application documents is presented, using the Comsys utilities package. The use of Java server pages and style sheets in SpeechFrame applications is illustrated by complete (i.e. fully functional) examples.

1.3. VoiceXML

SpeechFrame applications are written in VoiceXML. VoiceXML is a XML-based mark-up language for distributed voice applications, much as HTML is a language for distributed visual applications. Originally a joint initiative of AT&T, IBM, Lucent Technologies and Motorola. For detailed specifications, explanation and code samples we refer to: <http://www.w3c.org/TR/VoiceXML20>. The basic principles of VoiceXML and all relevant elements of VoiceXML for application development on SpeechFrame are discussed in Chapters 4 and 5.

1.4. Voice User Interface

SpeechFrame is a so-called *voice browser*: it services interaction with the user by means of voice, or more specifically, by automated telephone conversations.

Like (X)HTML, VoiceXML follows the 'question&answer' format known from browsing the net: following a click on a link in a web browser, the browser sends a request to a web server for a new web page. Systems like SpeechFrame work in the same way, hence the name *voice browser*.

With SpeechFrame, VoiceXML and other languages such as JSP and Java, existing web applications can be easily converted into voice applications: existing database and back office interfaces can be reused, only the front end, the *voice user interface* needs to be designed. As opposed to the graphic - visual - interface of a web browser, a voice user interface is auditive in nature. This means that even if the questions and answers are the same, the dialogue will need some restructuring. It is obvious that a telephone conversation has different rules and requirements compared to, for example, an interactive web search.

1.5. SpeechFrame VoiceXML Applications

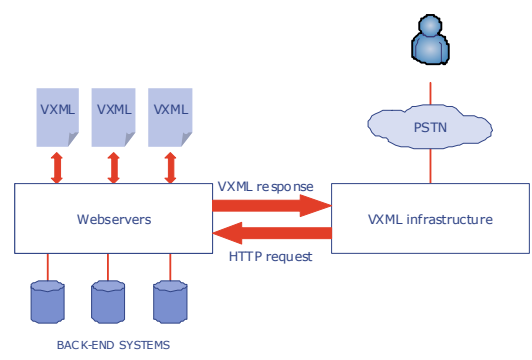
A SpeechFrame application is an implementation of a dialogue with a caller. All applications are developed for one specific purpose (or customer), but reuse of existing standard SpeechFrame *modules* makes the development quick, easy and cost effective.

For developers, creating new applications is made easy thanks to the SpeechFrame Dialogue Template, which will be presented in Chapter 3.

2. System Architecture

2.1. VoiceXML architecture in general

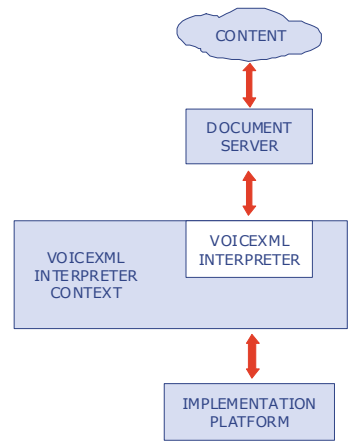
The picture below shows how a VoiceXML-dialogue works.



The dialogue starts with a call (usually inbound to a VXML-enabled IVR), switched by PSTN. The VXML infrastructure functions as a browser: it starts by sending a predefined http request to a traditional web server to fetch documents and other resources: VXML documents, audio, grammars, scripts. After receiving the document (some extra exchange with back-end systems may be involved), the VXML interpreter within the VXML infrastructure starts executing the application, thereby entering in a conversation with the caller. During the dialogue, more exchanges with the webserver can take place: based on the collected user information, a new URI is resolved by the VXML interpreter and a http request is sent there. The server responds by sending a new VXML document.

Software and resources for the execution and interpretation of the VXML application, such as ASR, TTS, ECMAScript processing etc, are part of the VXML infrastructure.

The next picture illustrates the integration of voice services with data services via VoiceXML. It is the translation of the VXML code into the appropriate actions of the browser that gives meaning to a VXML document.



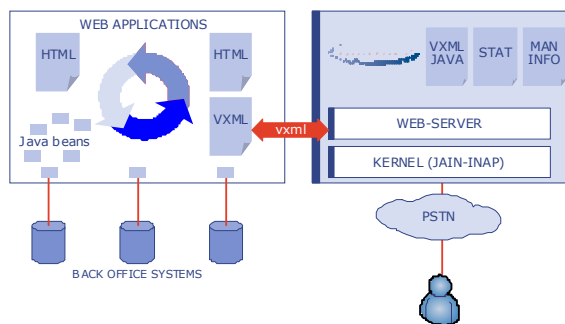
The VXML interpreter and the VXML interpreter context both control the implementation platform. The implementation platform generates events in response to user actions on the one hand (such as either spoken or

DTMF input, hanging up the phone) and system events on the other (such as time-out). To some of these events the interpreter reacts according to the VXML document, to others the interpreter context reacts. Database connections are handled by the document server. Allocation of resources is handled by the implementation platform.

2.2. SpeechFrame architecture

The next picture shows how the general principles explained above are implemented in the SpeechFrame application architecture.

On the right: SpeechFrame itself, handles the phone calls switched to the IVR. The VXML service takes care of interpretation of VXML documents (generated by the webserver) and invoking and handling the appropriate



platform events. Managing (start/stop) and configuration of services is done by means of a management webgui. On the left the application part is shown. The application makes use of a servlet container, in which servlets are used for generation of web pages (VXML and HTML). All Java techniques are available through the servlets: Java beans, JSP's, JDBC etc. The application itself (including application specific management HTML pages), as well as the SpeechFrame management webgui, runs on a webserver. The webserver does the hosting and holds the application logic as well as the various utilities to facilitate database connections and dynamic generation of documents

3. VoiceXML Application

3.1. Basic principles and elements

A VoiceXML application is the implementation of a dialogue. VoiceXML defines the interactive session with a caller on the basis of *menus, forms, fields* and *transitions*.

Every VXML application contains either at least one *form*, each containing zero or more *fields*, or at least one *menu*. Forms typically consist of playing a prompt to the caller, listening for a response, which implies filling out one or more fields, and transitioning to the next form or document based on the result. A menu is a simplified version of a form. It defines in its choices the possible transitions.

Forms are not required to play audio or contain any fields, but they must explicitly transition to the next form or document. Failing to do so will result in an implicit exit of the document, thereby exiting the entire application. Document execution starts at the first form of that document and will proceed according to the specified transitions until no transition is specified or a `<disconnect/>` transition ends the call. The execution order is determined by the FIA, form interpretation algorithm (see specification at <http://www.w3c.org/TR/VoiceXML20>, section 2.1.6.).

In the following sections, all the relevant elements and aspects of VoiceXML will be discussed which developers need to know in order to write correct VXML documents for a SpeechFrame application.

3.2. Structure of a VoiceXML application

A VoiceXML application consists of one or more VoiceXML documents and possible resources, such as scripts, grammars and audio files, referenced in the VXML documents. A multi document application may have an application root document that is shared by all VXML documents belonging to the application. The application root document is a way for applications that contain multiple documents - especially dynamically generated ones - to easily share context and data between them. All VXML documents that share the same application root document (these are then called "supporting" documents) share a common "application scope" context for variables, grammars, scripts, and event handlers. If an application consists of a single document this document is also the root document of the application.

Some VXML documents can be generated dynamically by means of Java Server Pages (JSP's) and style sheets.

3.3. Root document

It is good practice to have an application root document for each application: `<application name>.VXML`. In each supporting VXML document the attribute `application = "<application name>.VXML"` is added to the `<VXML>` tag. This reference makes it possible for the browser to load the root document simultaneously when loading the supporting VXML document.

This will be:

```
<VXML version="2.0" application="app.VXML">
```

The root document holds elements that are shared through the whole application. Normally it contains scripts, properties, default event handling and global variables (these are not lost in transitions between documents). In the next sections an overview is given of a typical content of a root document:

3.3.1. Scripts

In order to invoke ECMA script functions in a VXML document the `<script>` element is used. The root document holds references to scripts which are defined in the `<application root>/scripts` directory.

Example:

```
<script src="scripts/system.js"/>
```

3.3.2. Default platform properties

The default platform properties are defined by means of the `<property>` element. The following defaults can be set for the entire application in the root document (if no defaults are set on the application scope-level, the platform-defined defaults apply):

```
<property name="bargein"           value="true"/>
<property name="timeout"          value="15s"/>
<property name="interdigittimeout" value="5s"/>
<property name="termtimeout"      value="5s"/>
<property name="delayedInboundConnect" value="true"/>
<property name="clearingCause"    value="NUMBER_UNOBTAINABLE"/>
```

The value of "bargein" indicates whether the prompts are interruptable: if the value is "true", the playing of the prompt is stopped when input is received during the playing of the prompt; the system then continues the dialogue according to the collected user input.

The value of "timeout" indicates the time, either in seconds or milliseconds, given to the user to react; when no user input is received after this period has elapsed, the noinput event is thrown.

The value of "interdigittimeout" indicates the maximum allowed time between DTMF digits. When this time is exceeded while no grammar has yet been matched, a nomatch event is thrown. If a grammar may be matched by a DTMF string of variable length and "interdigittimeout" is exceeded, the input collected so far is returned as a completed recognition result.

The value of "termtimeout" indicates the time in seconds for the user to enter the "termchar" indicating the input is complete (usually the "#" character). If the entry of the last DTMF results in a matched grammar and the termchar is non-empty, the user is allowed to enter an optional termchar DTMF. If the user fails to do so within termtimeout, the recognition ends and the recognised value is returned. If the termtimeout is 0s (the default), then the recognised value is returned immediately after the last DTMF allowed by the grammar, without waiting for the optional termchar. Note: the termtimeout applies only when no additional input is allowed by the grammar; otherwise, the interdigittimeout applies.

For an explanation of the delayedInboundConnect and the clearingCause properties, please see chapter 6, Pre-call Announcements.

The values of the defaults discussed above can be changed in the application root document or overruled in supporting VXML documents (the standard scoping rules apply - see next subsection) as required by the application.

