| Version | Date | Adjustments |
|---|---|---|
| A | 2006-10-05 | First revision (ERMKESE). |

GREEN color means supported.

RED color means unsupported.

# Voice Extensible Markup Language (VoiceXML) 2.1

## W3C Working Draft 15 September 2006

**This version:**
http://www.w3.org/TR/2006/WD-voicexml21-20060915/
**Latest version:**
http://www.w3.org/TR/voicexml21/
**Previous version:**
http://www.w3.org/TR/2005/CR-voicexml21-20050613/
**Editors:**
Matt Oshry, Tellme Networks (*Editor-in-Chief*)
RJ Auburn, Voxeo Corporation
Paolo Baggia, Loquendo
Michael Bodell, Tellme Networks
David Burke, Voxpilot Ltd.
Daniel C. Burnett, Nuance Communications
Emily Candell, Comverse
Jerry Carter, Nuance Communications
Scott McGlashan, Hewlett-Packard
Alex Lee, Genesys
Brad Porter, Tellme Networks
Ken Rehor, Invited Expert



---

## Abstract

*VoiceXML 2.1* specifies a set of features commonly implemented by Voice Extensible Markup Language platforms. This specification is designed to be fully

backwards-compatible with VoiceXML 2.0 [VXML2]. This specification describes only the set of additional features.

## Status of this Document

*This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.*

This is the 15 September 2006 W3C Last Call Working Draft of "VoiceXML Version 2.1". The Last Call period ends on 6 October 2006.

Following the publication of this specification as Candidate Recommendation, substantial comments were received, and in accordance with section 7.4.6 of the W3C Process, the Director returned the Voice Browser Working Group for further work. This work resulted in significant changes to the specification (see **6 Using <foreach> to Concatenate Prompts and Loop through Executable Content** ), which led to a return as Last Call Working Draft. A detailed list of changes since the last published draft is available in section **G.2 Summary of changes since the Candidate Recommendation**.

The Voice Browser Working Group believes that this specification addresses its requirements and all previous Last Call and Candidate Recommendation issues (see the Disposition of Comments document).

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

This document has been produced as part of the Voice Browser Activity (activity statement), following the procedures set out for the W3C Process. The authors of this document are members of the Voice Browser Working Group *(W3C Members only)*.

This document is for public review, and comments and discussion are welcomed on the (archived) public mailing list <www-voice@w3.org>.

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

The sections in the main body of this document are normative unless otherwise specified. The appendices in this document are informative unless otherwise indicated explicitly.

## Table of Contents



### Appendices



---

# 1 Introduction

The popularity of VoiceXML 2.0 [VXML2] spurred the development of numerous voice browser implementations early in the specification process. [VXML2] has been phenomenally successful in enabling the rapid deployment of voice applications that handle millions of phone calls every day. This success has led to the development of additional, innovative features that help developers build even more powerful voice-activated services. While it was too late to incorporate these additional features into [VXML2], the purpose of VoiceXML 2.1 is to formally specify the most common features to ensure their portability between platforms and at the same time maintain complete backwards-compatibility with [VXML2].

This document defines a set of 8 commonly implemented additional features to VoiceXML 2.0 [VXML2].

## 1.1 Terminology

In this document, the key words "must", "must not", "required", "shall", "shall not", "should", "should not", "recommended", "may", and "optional" are to be interpreted as described in [RFC2119] and indicate requirement levels for compliant VoiceXML implementations.

## 1.2 Elements Introduced or Enhanced in VoiceXML 2.1

The following table lists the elements that have been introduced or enhanced in VoiceXML 2.1.

| Element | Purpose | Section | New/Enhanced |
|---|---|---|---|
| <data> | Fetches arbitrary XML data from a document server. | 5 | New |
| <disconnect> | Disconnects a session. | 8 | Enhanced |
| <grammar> | References a speech recognition or DTMF grammar. | 2 | Enhanced |
| <foreach> | Iterates through an ECMAScript array. | 6 | New |
| <mark> | Declares a bookmark in a sequence of prompts. | 4 | Enhanced |
| <property> | Controls platform settings. | 5.1, 7 | Enhanced |
| <script> | References a document containing client-side ECMAScript. | 3 | Enhanced |
| <transfer> | Transfers the user to another destination. | 9 | Enhanced |