3.3.3. Scoping rules

The VXML scoping rules for variables, event handlers and grammars follow the same standard as known from many languages. In short:

- each scope defines a new and unique context
- variables are inherited by reference
- event handlers, links, and grammars are inherited as if by copy
- contexts are re-initialised on re-entry

The scoping hierarchy the FIA follows is:

session

application (the sub-tree starting with the <VXML> element in the application root document) **document** (the sub-tree starting with the <VXML> element in a given document)

dialog (the sub-tree starting with a <form> or <menu> element)

anonymous (the sub-tree starting with a <block>, <filled> or <catch> element)

3.3.4. Global variables

Global variables should be declared here, but only assigned a value if they are not subject to change during the dialogue. Every time a new page is loaded with the same root, the variables are re-initialised, and the change is lost. Assigning a value can be done by means of the <assign> element or directly in the declaration.

Example:

```
<var name="company" expr="'comsys'"/>
```

`company` does not change in the example-application, so a value can be assigned to it in the declaration.

Not so for the language variable:

```
<var name="ivrLanguage"/>
```

The language can change during the dialogue. When an application uses prompts in different languages, a language switch during the dialogue is simply done by resetting the reference to the language specific audiopath when invoking the prompt.

3.3.5. Default error handling

Events that are dealt with in the same way in the entire application also belong in the root document. Typically this applies for the handlers of error, nomatch and noinput events.

`<noinput>`, `<nomatch>` and `<error>` are shorthand VXML notations for `<catch event="...">`.

Example:

```
<nomatch>
  <audio src="'prompts/std/nomatch.wav'"/>
  <reprompt/>
</nomatch>
```

Of course, other application specific event handlers can be defined here; this is done by means of the `<catch>` element.

3.3.6. Finishing the dialogue

To finish the dialogue after the call has ended, for example writing call statistic information in a database, *disconnect* events can be handled by means of `<catch>`. In case of a disconnect, either because the caller or the system hangs up, and in case of a transfer to another connection, a disconnect event is thrown. Older (VXML1.0) interpreters can still throw *telephone.disconnect* events (instead of *connection.disconnect*), it is therefore advisable to catch those, too.

Always finish the dialogue, in this case the disconnect catch, with the `<exit>` element. `<exit>` simply exits the application without throwing an event, while the `<disconnect>` element throws the disconnect event but does not necessarily exit. Developers can use this event for post-hangup processing by implementing the desired actions in a specific event handler, as illustrated here.

The following events can be thrown:

```
connection.disconnect.hangup
connection.disconnect.transfer
telephone.disconnect.hangup
telephone.disconnect.transfer
```

Examples of handlers:

```
<catch event="telephone.disconnect.transfer">
  <!--just in case we are dealing with an older interpreter-->
  <throw event="connection.disconnect.hangup"/>
</catch>

<catch event="connection.disconnect.hangup">
  <submit method="post" next="jsp/endcall.jsp"/>
  <exit/>
</catch>
```

4. Overview of VoiceXML elements

In this section, all VXML elements will be presented in alphabetical order. All the code samples presented here come from functional SpeechFrame applications.

4.1. Assign

Assigns a value to a variable.

Before using <assign> variables must be declared using <var>. If multiple variables with the same name (from different scopes) are active at the current scope (for scoping rules, see section 3.3.3.), then the one with the closest scope (for example, dialog versus document, document versus application etc.) will be used if not otherwise specified.

Required attribute is *name*, which contains the name of the variable. The optional attribute *expr* contains the ECMAScript expression that is evaluated and assigned to the variable.

Example:

```
<form id="start">
  <assign name="myvar" expr="0"/>
  ...
  ...
</form>
```

4.1.1. attributes

<assign>			
attributes	name	name of the variable	Required
	expr	ECMAScript expression that is evaluated and assigned to the variable	Required
parents	<block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>, <prompt>		
children	none		

4.2. Audio

Plays an audio file within a prompt. The pre-recorded sound file to be played is specified either in the *src* attribute, which contains the URI, or in the *expr* attribute which contains an ECMAScript expression that evaluates to a URI. For detailed explanation and examples, see section 7.1, Audio prompting.

The <audio> element can have alternate content in case the audio sample is not available (not supported by SpeechFrame).

4.2.1. Comsys enhancements to the <audio> tag

The way how audio fragments are played back can be altered through adding extra 'query' parameters to the src or expr attributes. If you need to add more than one enhanced attribute, separate them with a semicolon (";").

Example:

```
<var name="prm" expr="getTimeOfDayPrompt(101,102, 103, 101)"/>
<block>
  <!-- Play welcome message -->
  <prompt>
    <audio src="prompts/std/welcome.wav?speed=5;volume=15"/>
  </prompt>
</block>
```

4.2.2. Attributes

<audio>			
attributes	src	URI to a prerecorded audiofile	Optional
	expr	ECMAScript expression that evaluates to a URI to a prerecorded audiofile	Optional
Additional parameters for src or expr	offset	Offset in ms where play back begins	Optional
	volume	The volume denotes the amplification and is in the range - 50 - +50	Optional
	automaticGain Enabled	Whether or not to adapt the volume automatically to environmental noise	Optional
	speed	The play back speed from 0.1 – 10.0.	Optional
parents	<audio>, <block>, <catch>, <choice>, <enumerate>, <error>, <field>, <filled>, <help>, <if>, <initial>, <menu>, <noinput>, <nomatch>, <object>, <prompt>, <pros>, <record>, <subdialog>, <transfer>		
children	<audio>, <break>, <enumerate>, <value>		

4.3. Block

Container of so called executive content: specifies a block of directives that are executed in document source order. These can be ECMAScript logic, variable manipulation, playing audio prompts etc. <block> is a child of the form element, it may be placed either before any <field>s or among them; it is neither allowed as a child, nor as parent of <field>.

The following optional attributes may be used in <block>: *name*, *expr*, which contains an ECMAScript expression that supplies an initial value for the form item variable (default is "undefined"). If a value is provided, then the form item will not be visited until/unless it is first cleared. The optional *cond* attribute contains an ECMAScript expression that must evaluate to true for this element to be selected by the FIA (=form interpretation algorithm) when processing the document, that is, for the content of <block> to be executed.

Example:

```
<form id="example">
  <block>
    <assign name="world" expr="'hello'"/>
    <audio src="prompts/app/hello.wav"/>
    <goto next="#mainMenu"/>
  </block>
</form>
```

4.3.1. Attributes

<block>			
attributes	name	Valid ECMAScript variable name that declares the form item variable for this block	Optional
	expr	ECMAScript expression that supplies an initial value for the form item variable (default=undefined)	Optional
	cond	ECMAScript expression that must evaluate to true for this element to be selected by the FIA	Optional
parents	<form>		
children	<assign>, <audio>, <clear>, <data>, <disconnect>, <enumerate>, <exit>, <goto>, <if>, <log>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>		

4.4. Catch

Catches an event thrown by a VoiceXML application or the interpreter and defines an event handler to handle it. The required attribute *event* specifies the name of the event(s) to catch. Optionally the *count* attribute contains the number that specifies how many times event must be thrown before the FIA selects this particular handler (default is 1). Also optional is the *cond* attribute containing the ECMAScript expression that must evaluate to true for this handler to catch a given event (default is true).

Example of catches using the *count* attribute:

```
<catch event="recording_failed">
  <goto next="#try_again"/>
</catch>
<catch event="recording_failed" count="3">
  <audio src="prompts/std/failed.wav"/>
  <goto next="disconnect.vxml"/>
</catch>
```

4.4.1. Attributes

<catch>			
attributes	Event	Name of the event(s) to catch	Required
	Count	Number that specifies how many times <i>event</i> must be thrown before the FIA selects this particular handler (default is 1).	Optional
	Cond	ECMAScript expression that must also evaluate to true for this handler to catch a given event (default is true).	Optional
parents	<field>, <form>, <initial>, <menu>, <object>, <record>, <subdialog>, <transfer>, <vxml>		
children	<assign>, <audio>, <break>, <clear>, <data>, <disconnect>, <enumerate>, <exit>, <goto>, <if>, <log>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>,		

4.5. Choice

Child element of <menu>; defines a menu option. The <choice> element specifies the DTMF tone and either an event to throw or a URI to transition to if the caller presses a key that matches the DTMF tone. The required *next* attribute defines the URI to transition to when this choice is selected. Alternatively, the *event* attribute defines the event to throw when this choice is selected, and finally, *expr* can hold an ECMAScript expression that evaluates to a URI to transition to when this choice is selected. *next*, *event* and *expr* are mutually exclusive. The DTMF option that the caller can choose can be specified in the *dtmf* attribute; when not specified, a sequential ordering is provided by the *dtmf* attribute of the parent <menu>:

<menu dtmf="true">.

Example:

```
<menu id="mainMenu" dtmf="true">
  <prompt>
    <audio src="prompts/app/menuoptions.wav"/>
  </prompt>
  <choice dtmf="1" next="#option1"/>
  <choice dtmf="2" next="#option2"/>
  <choice dtmf="0" next="disconnect.vxml"/>
</menu>
```

4.5.1. Attributes

<choice>			
Attributes	Next	URI to transition to when this choice is selected.	Required
	Event	Event to throw when this choice is selected.	Required
	Expr	ECMAScript expression that evaluates to a URI to transition to when this choice is selected.	Required
	Dtmf	A dual-tone multifrequency (DTMF) tone that the caller can use (in lieu of speaking) to select this choice.	Optional
parents	<menu>		
children	<audio>, <enumerate>, <grammar>, <value>		

4.6. Clear

Resets the specified form items: reinitialises variables to "undefined"; within a form item, also resets the prompt and event counters for that form item (sets counter value to zero). An optional *namelist* attribute holds a space-delimited list of variables to reset; default is to reset all form item variables in the current context.

Example:

```
<filled>
  <if cond="answer == 'OK'">
    <goto next="#continue"/>
  <else/>
    <prompt>
      <audio src="wronganswer.wav"/>
    </prompt>
    <clear namelist="answer"/>
    <goto nextitem="start"/>
  </if>
</filled>
```

4.6.1. Attributes

<clear>			
attributes	namelist	Space-delimited list of variables to reset (default is to reset all form item variables in the current context).	Optional
parents	<block>, <catch>, <error>, <filled>, <if>, <noinput>, <nomatch>, <prompt>		
children	none		

4.7. Disconnect

Terminates the phone call: hangs up the telephone call and ends the VXML session. Before ending the session, `<disconnect>` first throws a *connection.disconnect.hangup* event. See section 3.3.6 for explanation of how this element is used to perform posthangup processing.

Example:

```
<form id="end">
  <block>
    <audio src="thanks.wav"/>
    <audio src="goodbye.wav"/>
    <disconnect/>
  </block>
</form>
```

4.7.1. Attributes

<disconnect>	
parents	<code><block></code> , <code><catch></code> , <code><error></code> , <code><filled></code> , <code><help></code> , <code><if></code> , <code><noinput></code> , <code><nomatch></code> , <code><prompt></code>
children	none

4.8. Else, elseif

Else provides the final alternative in an *if* construct; elseif provides an alternative conditional statement in an *if* construct. If the *cond* attributes of the containing `<if>` and all preceding `<elseif>` elements evaluate to false, then the content between this `<elseif>` and the next `<elseif>`, `<else>` or `</if>` are executed.

As per ECMAScript convention, expressions that evaluate to 0, -0, null, false, NaN, undefined, and the empty string are evaluated as false. All other values, including the strings "0" and "false" are equivalent to true.

Example:

```
<block>
  <if cond="calledNumber=='2116'">
    <goto next="application1.vxml"/>
  <elseif cond=" calledNumber == '2117'"/>
    <goto next=" application2.vxml"/>
  <else/>
    <audio src="prompt/app/error.wav"/>
    <throw event="unknown_application"/>
  </if>
</block>
```

4.8.1. Attributes

<else>			
Attributes		none	
Parents	<if>		
Children	None		

<elseif>			
Attributes	cond	ECMAScript expression that must evaluate to true for the clause to execute.	Required
Parents	<if>		
Children	None		

4.9. Error

One of the shorthand notations for platform-defined event handlers; shorthand for: `<catch event="error">`. Catches and handles platform-defined and user-defined events with names that begin with the string "error".

Example:

```
<error>
  <audio src="prompts/std/error.wav"/>
  <disconnect/>
</error>
```

4.9.1. Attributes

<error>			
attributes	count	Number that specifies how many times an error event must be thrown before the FIA selects this particular handler (default is 1).	Optional
	cond	ECMAScript expression that must also evaluate to true for this handler to catch a given event (default is true).	Optional
parents	<code><field></code> , <code><form></code> , <code><initial></code> , <code><menu></code> , <code><object></code> , <code><record></code> , <code><subdialog></code> , <code><transfer></code> , <code><vxml></code>		
children	<code><assign></code> , <code><audio></code> , <code><break></code> , <code><clear></code> , <code><disconnect></code> , <code><enumerate></code> , <code><exit></code> , <code><goto></code> , <code><if></code> , <code><log></code> , <code><prompt></code> , <code><reprompt></code> , <code><return></code> , <code><script></code> , <code><submit></code> , <code><throw></code> , <code><value></code> , <code><var></code>		

4.10. Exit

Immediately terminates the current application and returns control to the interpreter. Unlike `<return>` which terminates a `<subdialog>` and returns control to the calling context, `<exit>` immediately ends the entire session. It does not throw an exit event; the platform immediately hangs up the phone and does NOT throw a `connection.disconnect.hangup` event for posthangup processing as with `<disconnect/>`. If specified, the optional *expr* (ECMAScript expression that evaluates to the value assigned to the variable) and *namelist* (list of variables to return) attributes provide a mechanism for developers to send particular information to the VXML interpreter upon exiting.

Example:

```
<catch event="connection.disconnect.hangup">
  <audio src="prompts/std/goodbye.wav"/>
  <exit/>
</catch>
```

4.10.1. Attributes

<exit>			
attributes	expr	ECMAScript expression that evaluates to the value assigned to the variable.	Optional
	namelist	List of variable names to return (default is return nothing).	Optional
parents	<code><block></code> , <code><catch></code> , <code><error></code> , <code><filled></code> , <code><help></code> , <code><if></code> , <code><noinput></code> , <code><nomatch></code>		
children	None		

4.11. Field

One of the basic elements of VXML: formulates an interactive dialog between the user and the system by means of prompting the caller for input, collecting and matching the input to the appropriate grammar and assigning the value to the field item variable.

If one or more grammars have been matched within a field, the <filled> element contained within it is executed. The only required attribute is *name*, which holds a valid ECMAScript variable name that declares the field item variable for this field. A successful recognition (=grammar match) within this field will store the recognition result in *name* before executing the field's <filled> element. In the optional *type* attribute one of the platform-defined ("builtin") grammars can be specified. See section 4.15, Grammar, for an overview of the available platform-defined grammars and their syntax. As *type* already specifies a grammar, this attribute is mutually exclusive with including <grammar> elements within the field.

Example:

```
<field name="pincode" type="digits">
  <prompt>
    <audio src="prompts/app/enterpin.wav"/>
  </prompt>
</field>
```

4.11.1. Attributes

<field>			
attributes	cond	ECMAScript expression that must also evaluate to true for the FIA to select this field (default is true).	Optional
	expr	ECMAScript expression that, if specified, is evaluated and provides an initial value for the field's field item variable <i>name</i> (default is undefined). If a value is provided, then the form item will not be visited until/unless it is first cleared.	Optional
	name	Valid ECMAScript variable name that declares the field item variable for this field. A successful recognition within this field will store the recognition result in <i>name</i> before executing the field's <filled> element.	Required
	type	One of several platform-defined grammars. The <i>type</i> attribute is mutually exclusive with including <grammar> elements within the field.	
parents	<form>		
children	<audio>, <catch>, <enumerate>, <error>, <filled>, <grammar>, <help>, <link>, <noinput>, <nomatch>, <prompt>, <property>, <value>		

4.12. Filled

Specifies an action to perform when a given field or fields have been "filled in", that is, when the VoiceXML interpreter assigns a value to the named variable of a field, transfer, record, or subdialog element.

If used as a child of a specific <field> or form item (for example, <transfer>), then it is executed when that field's item variable becomes filled in, either via a successful recognition or explicit assignment of the field item variable.

Alternatively, <filled> can be used as a child of <form> to cover multiple fields, using the *mode* and *namelist* attributes to specify which fields to act upon. When used as a child of <form>, *mode* specifies whether "all" or "any" of the fields specified in *namelist* must be filled in order to execute this <filled> element (default is "all"); *namelist* specifies a space-delimited list of field *names* within this form used in conjunction with *mode* to determine behaviour (default is all fields in the form). When used as a child of <field> or another form item, it is invalid to specify *mode* and/or *namelist*.

Example:

```
<field name="pincode" type="digits">
  <prompt>
    <audio src="prompts/app/enterpin.wav"/>
  </prompt>
  <filled>
    <goto next="checkpincode.jsp"/>
  </filled>
</field>
```

4.12.1. Attributes

<filled>			
attributes	mode	When used as a child of <form>, specifies whether "all" or "any" of the fields specified in <i>namelist</i> must be filled in order to execute this <filled> element (default is all). When used as a child of <field> or another form item, it is invalid to specify <i>mode</i> .	Optional
	namelist	When used as a child of <form>, specifies a space-delimited list of field <i>names</i> within this form used in conjunction with <i>mode</i> to determine behavior (default is all fields in the form). When used as a child of <field> or another form item, it is invalid to specify <i>namelist</i> .	Optional
parents	<field>, <form>, <object>, <record>, <subdialog>, <transfer>		
children	<assign>, <audio>, <break>, <clear>, <disconnect>, <enumerate>, <exit>, <goto>, <if>, <log>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>		

4.13. Form

One of the basic elements of VXML. It contains a series of so called field items (as opposed to "control items"): this is typically <field>, but it may be another element that needs to be filled through interaction with the caller such as <record> or <subdialog>, and/or control items with executable content such as event handlers, links, or a <transfer> or <block> element. As the primary unit of dialog in VXML applications, <form> may only be a direct child of <VXML>, the root element of a VXML document. The optional *id* attribute holds the name for the form, so that <goto> and other elements can explicitly transition to it using "#" notation (=transition within the same document).

Example:

```
<form id="recordingDemo">
  <record name="recording" beep="true" maxtime="120s" finalsilence="4s" dtmfterm="true">
    <prompt>
      <audio src="prompts/app/saysomething.wav"/>
    </prompt>
    <filled>
      <audio src="prompts/app/youaid.wav"/>
      <audio expr="cal"/>
      <submit next="saveRecording.jsp" namelist="recording"/>
    </filled>
  </record>
</block>
  <throw event="recording_failed"/>
</block>
</form>
```

4.13.1. Attributes

<form>			
attributes	id	Name for the form, so that <goto> and other elements can explicitly transition to it using "#" notation (e.g. <goto next="#top"/>). As with HTML anchors, <i>ids</i> must be unique within a given VoiceXML document.	Optional
	scope	Sets the default scope of the form's grammars: Document: the form's grammars are active throughout the current document. Dialog: the form's grammars are active throughout the current form (default is dialog).	Optional
parents	<vxml>		
children	<block>, <catch>, <data>, <error>, <field>, <filled>, <grammar>, <help>, <initial>, <link>, <noinput>, <nomatch>, <object>, <property>, <record>, <script>, <subdialog>, <transfer>, <var>		

4.14. Goto

Jumps to the specified URI. The location to transition to within the same or different document needs to be specified by one of the required, mutually exclusive attributes: *expr*, *exritem*, *next* or *nextitem*.