Table 1: Elements Introduced or Enhanced in VoiceXML 2.1

## 2 Referencing Grammars Dynamically

As described in section 3.1 of [VXML2], the <grammar> element allows the specification of a speech recognition or DTMF grammar. VoiceXML 2.1 extends the <grammar> element to support the following additional attribute:

| | |
|---|---|
| **srcexpr** | Equivalent to src, except that the URI is dynamically determined by evaluating the given ECMAScript expression in the current scope (e.g. the current form item). The expression must be evaluated each time the grammar needs to be activated. If srcexpr cannot be evaluated, an *error.semantic* event is thrown. |

Table 1: <grammar> Attributes

Exactly one of "src", "srcexpr", or an inline grammar must be specified; otherwise, an *error.badfetch* event is thrown.

The following example demonstrates capturing a street address. The first field requests a country, the second field requests a city, and the third field requests a street. The grammar for the second field is selected dynamically using the result of the country field. The grammar for the third field is selected dynamically using the result of the city field.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
  version="2.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
  http://www.w3.org/TR/voicexml21/vxml.xsd">

  <form id="get_address">
    <field name="country">
      <grammar type="application/srgs+xml" src="country.grxml"/>
      <prompt>Say a country.</prompt>
    </field>

    <field name="city">
      <grammar type="application/srgs+xml" srcexpr="country +
'/cities.grxml"/>
      <prompt>What city in <value expr="country"/>.</prompt>
    </field>

    <field name="street">
      <grammar type="application/srgs+xml" srcexpr="country + '/' +
city + '/streets.grxml'"/>
      <prompt> What street in <value expr="city"/> are you looking
for? </prompt>
    </field>

    <filled>
      <prompt>
```

```
       You chose
       <value expr="street"/>
       in
       <value expr="city"/>
       <value expr="country"/>
      </prompt>
      <exit/>
    </filled>
  </form>
</vxml>
```

# 3 Referencing Scripts Dynamically

As described in section 5.3.12 of [VXML2], the <script> element allows the specification of a block of ECMAScript [ECMA] client-side scripting language code, and is analogous to the [HTML4] <SCRIPT> element. VoiceXML 2.1 extends the <script> element to support the following additional attribute:

| | |
|---|---|
| **srcexpr** | Equivalent to src, except that the URI is dynamically determined by evaluating the given ECMAScript expression. The expression must be evaluated each time the script needs to be executed. If srcexpr cannot be evaluated, an *error.semantic* event is thrown. |

Table 2: <script> Attributes

Exactly one of "src", "srcexpr", or an inline script must be specified; otherwise, an *error.badfetch* event is thrown.

The following example retrieves a script from a URI that is composed at run-time using the variable *scripts_baseuri*.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
  version="2.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
  http://www.w3.org/TR/voicexml21/vxml.xsd">

  <var name="scripts_baseuri" expr="'http://www.example.org/'"/>

  <form>
    <script srcexpr="scripts baseuri + 'lib/util.js'"/>
  </form>
</vxml>
```

# 4 Using <mark> to Detect Barge-in During Prompt Playback

As described in section 3.3.2 of [SSML], the <mark> element places a marker into the text/tag sequence. An SSML processor must either allow the VoiceXML interpreter to retrieve or must inform the interpreter when a <mark> is executed during audio output.

[SSML] defines a single attribute, name, on the <mark> element, allowing the programmer to name the mark. VoiceXML 2.1 extends the <mark> element to support the following additional attribute:

| | |
|---|---|
| **nameexpr** | An ECMAScript expression which evaluates to the name of the mark when the prompt is queued. If nameexpr cannot be evaluated, an *error.semantic* event is thrown. |

Table 3: <mark> Attributes

Exactly one of "name" and "nameexpr" must be specified; otherwise, an *error.badfetch* event is thrown.