To transition to another form item within the same form, use the *nextitem* attribute, or the *exritem* attribute if the form item name is computed using an ECMAScript expression:

```
<goto nextitem="ssn_confirm"/>
```

```
<goto exritem="(type==12)? 'ssn_confirm' : 'reject'"/>
```

Note : the conditional operator statement `(type==12)? 'ssn_confirm' : 'reject'` is a shorthand for :

```
<if cond="type==12">  
  <goto nextitem="ssn_confirm"/>  
<else/>  
  <goto nextitem="reject"/>  
</if>
```

To go to another dialog in the same document, use *next* (or *expr*) with only a URI fragment:

```
<goto next="#another_dialog"/>
```

```
<goto expr="'#' + 'another_dialog'"/>
```

To transition to another document, use *next* (or *expr*) with a URI:

```
<goto next="http://flight.example.com/reserve_seat"/>
```

```
<goto next="./special_lunch#wants_vegan"/>
```

```
<form>  
  <block>  
    <script>  
      var startPage="Dnis" + session.telephone.dnis + ".vxml";  
    </script>  
    <goto expr="startPage"/>  
  </block>  
</form>
```

4.14.1. Attributes

<goto>			
attributes	expr	ECMAScript expression that evaluates to the URI of a dialog in the current or a different document (e.g. "http://foo.com/bar.vxml," "#top," "bar.vxml#menu").	Required
	exritem	ECMAScript expression that evaluates to the <i>name</i> of another form item (e.g. field, transfer, etc.) within the current <form> or <menu>.	Required
	next	URI of a dialog in the current or a different document.	Required
	nextitem	<i>Name</i> of another form item within the current <form> or <menu>.	Required
parents	<block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>, <prompt>		
children	none		

4.15. Grammar

Specifies a permissible vocabulary for user interaction, either in natural language expressions (words, phrases) when a speech or a speech grammar (=based on ASR, automatic speech recognition) is deployed or in digits and symbols (0-9, *, #) when a DTMF grammar is deployed.

In SpeechFrame, the following platform-defined (builtin) DTMF grammars are supported:

grammar	DTMF input	Result
boolean	1 = yes, 2 = no	"true" for "yes", "false" for "no"
date	0-9; format: yyyyymmdd	yyyyymmdd
digits	0-9	string of 0-9
currency	0-9; "*" acts as decimal point	mm.nn
number	0-9; "*" acts as decimal point	string of 0-9, decimal point "."
phone	0-9;	string of 0-9
time	0-9; format: hhmm	hhmm (24 hour clock time)

These grammars can be accessed either using the *type* attribute in <field> (Example:

```
<field type="boolean">
```

), or using the "builtin" URI namespace for a <grammar> *src* (Example:

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

4.15.1. Attributes

<grammar>			
attributes	mode	Either "dtmf" or "voice," indicating whether the grammar is for touchtone or spoken input. If the specified grammar does not match the specified mode, a <i>badfetch</i> event is thrown (default is voice).	Optional
	root	For inline XML grammar, defines the public rule which acts as the root rule of the grammar. The root rule is only used when the grammar is inline and must be present when using an inline XML grammar to identify which rule to activate.	Optional
	scope	Sets the default scope of the grammar: Document: the form's grammar are active throughout the current document. Dialog: the form's grammar are active throughout the current form. If omitted, the grammar scoping is resolved by looking at the parent element.	Optional
	src	URI to a valid grammar document. <i>src</i> and inline grammar definitions are mutually exclusive, but one must be specified. This URI may reference one of the platform-defined VoiceXML grammars using the special "builtin:" URI namespace.	Required
	type	MIME type for the grammar format being used either inline or via <i>src</i> . If omitted, the recognizer will attempt to determine the format dynamically. MIME types for commonly used grammar formats: "application/grammar+xml" W3C XML Grammar Format, "Application/grammar" W3C Augmented BNF Grammar Format, "application/x-gsl" Nuance Grammar Specification Language, "application/x-jsjf" Java Grammar Format	Optional
	version	Defines the version of the grammar format specified by <i>type</i> being used (default is 1.0).	Optional
	weight	Positive floating point (or 0) number that indicates the likelihood that the caller will say something to match this grammar, relative to other active grammars (default is 1.0).	Optional
	xml:lang	Specifies the language and locale identifier of the contained or referenced grammar according to RFC 1766 (e.g. "fr-CA" for Canadian French). If omitted, the value is inherited down from the document hierarchy, and ultimately a VoiceXML interpreter default.	Optional
parents	<choice>, <field>, <form>, <link>, <record>, <transfer>		
children	Grammar format-specific		

4.16. If

Performs conditional logic. If the *cond* attribute evaluates to true, then the content between this `<if>` and the next `<elseif>`, `<else>` or `</if>` is executed.

As per ECMAScript convention, expressions that evaluate to 0, -0, null, false, NaN, undefined, and the empty string are evaluated as false. All other values, including the strings "0" and "false" are equivalent to true.

Example:

```
<form id="askPincode">
  <field name="pincode" type="digits">
    <prompt>
      <audio src="prompts/app/enterpin.wav"/>
    </prompt>
    <filled>
      <if cond="pincode != checkpincode">
        <clear namelist="pincode"/>
        <throw event="nomatch"/>
      </if>
      <goto next="continue.vxml"/>
    </filled>
  </field>
</form>
```

4.17. Log

Writes a message to the debug log. <log> outputs debug messages to the platform-specific debug logs.

Example:

```
<field name="choice" type="digits?length=8">
  <prompt>
    <audio src="prompts/app/enterchoice.wav"/>
  </prompt>
  <filled>
    <log>The caller has entered: <value expr="choice"/>
  </log>
  </filled>
</field>
```

4.17.1. Attributes

<if>			
attributes	cond	ECMAScript expression that must evaluate to true for the clause to execute.	Required
parents	<block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>		
children	<clear>, <data>, <disconnect>, <else>, <elseif>, <enumerate>, <exit>, <goto>, <if>, <log>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>		

4.18. Menu

A shorthand for a simple form that contains a single field for making a selection between fixed choices. By means of the `<choice>` element `<menu>` presents a list of choices to the user and transitions to the chosen URI, defined in the *next* attribute of `<choice>`.

The optional *dtmf* attribute enables DTMF selection for all choices in this menu; true: for child `<choice>` elements that do not explicitly specify a DTMF sequence using *dtmf*, 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 (default is false). The optional *id* attribute names the menu, so that `<goto>` and other elements can explicitly transition to it using "#" notation (-transition within the same document). *ids* must be unique within a given VXML document.

Example:

```
<menu id="mainMenu" dtmf="true">
  <prompt>
    <audio src="prompts/app/menu.wav"/>
  </prompt>
  <choice dtmf="1" next="#choice1"/>
  <choice dtmf="2" next="#choice2"/>
</menu>

<form id="choice1">
  <block>
    <prompt>
      <audio src="prompts/app/thisischoice1.wav"/>
    </prompt>
    <exit/>
  </block>
</form>

<form id="choice2">
  <block>
    <prompt>
      <audio src="prompts/app/thisischoice2.wav"/>
    </prompt>
    <exit/>
  </block>
</form>
```

4.18.1. Attributes

<menu>			
Attributes	dtmf	Enables DTMF selection for all choices in this menu: True: for child <choice> elements that do not explicitly specify a DTMF sequence using <i>dtmf</i> , 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 (default is false).	Optional
	id	Name for the menu, so that <goto> and other elements can explicitly transition to it using "#" notation (e.g. <goto next="#top"/>). As with HTML anchors, <i>ids</i> must be unique within a given VoiceXML document.	Optional
	scope	Sets the default scope of the menu's grammars: Document: the menu's grammars are active throughout the current document. Dialog: the menu's grammars are active throughout the current form (default is dialog).	Optional
parents	<vxml>		
children	<audio>, <catch>, <choice>, <enumerate>, <error>, <help>, <noinput>, <nomatch>, <prompt>, <property>, <script>, <value>, <var>		

4.19. Noinput

One of the shorthand notations for platform-defined event handlers; shorthand for `<catch event="noinput">`. Executes when the user does not speak or enter DTMF within the active field. The properties *termtimeout* and *timeout* govern how long the interpreter waits for caller input before throwing the *noinput* event. The optional *count* attribute specifies how many times a *noinput* event must be thrown before the FIA selects this particular handler (default is 1).

Example:

```
<noinput>
  <reprompt/>
</noinput>

<noinput count="3">
  <audio src="prompts/std/noinput.wav" />
  <goto next="disconnect.vxml"/>
</noinput>
```

4.19.1. Attributes

<noinput>			
attributes	count	Number that specifies how many times a noinput event must be thrown before the FIA selects this particular handler (default is 1).	Optional
	cond	ECMAScript expression that must also evaluate to true for this handler to catch a given event (default is true).	Optional
parents	<field>, <form>, <initial>, <menu>, <object>, <record>, <subdialog>, <transfer>, <vxml>		
children	<assign>, <audio>, <break>, <clear>, <disconnect>, <enumerate>, <exit>, <goto>, <if>, <log>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>		

4.20. Nomatch

One of the shorthand notations for platform-defined event handlers; shorthand for `<catch event="nomatch">`. Catches and handles the platform-defined event for when the caller responds something that is not recognised as part of any of the active grammars. Several properties, such as *termtimeout*, *interdigittimeout* etc, govern exactly how the interpreter decides when to throw the *nomatch* event. The optional *count* attribute specifies how many times a *nomatch* event must be thrown before the FIA selects this particular handler (default is 1).

Example:

```
<nomatch>
  <audio src="prompts/std/nomatch.wav" />
  <reprompt/>
</nomatch>

<nomatch count="3">
  <audio src="prompts/std/toomanyerrors.wav" />
  <goto next="disconnect.vxml" />
</nomatch>
```

4.20.1. Attributes

<nomatch>			
attributes	count	Number that specifies how many times a match event must be thrown before the FIA selects this particular handler (default is 1).	Optional
	cond	ECMAScript expression that must also evaluate to true for this handler to catch a given event (default is true).	Optional
parents	<field>, <form>, <initial>, <menu>, <object>, <record>, <subdialog>, <transfer>, <vxml>		
children	<assign>, <audio>, <break>, <clear>, <disconnect>, <enumerate>, <exit>, <goto>, <if>, <log>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>		

4.21. Option

Specifies an option to a user - a simple available choice and grammar - within a field. Similar to how <choice> works within the even simpler <menu> construct, <option> makes it possible to quickly provide a list of simple alternatives within a <field>. The optional *dtmf* attribute specifies the DTMF sequence that the caller can enter to select this option. Also optionally, *value* can be attributed to the option, to hold the string to assign to the field's field item variable if the caller selects this option. If not specified, *dtmf* is returned.

Example:

```
<form id="mainMenu">
  <field name="mainMenuChoice">
    <prompt>
      <audio src="prompts/app/entersomething.wav"/>
      <option dtmf="1" value="'1pressed'"/></option>
      <option dtmf="2" value="'2pressed'"/></option>
    </prompt>
    <filled>
      <if cond="mainMenuChoice == '1pressed'">
        <goto next="document1.vxml"/>
      <elseif cond="mainMenuChoice == '2pressed'"/>
        <goto next="document2.vxml"/>
      <else/>
        <goto next="whateverDocument.vxml"/>
      </if>
    </filled>
  </field>
</form>
```

4.21.1. Attributes

<option>			
attributes	dtmf	DTMF sequence that the caller can enter to select this option	Optional
	Value	String to assign to the field's field item variable if the caller selects this option. If not specified, <i>dtmf</i> is returned.	Optional
parents	<field>		
children	none		

4.22. Param

Specifies a value to pass to a subdialog element. Submits data to a <subdialog>, without the need for server-side scripting. For a <subdialog>, all data passed via <param> becomes available as a series of variables in the new interpreter context. These variables must be explicitly declared in the document invoked by the <subdialog>, at which point they are initialised with the values passed in from the invoking dialog via <param>. Any *expr* attribute on the declaring <var> elements are ignored.

Either the *expr* (ECMAScript expression) or the *value* attribute is required to hold the value that is used for this parameter. Also required is *name*, which specifies the ECMAScript variable name to label this parameter. When used with <subdialog>, each <param> generates a corresponding variable for its *name* in the new application context.

Example:

```
<subdialog name="subdiagdemo" src="page2.vxml">
  <param name="var1" expr="'hello'"/>
  <filled>
    <if cond = "subdiagdemo.result == 'OK'">
      <log>Subdialogue correctly executed</log>
    <else/>
      <log>Subdialogue reported <value expr="subdiagdemo.result"/>
      </log>
    </if>
  </filled>
</subdialog>

page2.vxml:

<?xml version='1.0'?>
<VXML version="2.0">

  <form id="start">
    <!--Note: the param variable var1 must be declared within
    The form, otherwise a symantic error will be thrown!
    -->
    <var name="var1"/>
    <var name="result"/>

    <block>
      <if cond="var1=='hello'">
        <assign name="result" expr="'OK'"/>
      <else/>
        <assign name="result" expr="'NOTOK'"/>
      </if>
    </block>
  </form>
</vxml>
```

4.22.1. Attributes

<param>			
attributes	Expr	ECMAScript expression that is used as the value for this parameter.	Optional
	name	Legal ECMAScript variable name to label this parameter. When used with <subdialog>, each <param> generates a corresponding variable for its <i>name</i> in the new application context.	Required
	Value	Value that is used for this parameter.	Required
Parents	<object>, <subdialog>		
Children	None		

4.23. Prompt

Queues recorded audio and synthesized speech in an interactive dialog. The optional attribute *bargein* specifies whether or not the caller can interrupt the prompt (default is true). In order to facilitate tapered prompting, the optional *count* attribute specifies how many times the caller must have visited the form item containing this prompt before the FIA selects this particular prompt (default is 1).

For detailed explanation and examples, see section 7.1, Audio prompting.

Example:

```
<form id="herhaal">
  <block>
    <prompt>
      <audio src="prompts/app/myprompt.wav"/>
    </prompt>
  </block>
</form>
```

4.23.1. Attributes

<prompt>			
attributes	bargein	Boolean specifying whether or not the caller can interrupt the prompt (default is true).	Optional
	Bargeintype	Specifies whether the recognition engine should use the "speech," or "hotword" method to determine bargein (default is platform-specific).	Optional
	cond	ECMAScript expression that must also evaluate to true for this prompt to be spoken (default is true).	Optional
	count	Number that specifies how many times the caller must have visited the form item containing this prompt before the FIA selects this particular prompt (default is 1).	Optional
	timeout	Time to wait in seconds (s) or milliseconds (ms) before throwing a noinput event. If multiple prompt elements specify timeout, the most recent one is used (default is platform-specific).	Optional
parents	<block>, <catch>, <error>, <field>, <filled>, <help>, <if>, <initial>, <menu>, <noinput>, <nomatch>, <object>, <record>, <subdialog>, <transfer>		
children	<assign>, <audio>, <break>, <clear>, <data>, <disconnect>, <enumerate>, <exit>, <goto>, <if>, <log>, <prompt>, <reprompt>, <return>, <script>, <submit>, <throw>, <value>, <var>		

4.24. Property

Specifies a platform-defined behaviour setting for an entire application, document, or form. Property settings follow standard VXML scoping rules: for example, local settings within a <field> override settings from the parent <form> etc.

The required *name* attribute specifies the property name and the *value* attribute holds the assigned value.

Example:

```
<property name="bargein" value="true"/>
<property name="timeout" value="15s"/>
<property name="interdigittimeout" value="5s"/>
<property name="termtimeout" value="5s"/>
```

4.24.1. Attributes

<property>			
attributes	name	Property name.	Required
	value	Property value. Note: Some VoiceXML platforms extend the <property> element, allowing developers to specify an <i>expr</i> attribute on <property> as a programmatic way of computing <i>value</i> .	Required
parents	<field>, <form>, <initial>, <menu>, <object>, <record>, <subdialog>, <transfer>, <vxml>		
children	none		

4.25. Record

Records audio from the caller. As a form item like `<field>`, the `<record>` element can include prompts, event handlers and a `<filled>` item for handling a successful recording. If the caller never begins to speak and remains silent for *finalsilence*, then the interpreter throws *noinput*.

Once a successful recording has been made, the `<filled>` element contained within is executed. The name attribute of `<record>` is filled with the URI where the recording can be found. This can be used to play the recording back to the user and/or be submitted to a web server to make the recording externally available. Use the *expr* attribute on `<audio>` to play the recording back to the caller, and the *namelist* attribute of `<submit>` or `<subdialog>` to transfer it to a web server.

Example:

```
<form id="recordingDemo">
  <record name="recording" beep="true" maxtime="120s" finalsilence="4s" dtmfterm="true">
    <prompt>
      <audio src="prompts/app/saysomething.wav"/>
    </prompt>
    <filled>
      <audio src="prompts/app/yousaid.wav"/>
      <audio expr="cal"/>
      <submit next="saveRecording.jsp" namelist="recording"/>
    </filled>
  </record>
</block>
  <throw event="recording_failed"/>
</block>
</form>
```

4.25.1. Attributes

<record>			
attributes	beep	If "true," play a recognizable tone to signal the caller that recording is about to begin (default is false).	Optional
	cond	ECMAScript expression that must also evaluate to true for the FIA to select this form item (default is true).	Optional
	dest	URI for the destination of the recording, for platforms that may support storage of recording to streaming media or messaging servers. If the recording destination cannot be accessed for audio playback and/or HTTP POST submit, <i>error.semantic</i> is thrown when the recording form item variable is referenced. Platforms are not required to support <i>dest</i> .	Optional
	dtmfterm	If "true," recording is terminated when the caller presses any DTMF key. Otherwise, DTMF input is ignored and just considered part of the recording. If true, the <i>name\$.termchar</i> shadow variable contains which key was pressed first (default is false).	Optional
	expr	ECMAScript expression that, if specified, is evaluated and provides an initial value for the record's form item variable <i>name</i> (default is undefined). If a value is provided, then the form item will not be visited until/unless it is first cleared.	Optional
	finalsilence	Time in seconds (s) or milliseconds (ms) of silence before the recognizer stops recording (default is platform-specific). If the caller remains silent for <i>finalsilence</i> , the interpreter throws <i>noinput</i> .	Optional
	maxtime	Maximum time allowed for recording in seconds or milliseconds (default is platform-specific). For security and performance, most platforms will enforce a platform-specific maximum for <i>maxtime</i> .	Optional
	name	Valid ECMAScript variable name that declares the form item variable for this record. A successful recording will store the result in <i>name</i> before executing the record's <filled> element.	Optional
	type	MIME type to use for encoding the recorded data (e.g. "audio/wav"). If an unsupported format is specified, the interpreter throws <i>error.unsupported.format</i> (default is platform-specific).	Optional
parents	<form>		
children	<audio>, <catch>, <enumerate>, <error>, <filled>, <grammar>, <help>, <noinput>, <nomatch>, <prompt>, <property>, <value>		

4.26. Reprompt

Indicates that the Platform should select and queue the appropriate prompt element before entering a listen state in an interactive dialog. <reprompt> requeues the most recent <prompt> before listening again for the caller's input.

Within an event handler such as noinput, nomatch or help, <reprompt> increases the prompt counter by 1 and reenters the FIA (=form interpretation algorithm) for the current form item.

It is important to note that the reprompt element has no effect within a filled element because the field item variable has already been filled and the FIA will not reselect it. To reenter the dialog and get prompts requeued after the form item has been filled, <clear> the form item variable.

Example:

```
<field name="ddi" type="digits">
  <prompt>
    <audio src="prompts/app/enterddi.wav"/>
  </prompt>
  <filled>
    <assign name="Gddi" expr="ddi"/>
    <goto nextitem="get_ddi"/>
  </filled>

  <nomatch>
    <prompt>
      <audio src="prompts/app/noddientered.wav"/>
    </prompt>
    <clear namelist="ddi"/>
    <reprompt/>
  </nomatch>
</field>
```

4.26.1. Attributes

<reprompt>	
parents	<block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>, <prompt>
children	none

4.27. Return

Ends a <subdialog> and returns control to the calling application. <return> can specify either an event to throw (which it throws at the scope of the invoking (=calling) <subdialog> and can be handled there locally or by inherited handlers from parent scopes) or a list of variables to return and "fill" the subdialog's form item variable. The event to throw when the <subdialog> returns control to the calling application is specified in the *event* attribute. The *namelist* attribute specifies a space-delimited list of variables to return when the <subdialog> returns to the calling application. The original subdialog's <filled> element is executed, and all variables are returned as properties of an ECMAScript object that is assigned to the subdialog's *name* form item variable. *Event* and *namelist* are mutually exclusive, but neither is required.