As described in section 4.1.1 of [VXML2], the <mark> element is permitted in conforming VoiceXML documents, but [VXML2] does not specify a standard way for VoiceXML processors to access <mark> element information. Processors of conforming VoiceXML 2.1 documents must set the following two properties on the application.lastresult$ object whenever the application.lastresult$ object is assigned (e.g. a <link> is matched) and a <mark> has been executed.

| | |
|---|---|
| **markname** | The name of the mark last executed by the SSML processor before barge-in occurred or the end of audio playback occurred. If no mark was executed, this variable is undefined. |
| **marktime** | The number of milliseconds that elapsed since the last mark was executed by the SSML processor until barge-in occurred or the end of audio playback occurred. If no mark was executed, this variable is undefined. |

Table 4: <mark>-related application.lastresult$ properties

When these properties are set on the application.lastresult$ object, if an input item (as defined in section 2.3 of [VXML2]) has also been filled and has its shadow variables assigned, the interpreter must also assign *markname* and *marktime* shadow variables, the values of which equal the corresponding properties of the application.lastresult$ object.

When a <mark> is executed during the flushing of the prompt queue as a result of encountering a fetchaudio, the application.lastresult$ object should not be modified and no shadow variables should be assigned.

The following example establishes marks at the beginning and at the end of an advertisement. In the <filled>, the code checks which mark, if any, was last executed when bargein occurred. If the "ad_start" mark is executed but "ad_end" is not, the code checks that at least 5 seconds of the advertisement has been played and sets the played_ad variable appropriately.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
  version="2.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
  http://www.w3.org/TR/voicexml21/vxml.xsd">

  <var name="played_ad" expr="false"/>

  <form>
    <field name="team">
      <prompt>
        <mark name="ad_start"/>
        Baseball scores brought to you by Elephant Peanuts.
        There's nothing like the taste of fresh roasted peanuts.
        Elephant Peanuts. Ask for them by name.
        <mark name="ad_end"/>
        <break time="500ms"/>
        Say the name of a team. For example, say Boston Red Sox.
      </prompt>

      <grammar type="application/srgs+xml" src="teams.grxml"/>

      <filled>
        <if cond="typeof(team$.markname) == 'string' &amp;&amp;
           (team$.markname=='ad_end' ||
           (team$.markname=='ad_start' &amp;&amp;
              team$.marktime &gt;= 5000))">
          <assign name="played_ad" expr="true"/>
        <else/>
          <assign name="played_ad" expr="false"/>
        </if>
      </filled>

    </field>
  </form>

</vxml>
```

## 5 Using <data> to Fetch XML Without Requiring a Dialog Transition

The <data> element allows a VoiceXML application to fetch arbitrary XML data from a document server without transitioning to a new VoiceXML document. The XML data fetched by the <data> element is bound to ECMAScript through the

named variable that exposes a read-only subset of the W3C Document Object Model (DOM).

Attributes of <data> are:

| | |
|---|---|
| **src** | The URI specifying the location of the XML data to retrieve. |
| **name** | The name of the variable that exposes the DOM. |
| **srcexpr** | Like src, except that the URI is dynamically determined by evaluating the given ECMAScript expression when the data needs to be fetched. If srcexpr cannot be evaluated, an *error.semantic* event is thrown. |
| **method** | The request method: get (the default) or post. |
| **namelist** | The list of variables to submit. By default, no variables are submitted. If a namelist is supplied, it may contain individual variable references which are submitted with the same qualification used in the namelist. Declared VoiceXML and ECMAScript variables can be referenced. |
| **enctype** | The media encoding type of the submitted document. The default is application/x-www-form-urlencoded. Interpreters must also support multipart/form-data [RFC2388] and may support additional encoding types. |
| **fetchaudio** | See Section 6.1 of [VXML2]. This defaults to the fetchaudio property described in Section 6.3.5 of [VXML2]. |
| **fetchhint** | See Section 6.1 of [VXML2]. This defaults to the datafetchhint property described in Section 5.1. |
| **fetchtimeout** | See Section 6.1 of [VXML2]. This defaults to the fetchtimeout property described in Section 6.3.5 of [VXML2]. |
| **maxage** | See Section 6.1 of [VXML2]. This defaults to the datamaxage property described in Section 5.1. |
| **maxstale** | See Section 6.1 of [VXML2]. This defaults to the datamaxstale property described in Section 5.1. |

Table 5: <data> Attributes

Exactly one of "src" or "srcexpr" must be specified; otherwise, an *error.badfetch* event is thrown. If the content cannot be retrieved, the interpreter throws an error as specified for fetch failures in Section 5.2.6 of [VXML2].

Platforms should support parsing XML data into a DOM. If an implementation does not support DOM, the name attribute must not be set, and any retrieved content must be ignored by the interpreter. If the name attribute is present, these implementations will throw *error.unsupported.data.name*.

If the name attribute is present, and the returned document is XML as identified by [RFC3023], the VoiceXML interpreter must expose the retrieved content via a read-only subset of the DOM as specified in Appendix D. An interpreter may support additional data formats by recognizing additional media types. If an interpreter receives a document in a data format that it does not understand, or the data is not well-formed as defined by the specification of that format, the interpreter throws *error.badfetch*. If the media type of the retrieved content is one of those defined in [RFC3023] but the content is not well-formed XML, the interpreter throws *error.badfetch*.

Like the <var> element, the <data> element can occur in executable content or as a child of <form> or <vxml>. In addition, it shares the same scoping rules as the <var> element. If a <data> element has the same name as a variable already declared in the same scope, the variable is assigned a reference to the DOM exposed by the <data> element.

If use of the DOM causes a DOMException to be thrown, but the DOMException is not caught by an ECMAScript exception handler, the VoiceXML interpreter throws *error.semantic*.

Like the <submit> element, when an ECMAScript variable is submitted to the server its value is first converted into a string before being submitted. If the variable is an ECMAScript Object the mechanism by which it is submitted is not currently defined. If a <data> element's namelist contains a variable which references recorded audio but does not contain an *enctype* of multipart/form-data [RFC2388], the behavior is not specified. It is probably inappropriate to attempt to URL-encode large quantities of data.

If the value of the *src* or *srcexpr* attribute includes a fragment identifier, the processing of that fragment identifier is platform-specific.

In the examples that follow, the XML document fetched by the <data> element is in the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<quote xmlns="http://www.example.org">
  <ticker>F</ticker>
  <name>Ford Motor Company</name>
  <change>1.00</change>
  <last>30.00</last>
</quote>
```