Example:

```
<form id="get_mailbox">
  <var name="result" expr="'OK'"/>
  <var name="mailbox" expr="'12345'"/>
  <block>
    <if cond="result=='OK'">
      <return namelist="result mailbox"/>
    <else/>
      <return event="nomailbox"/>
    </if>
  </block>
</form>
```

4.27.1. Attributes

<return>			
attributes	event	Event to throw when the <subdialog> returns control to the calling application.	Optional
	namelist	Space-delimited list of variables to return when the <subdialog> returns to the calling application. The original subdialog's <filled> element is executed, and all variables are returned as properties of an ECMAScript object that is assigned to the subdialog's <i>name</i> form item variable.	Optional
parents	<block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>		
children	none		

4.28. Script

Includes a block of ECMAScript code. `<script>` defines and executes a block of client-side code. An optional *src* attribute specifies the URI of the location of a valid ECMAScript document. If not specified, an inline script definition must be provided. *Src* and inline script are mutually exclusive. The VXML interpreter uses an XML parser to parse a VXML document, so it is necessary to embed a block of inline script within an XML CDATA section to prevent the XML parser from parsing the script. Otherwise, characters such as `<`, `>`, and `&` will generate an error.

Example:

```
<script><![CDATA[
function getTODPrompt() {
  var d = new Date();
  var h = d.getHours();
  var promptNr;

  if (h >= 0 && h < 12)
    promptNr=101
  else if (h >= 12 && h < 18)
    promptNr=102
  else if (h >= 18 || h < 24)
    promptNr=103
  else
    promptNr=0;

  return promptNr;
}
]]>
</script>

<form id="greeting">
  <var name="prm" expr="getTODPrompt()"/>
  <block>
    <if cond="prm > 0">
      <!-- Play time dependent message -->
      <prompt><audio expr="prm"/></prompt>
    </if>
  </block>
</form>
```

4.28.1. Attributes

<script>			
attributes	charset	If <i>src</i> is specified, the character encoding of the script. UTF-8 and UTF-16 encodings of 10646 must be supported (as in XML) and other encodings, as defined in the IANA character set registry, may be supported. The default value is UTF-8.	Optional
	src	URI specifying the location of a valid ECMAScript document. If not specified, an inline script definition must be provided. If both are specified, <i>src</i> is used and the inline script is ignored.	Optional

<i>parents</i>	<block>, <catch>, <error>, <filled>, <help>, <if>, <menu>, <noinput>, <nomatch>, <prompt>, <vxml>
<i>children</i>	none

4.29. Subdialog

Jumps to and executes an encapsulated dialog. In the required *src* attribute the URI is specified to a valid VXML document that will be invoked in a completely independent interpreter context as a subdialog. At some point, this document is expected to use `<return>` to return control to the invoking application. The required *name* attribute holds a valid ECMAScript variable name that declares the form item variable for this subdialog. Optionally, the *namelist* attribute can hold a space-separated list of variables to be submitted via HTTP along with the request to the URI specified by *src*.

`<subdialog>` invokes modularized VXML code in an **independent application context**. Once a subdialog has been invoked, `<return>` can specify either an event to throw or a list of variables to return. If an event is thrown, it is thrown at the scope of the invoking dialog and can be handled there locally or by inherited handlers from parent scopes. If a list of variables is returned, the *name* form item variable is filled with an ECMAScript object whose properties correspond to the returned variables and the subdialog's `<filled>` element is executed.

Example:

```
<subdialog name="subdiagdemo" src="page2.vxml">
  <param name="var1" expr="'hello'"/>
  <filled>
    <if cond = "subdiagdemo.result == 'OK'">
      <log>Subdialogue correctly executed</log>
    <else/>
      <log>Subdialogue reported <value expr="subdiagdemo.result"/>
      </log>
    </if>
  </filled>
</subdialog>

page2.vxml:

<?xml version='1.0'?>
<VXML version="2.0">

  <form id="start">
    <!--Note: the param variable var1 must be declared within
      The form, otherwise a symantic error will be thrown!
    -->
    <var name="var1"/>
    <var name="result"/>

    <block>
      <if cond="var1=='hello'">
        <assign name="result" expr="'OK'"/>
      <else/>
        <assign name="result" expr="'NOTOK'"/>
      </if>
    </block>
  </form>
</vxml>
```

4.29.1. Attributes

<subdialog>			
attributes	cond	ECMAScript expression that must also evaluate to true for the FIA to select this form item (default is true).	Optional
	enctype	MIME type to use to encode any included data specified by <i>namelist</i> (default is "application/x-www-form-urlencoded". Interpreters must also support "multipart/form-data" and may support additional encoding types).	Optional
	expr	ECMAScript expression that, if specified, is evaluated and provides an initial value for the subdialog's form item variable <i>name</i> (default is undefined). If a value is provided, then the form item will not be visited until/unless it is first cleared.	Optional
	method	The HTTP method to use to send the request; either "get" or "post" (default is get).	Optional
	name	Valid ECMAScript variable name that declares the form item variable for this subdialog. If the invoked subdialog uses the <return> <i>namelist</i> attribute to return data to this dialog, <i>name</i> is set to a ECMAScript object whose properties correspond to the list of variables returned via <i>namelist</i> .	Required
	namelist	A space-separated list of variables to be submitted via HTTP along with the request to the URI specified by <i>src</i> . <i>Namelist</i> is only valid if <i>src</i> references a URI to another document (e.g. requires an actual HTTP request to be made to the Web server).	Optional
	src	URI to a valid VXML document that will be invoked in a completely independent interpreter context as a subdialog. At some point, this document is expected to use <return> to return control to the invoking application.	Required
Parents	<form>		
Children	<audio>, <catch>, <enumerate>, <error>, <filled>, <help>, <noinput>, <nomatch>, <param>, <prompt>, <property>, <value>		

4.30. Submit

Obtains a new document via an HTTP GET or POST request. The required *next* or *expr* (ECMAScript expression that evaluates to a URI) attribute specifies the URI to explicitly transition to. The URI must reference to another document; to transition to another dialog within the same document, <goto> or <subdialog> is used. In the case of a fragment, the URI requested is the base URI of the current document.

An optional *namelist* attribute holds a space-separated list of variables to be submitted via HTTP along with the request to the specified URI.

Example:

```
<form id="activate">
  <block>
    <submit next="activate.jsp" namelist="mailbox pincode"/>
  </block>
</form>
```

4.30.1. Attributes

<submit>			
Attributes	enctype	MIME type to use to encode any included data specified by <i>namelist</i> (default is "application/x-www-form-urlencoded." Interpreters must also support "multipart/form-data" and many support additional encoding types). Note: If a variable containing recorded audio from <record> is included in <i>namelist</i> , then <i>enctype</i> should be set to "multipart/form-data." If it is not, the behavior is unspecified-some platforms may override <i>enctype</i> and send as multipart/form-data; others may attempt to send as specified. However, this may severely impact performance.	Optional
	expr	ECMAScript expression that evaluates to a URI to explicitly transition to. For <submit>, the URI must reference another document; to transition to another dialog within the same document, use <goto> or <subdialog>.	Required
	method	The HTTP method to use to send the request; either "get" or "post" (default is get).	Optional
	namelist	A space-separated list of variables to be submitted via HTTP along with the request to the URI specified by <i>src</i> . If an ECMAScript object is included in <i>namelist</i> , then all of its fields are submitted using corresponding names (e.g. <i>myobject.a</i> , <i>myobject.b</i> , etc.)	Optional
	next	URI to explicitly transition to. For <submit>, the URI must reference another document; to transition to another dialog within the same document, use <goto> or <subdialog>.	Required
Parents	<block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>, <prompt>		
Children	none		

4.31. Throw

Generates a system or user-defined event to catch with an event handler. Platform-defined events are for example `noinput`, `nomatch`. For these, generic platform-defined handlers are provided. Also user-defined events can be triggered using `<throw>`. The name of the event to throw is contained in the required *event* or *eventexpr* attribute.

Example:

```
<field name="choice" type="digits?length=8">
  <prompt>
    <audio src="prompts/app/enternumber.wav"/>
  </prompt>
  <filled>
    <assign name="test" expr="validateNumber(choice)"/>
    <if cond="test != true">
      <clear namelist="keuze1"/>
      <throw event="nomatch"/>
    </if>
    <return namelist="choice"/>
  </filled>
</field>
```

4.31.1. Attributes

<throw>			
Attributes	event	Name of event to throw. The complete event name will be available in the handler that catches this event via the platform-defined variable <i>_event</i> .	Required
	eventexpr	ECMAScript expression that evaluates to a name of an event to throw. The complete event name will be available in the handler that catches this event via the platform-defined variable <i>_event</i> .	Required
	message	String specifying additional contextual information about the event being thrown. The string will be available in the handler that catches this event via the platform-defined variable <i>_message</i> .	Optional
	messageexpr	ECMAScript expression that evaluates to a string specifying additional contextual information about the event being thrown. The string will be available in the handler that catches this event via the Platform-defined <i>_message</i> .	Optional
parents	<block>, <catch>, <error>, <filled>, <help>, <if>, <noinput>, <nomatch>, <prompt>		
Children	None		

4.32. Transfer

Transfers the caller to the specified telephone number. The required *dest* or *destexpr* (ECMAScript expression that evaluates to a string) attribute holds the string that specifies the number.

When transfer completes, the <filled> element contained within it is executed.

In addition, the transfer may throw an event. For the SpeechFrame platform, handlers are provided for the generic *connection* and *error* event spaces.

If a prompt tag is present within the transfer tag, the prompt is played to the caller prior to the transfer.

Example:

```
<form id="doTransfer">
  <transfer name="toCallcenter" dest="12345" bridge="true">
    <prompt>
      <audio src="prompts/app/youareconnected.wav"/>
    </prompt>
    <filled>
      <log> Transfer returned : <value expr="toCallcenter"/></log>
    </filled>
  </transfer>
</form>
```

4.32.1. Comsys enhancements on the transfer

Comsys has developed some enhancements on the transfer tag, since the standard requirements of the W3C consortium does not offer them.

4.32.1.1. Enhanced caller- and called party info

The destination address can be enriched with information as the number plan, number format, number screening and the number presentation.

These items can contain the following values:

- NumberPlan
UNKNOWN, ISDN, DATA, TELEX, NATIONAL_STANDARD, PRIVATE
- NumberFormat
UNKNOWN, INTERNATIONAL, NETWORK_SPECIFIC, SUBSCRIBER_NUMBER, ABBREVIATED_NUMBER
- NumberScreening
UNKNOWN, USER_NOT_SCREENED, USER_VERIFIED_PASSED, USER_VERIFIED_FAILED, NETWORK_PROVIDED
- NumberPresentation
UNKNOWN, ALLOWED, RESTRICTED

Example:

```
<form id="doTransfer">
  <transfer name="toCallcenter"
    dest="dnis=12345;
      NumberFormat=INTERNATIONAL;
      NumberScreening=NETWORK_PROVIDED"
    bridge="true">
    <prompt>
      <audio src="prompts/app/youareconnected.wav"/>
    </prompt>
  </transfer>
</form>
```

```
</prompt>
<filled>
  <log> Transfer returned : <value expr="toCallcenter"/></log>
</filled>
</transfer>
</form>
```

4.32.1.2. Play a prompt to the called party

If a prompt must be played to the called party instead of the calling party when phone is picked up, the audio tag within the transfer tag can be extended with the legId parameter.

Example:

```
<form id="doTransfer">
  <transfer name="toCallcenter" dest="12345" bridge="true">
    <prompt>
      <audio src="prompts/app/playtocalledparty.wav?legId=1"/>
    </prompt>
    <filled>
      <log> Transfer returned : <value expr="toCallcenter"/></log>
    </filled>
  </transfer>
</form>
```

The prompt is played as soon as the called party answers.

Only one prompt can be played to the called party at a time. Due to latency in the signalling path, it can happen that the calling party hears (a small) part of the dialogue that is started by the called party before the called party actually hears the prompt.

4.32.1.3. conferencing

The transfer tag can be used to set-up a conference call. Several types of conferences can be configured. Chapter 5 describes the conferencing in detail.

4.32.2. Attributes

<transfer>

Attributes	bridge	<p>Boolean that determines behavior once the call is connected. If "true," the underlying platform "stays on the line" (e.g. bridges, trombones, hairpins) while the transfer is under way.</p> <p>When the transfer ends (either because of a disconnection or because caller input that matches an active grammar), execution resumes with the transfer's <filled> element. Alternatively, certain conditons may throw an event.</p> <p>If "false," the interpreter throws <i>connection.disconnect.transfer</i> immediately after connecting the transfer.</p> <p>Failed transfers throw appropriate error events (default is false).</p>	Optional
	cond	ECMAScript expression that must also evaluate to true for the FIA to select this form item (default is true).	Optional
	connecttimeout	<p>Number of seconds (s) or milliseconds (ms) to wait while trying to connect the call.</p> <p>If <i>connecttimeout</i> passes, then the <i>name</i> form item variable is set to "noanswer" and the transfer's <filled> element is executed (default is platform-specific).</p>	Optional
	dest	String that specifies the destination of the transfer. VoiceXML platforms must support the "tel.//" URI syntax for phone numbers described in RFC 2806 but may support additional platform-defined syntaxes as well.	Required
	destexpr	ECMAScript expression that evaluates to a string that specifies the destination of the transfer (syntax see above).	Required
	expr	ECMAScript expression that, is specified, is evaluated and provides an initial value for the transfer's form item variable <i>name</i> (default is undefined). If a value is provided, then the form item will not be visited until/unless it is first cleared.	Optional
	maxtime	<p>If <i>bridge</i> is "true," an integer specifying the maximum time, in seconds, that the transfer is allowed to last.</p> <p>After <i>maxtime</i> elapses, the call is disconnected, <i>name</i> is set to "maxtime_disconnect", and the transfer's <filled> element is executed.</p> <p>Setting <i>maxtime</i> to 0 designates that there is no maximum time allowed (default is 0).</p>	Optional
	name	<p>Valid ECMAScript variable name that declares the form item variable for this transfer. Mostly, name is set to a status value (e.g. "busy") when the transfer completes and the <filled> is executed.</p> <p>Alternatively, some error cases throw an event rather than filling <i>name</i>.</p>	Required
parents	<form>		
children	<audio>, <catch>, <enumerate>, <error>, <filled>, <grammar>, <help>, <noinput>, <nomatch>, <prompt>, <property>, <value>		

4.33. Value

Evaluates and returns an ECMAScript expression. This element is mostly used within `<audio>` or `<prompt>` elements to introduce variable fragments in carrier-prompts played to the caller. The required *expr* attribute holds an ECMAScript expression that evaluates to a string to be inserted in the prompt.

Example:

```
<var name="phoneNumber"/>
<script>
  phoneNumber= session.telephone.ani ;
</script>
<form id="sayNumber">
  <block>
    Your phone number is <value expr="phoneNumber"/>
  </block>
</form>
```

4.33.1. Attributes

<value>			
attributes	expr	ECMAScript expression that evaluates to a string, which will be spoken as TTS to the caller.	Required
parents	<audio>, <block>, <catch>, <choice>, <enumerate>, <error>, <field>, <filled>, <help>, <if>, <initial>, <log>, <menu>, <noinput>, <nomatch>, <object>, <prompt>, <record>, <subdialog>, <transfer>, <voice>		
children	None		

4.34. Var

Declares a variable. The `<var>` element declares a variable within the scope defined by its container. A variable is accessible to all children of that container. In addition, the variable context is shared between VXML elements, *expr* expressions within VXML elements and `<script>` blocks. Variables defined in VXML using `<var>` are accessible from any `<script>` block or *expr* expression within the same scope or a child scope of where they are declared. Similarly, variables defined in `<script>` blocks are given the scope of the `<script>` block in which they occur, and are accessible within VXML elements and *expr* expressions accordingly.

`<var>` is also used at the beginning of a called subdialog to declare any variables that it expects to be passed via `<param>` from the invoking `<subdialog>`.

The required *name* attribute holds a valid ECMAScript name for a variable.

Example:

```
<form>
  <var name="voorkeur"/>
  <var name="checkdate"/>
  <var name="audioPath"/>
  <var name="markers"/>
  <var name="adinfos"/>
  <var name="language"/>
  <var name="callID"/>
  <block>
    <assign name="checkdate" expr="today()"/>
    <if cond="voorkeur != checkdate">
      <goto nextitem="keuze2"/>
    <else/>
      <goto nextitem="keuze1"/>
    </if>
  </block>
  ...
  ...
</form>
```

4.34.1. Attributes

<var>			
attributes	expr	ECMAScript expression that is evaluated and assigned to the variable <i>name</i> (default is undefined for new variables or the current value if already declared).	Optional
	name	Valid ECMAScript variable name - it can contain letters, digits, underscores and dollar signs but the first character must be a letter or a dollar sign.	Required
parents	<block>, <catch>, <error>, <filled>, <form>, <help>, <if>, <noinput>, <nomatch>, <prompt>, <vxml>		
children	none		

4.35. VXML

Represents the root document element of a VoiceXML document; contains all the other elements of a VXML document. The required *version* attribute specifies the VXML version number; the current version number is "2.0".

Example:

```
<?xml version='1.0'?>
<VXML version="2.0" application="app.vxml" >
  <form id="welcome">
    <block>
      ....
      ....
    </block>
  </form>
</VXML>
```

4.35.1. Attributes

<vxml>			
attributes	application	URI specifying the application root document for this VXML document. If specified, the root document is loaded. Default is none.	Optional
	base	URI that, if specified, causes the interpreter to prepend base to all relative URIs in the document before evaluating them.	Optional
	version	VXML version number. Current: "2.0".	Required
	xml:lang	Specifies the language and locale identifier of the contained or referenced grammar according to RFC 1766 (e.g. "fr-CA" for Canadian French). If omitted, the value is inherited down from the document hierarchy, and ultimately a VoiceXML interpreter default.	Optional
parents	none		
children	<catch>, <data>, <error>, <form>, <help>, <link>, <menu>, <meta>, <noinput>, <nomatch>, <property>, <script>, <var>		

5. Conferencing

A conference can be set-up using the VoiceXML transfer tag.

```
<form id="welkom">  
  <transfer name="conference" bridge="false" dest="<conference address>"/>  
</form>
```

In general there are two types of conference: simple and complex. This chapter describes how to set-up each of them.

5.1. conference address

The conference address contains the following items:

- telephone number
- box : the conference box name
- mode : the mode, this can be full, listen or talk
- exit: a DTMF that has to be pressed to leave the conference.
- Application : An application name

Conference boxes are created and destroyed automatically.

5.2. A simple conference

Simple conferences have two or more applicants in a box without the possibility to record the conference or play back some prompt in the conference

The following example will:

- Place the applicant in box 'CB123'
- Allow each applicant to speak and to listen
- Leave the conference by pressing a '7'

```
<form id="conference">  
  <transfer name="theConference" bridge="false"  
    dest="'000?box=CB123;mode=full;exit=7'">  
  </transfer>  
</form>
```

5.3. A complex conference

Complex conferences have one or two dummy Voice-XML sessions started which can be instructed to playback a prompt, record the conference etc.

The following example will:

- Place the applicant in box 'CB123'
- Allow each applicant to speak and to listen
- Leave the conference by pressing a '7'
- Start a dummy Voice-XML session for the (fake) DDI 4567

This enables recording the dialogue OR play back prompts in the dialogue.

```
<form id="conference">  
  <transfer name="theConference" bridge="false"  
    dest="'0000?box=CB123;mode=full;exit=7;application=4567">  
  </transfer>  
</form>
```

The following example will:

- Place the applicant in box 'CB123'
- Allow each applicant to speak and to listen
- Leave the conference by pressing a '7'
- Start a dummy Voice-XML session for the (fake) DDI 4567
- Start an inbound Voice-XML session for the dummy outbound 1289

This enables recording the dialogue AND play back prompts in the dialogue at one and the same time.
For best flexibility, these dialogues are instructed through use of a database.

```
<form id="conference">  
  <transfer name="theConference" bridge="false"  
    dest="'1289?box=CB123;mode=full;exit=7;application=4567">  
  </transfer>  
</form>
```

Note:

The extra Voice-XML session(s) is (are) terminated when either a session itself terminates or the last applicant leaves the conference.