The following example assigns the value of the "last" element to the ECMAScript variable "price":

```
<data name="quote" src="quote.xml"/>
<script><![CDATA[
```

```
  var price =
quote.documentElement.getElementsByTagNameNS("http://www.example.org"
, "last").item(0).firstChild.data;
]]></script>
```

The data is fetched when the <data> element is executed according to the caching rules established in Section 6.1 of [VXML2].

Before exposing the data in an XML document referenced by the <data> element via the DOM, the interpreter should check that the referring document is allowed to access the data. If access is denied the interpreter must throw *error.noauthorization*.

**Note:**

One strategy commonly implemented in voice browsers to control access to data is the "access-control" processing instruction described in the WG Note: Authorizing Read Access to XML Content Using the <?access-control?> Processing Instruction 1.0 [DATA_AUTH].

The following example retrieves a stock quote in one dialog, caches the DOM in a variable at document scope, and uses the DOM to playback the quote in another dialog.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
  version="2.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
  http://www.w3.org/TR/voicexml21/vxml.xsd">
  <var name="quote"/>
  <var name="ticker" expr="'f'"/>

  <form id="get quote">
     <block>
        <data name="quote"
srcexpr="'http://www.example.org/getquote?ticker=' + ticker"/>
        <assign name="document.quote" expr="quote.documentElement"/>
        <goto next="#play_quote"/>
     </block>
  </form>

  <form id="play_quote">

     <script><![CDATA[
     // retrieve the value contained in the node t from the DOM
exposed by d
     function GetData(d, ns, t, nodata)
     {
        try {
           return d.getElementsByTagNameNS(ns,
t).item(0).firstChild.data;
```

```
        }
        catch(e)
        {
           // the value could not be retrieved, so return this
instead
           return nodata;
        }
     }
     ]]></script>

     <block>
        <!-- retrieve the change in the stock's value -->
        <var name="change" expr="GetData(quote,
'http://www.example.org', 'change', 0)"/>
        <var name="last" expr="GetData(quote,
'http://www.example.org', 'last', 0)"/>
        <var name="last_parts" expr="last.split('.')"/>