5.4. Example

Paragraph 5.4.1 shows an example of a conference application. This application asks for the box number you want to participate, and then connect you to it. Paragraph 5.4.2 shows the root document for this application.

5.4.1. Application listing

```
<?xml version='1.0'?>
<vxml version="2.0" application="root_doc.vxml">
  <var name="firstTime" expr="true"/>
  <var name="boxNumber"/>

  <form id="main">
    <field name="givenNumber" type="digits">
      <prompt cond="firstTime==true">
        <audio src="../../prompts/welcome.wav"/>
      </prompt>
      <prompt>
        <audio src="../../prompts/enternumber.wav"/>
      </prompt>
      <filled>
        <assign name="firstTime" expr="false"/>
        <assign name="boxNumber" expr="givenNumber"/>
        <goto next="#conferenceTransfer"/>
      </filled>
      <noinput>
        <assign name="firstTime" expr="false"/>
      </noinput>
      <nomatch>
        <assign name="firstTime" expr="false"/>
      </nomatch>
    </field>
  </form>

  <form id="conferenceTransfer">
    <transfer name="theConference" bridge="false" destexpr="'4031?box=' + boxNumber +
';mode=full;exit=7;application=4030'">
      <prompt>
        <audio src="../../prompts/uwillbeconnected.wav"/>
      </prompt>
      <filled>
        <prompt>
          <audio src="../../prompts/thanks.wav"/>
        </prompt>
      </filled>
    </transfer>
    <block>
      <disconnect/>
    </block>
  </form>
</vxml>
```

5.4.2. Root document listing

```
<?xml version="1.0"?>
<!DOCTYPE vxml SYSTEM "/voicexml2-0.dtd">
<vxml version="2.0">
  <property name="bargein" value="true"/>
  <property name="timeout" value="5s"/>
  <property name="interdigittimeout" value="5s"/>
  <property name="termtimeout" value="5"/>
  <property name="termchar" value="#" />

  <catch event="telephone.disconnect.hangup">
    <throw event="connection.disconnect.hangup"/>
    <exit/>
  </catch>

  <catch event="connection.disconnect.hangup">
    <log>
      "Hangup detected"
    </log>
    <exit/>
  </catch>

  <error>
    <log>
      "Error detected"
    </log>
    <disconnect/>
  </error>
  <noinput>
    <reprompt/>
  </noinput>
  <noinput count="3">
    <disconnect/>
  </noinput>
  <nomatch>
    <reprompt/>
  </nomatch>
  <nomatch count="3">
    <disconnect/>
  </nomatch>
</vxml>
```

6. Pre-call Announcements

Pre-call Announcements lets you execute a dialogue before the call is connected. The call will not be charged, and it enables database lookups, application initialisation, portal issues etc. to take place. Until the call is connected, the caller will hear a ringtone.

To enable Pre-call Announcements the connectOn parameter in the SpeechFrame configuration should be set to "Application" (normally it would be set to "IncommingCall"). Refer to the Speechframe documentation how to set this parameter.

Secondly a separate VoiceXML dialogue must be created for all documents that should be used in the pre-call phase. In the pre-call phase the dialogue must have the property 'delayedInboundConnect' set to "true"

Note:

Due to the way how VXML-interpreters work, care should be taken that the pre-call part is executed (the dummy field ensures this) before the 'connect' is sent as can be seen in the following example:

```
<?xml version='1.0'?>
<!DOCTYPE vxml SYSTEM "/voicexml2-0.dtd">
<vxml version="2.0" application="../root_doc.vxml">
  <property name="delayedInboundConnect" value="true"/>
  <form id="preCall">
    <field name="dummy" type="digits">
      <prompt timeout="1s">
        <audio src="../prompts/tune.wav"/>
      </prompt>
      <noinput>
        <goto next="mainDialogue.vxml"/>
      </noinput>
      <filled>
        <goto next="mainDialogue.vxml"/>
      </filled>
    </field>
  </form>
</vxml>
```

If the property 'delayedInboundConnect' is not set at all, or it is set to "false", the dialogue continues in a connected phase.

6.1. Sending a specific clearing cause

When running a pre-call dialogue it can be useful to disconnect the (incoming) call with a specific clearing cause out of:

- NORMAL
- NUMBER_BUSY
- NO_ANSWER
- NUMBER_UNOBTAINABLE
- NUMBER_CHANGED
- OUT_OF_ORDER
- INCOMING_CALLS_BARRED
- CALL_REJECTED
- CALL_FAILED
- CHANNEL_BUSY
- NO_CHANNELS
- CONGESTION

```
<?xml version='1.0'?>
<!DOCTYPE vxml SYSTEM "/voicexml2-0.dtd">
<vxml version="2.0" application="../root_doc.vxml">
  <property name="delayedInboundConnect" value="true"/>
  <property name="clearingCause" value="NUMBER_UNOBTAINABLE"/>

  <form id="preCall">
    <field name="dummy" type="digits">
      <prompt timeout="1s">
        <audio src="../prompts/tune.wav"/>
      </prompt>
      <noinput>
        <disconnect/>
      </noinput>
      <filled>
        <disconnect/>
      </filled>
    </field>
  </form>
</vxml>
```

7. Audio prompting

7.1. Audio prompting

In order to play pre-recorded messages to the user during the dialogue the application needs to dispose of a (list of) audio files containing these messages and the VXML document needs to refer to the appropriate audio files in the correct way. Usually, the audio files have numerical names. This is practical for automated selection of audio files (indices), especially for concatenation of prompts for numbers, dates etc. In the following subsection, 6.1.1., the available SpeechFrame concatenation scripts are presented. These scripts are called *apf* = automatic pronunciation function.

Referencing of audio files in a VXML document is done by means of the `<audio>` element, in which the value of the *src* or *expr* attribute defines the url of the audiofile. Example:

```
<audio expr="audioPath + 'std/121.wav'"/>
```

IMPORTANT! The `<audio>` element within a `<field>` or `<menu>` element always has to be embedded in a `<prompt>` element! Example:

```
<prompt>  
  <audio expr="audioPath + 'std/100.wav'"/>  
</prompt>
```

A list of all prompts used by the application, with the verbatim text of the message and the name (=the number) of the appropriate audiofile is documented in the [prompt list](#), an obligatory section of each [Functional Design](#) of an application.

The audio format of sound files in SpeechFrame is: wave, 8-bits, 8 KHz, mono, A-law or μ -law.

7.1.1. Concatenation

Concatenation (in our specific usage) means automatic composition of complex prompts, that is the "pasting" of different prompts into one fluent and natural sounding utterance. Comsys has developed a special product for the pronunciation of numbers, currency, phone numbers, floats etc, called APF. APF stands for Automatic Pronunciation Functions.

For more information see the document "APF Manual v2.0.doc"

7.1.2. Structure of the prompts directory

The audio files used by SF applications are grouped and filed in corresponding directories and subdirectories in order to make automatic prompt selection straightforward and suitable for scripting. The directory structure is presented in Figure 2.

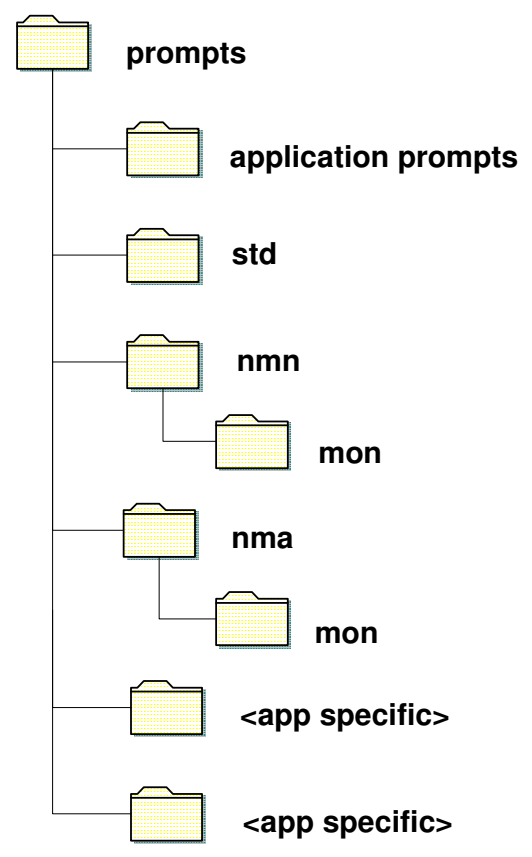


Figure 1

- application prompts
 This subdirectory has usually the name of the application, so the directory is called *<application name>* or an abbreviation thereof. This is the place for all prompts that are used for the dialogue flow and the control and navigation aspects of the application. These can be menus ("For registration press 1, for more information press 2"), utterances prompting the caller to enter specific input ("Please enter your date of birth.") or messages providing information ("Your registration number is printed on your membership card and consists of 8 digits.").

- std
 The *std* directory contains the group of the so-called standard or system prompts. These are announcements or messages to the caller which are used in handling standard situations, such as starting and finishing the dialogue ("Thank you for calling and have a nice day!"), system connected events, errors or malfunctions ("We haven't received correct input."; "Sorry, the system is not available right now."), and various "building blocks" for concatenated prompts, such as "For..." "... please press..." etc.

- nmn

This directory holds all numeral prompts pronounced with neutral (flat) intonation. This means that a numeral is pronounced in such a way that another numeral or another word can follow it making the concatenated result sound natural and fluent. This is why this intonation is often called "continuing intonation". In the *mon* subdirectory all utterances are filed which are used for concatenation of dates: names of months and weekdays and in some languages (like English and German) also the ordinal numbers needed for correct date pronunciation ("the first of...").

- nma

This directory (again with a *mon* subdirectory for date prompts) holds all numeral prompts pronounced with final (falling) intonation. This means that the numeral is pronounced with a falling intonation which sounds naturally at the end of utterances.

- <application specific>

To the prompts directory any further application specific subdirectories can be added. Various kinds of prompts can be stored here, for example names of streets or cities, full addresses, names of products, stocks, motorway numbers etc. When the prompt numbering system is thought-out well, own scripts can be designed for automatic prompt selection and concatenation. Various messages to the caller can be calculated automatically based on the input, either from direct caller input (feed-back and verification) or from a database query (information to the user).