        <!--play the company name -->
        <audio expr="ticker + '.wav'"><value expr="GetData(quote,
'http://www.example.org', 'name', 'unknown')"/></audio>
        <!-- play 'unchanged, 'up', or 'down' based on zero,
positive, or negative change -->
        <if cond="change == 0">
           <audio src="unchanged at.wav"/>
        <else/>
           <if cond="change &gt; 0">
              <audio src="up.wav"/>
           <else/> <!-- negative -->
              <audio src="down.wav"/>
           </if>
           <audio src="by.wav"/>
           <!-- play change in value as positive number -->
           <audio expr="Math.abs(change) + '.wav'"><value
expr="Math.abs(change)"/></audio>
           <audio src="to.wav"/>
        </if>
        <!-- play the current price per share -->
        <audio expr="last_parts[0] + '.wav'"><value
expr="last parts[0]"/></audio>
        <if cond="Number(last_parts[1]) &gt; 0">
           <audio src="point.wav"/>
           <audio expr="last_parts[1] + '.wav'"><value
expr="last_parts[1]"/></audio>
        </if>
     </block>
  </form>
</vxml>
```

## 5.1 <data> Fetching Properties

These properties pertain to documents fetched by the <data> element.

| | |
|---|---|
| **datafetchhint** | Tells the platform whether or not data documents may be pre- |

| | |
|---|---|
| | fetched. The value is either prefetch (the default), or safe. |
| **datamaxage** | Tells the platform the maximum acceptable age, in seconds, of cached documents. The default is platform-specific. |
| **datamaxstale** | Tells the platform the maximum acceptable staleness, in seconds, of expired cached data documents. The default is platform-specific. |

Table 6: <data> Fetching Properties

# 6 Using <foreach> to Concatenate Prompts and Loop through Executable Content

The <foreach> element allows a VoiceXML application to iterate through an ECMAScript array and to execute the content contained within the <foreach> element for each item in the array. The <foreach> element may appear within executable content and within <prompt> elements. Within executable content, except within a <prompt>, the <foreach> element may contain any elements of executable content; this introduces basic looping functionality by which executable content may be repeated for each element of an array. When <foreach> appears within a <prompt> element, it may contain only those elements valid within <enumerate> (i.e. the same elements allowed within <prompt> less <meta>, <metadata>, and <lexicon>); this allows for sophisticated concatenation of prompts as illustrated in this section.

Attributes of <foreach> are:

| | |
|---|---|
| **array** | An ECMAScript expression that must evaluate to an ECMAScript array (i.e. the result of the expression must satisfy instanceof(Array) in ECMAScript); otherwise, an *error.semantic* event is thrown. Note that the <foreach> element operates on a shallow copy of the array specified by the array attribute. |
| **item** | The variable that stores each array item upon each iteration of the loop. A new variable will be declared if it is not already defined within the parent's scope. |

Table 7: <foreach> Attributes

Both "array" and "item" must be specified; otherwise, an *error.badfetch* event is thrown.

The iteration process starts from an index of 0 and increments by one to an index of array_name.length - 1, where array_name is the name of the shallow copied array operated on by the <foreach> element. For each index, a shallow copy or reference to the corresponding array element is assigned to the item variable (i.e. <foreach> assignment is equivalent to item = array_name[index] in ECMAScript);

the assigned value could be undefined for a sparse array. VoiceXML 2.1 does not provide break functionality to interrupt a <foreach>.

The following example calls a user-defined function GetMovieList that returns an ECMAScript array. The array is assigned to the variable named 'prompts'. Upon entering the <field>, if a noinput or a nomatch event occurs, the VoiceXML interpreter reprompts the user by executing the second <prompt>. The second <prompt> executes the <foreach> element by iterating through the ECMAScript array 'prompts' and assigning each array element to the variable 'thePrompt'. Upon each iteration of the <foreach>, the interpreter executes the contained <audio> and <break> elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
  version="2.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
  http://www.w3.org/TR/voicexml21/vxml.xsd">

  <script src="movies.js"/>

  <form id="pick movie">

    <!--
    GetMovieList returns an array of objects
      with properties audio and tts.
      The size of the array is undetermined until runtime.
    -->
    <var name="prompts" expr="GetMovieList()"/>

    <field name="movie">
      <grammar type="application/srgs+xml" src="movie_names.grxml"/>

      <prompt>Say the name of the movie you want.</prompt>

      <prompt count="2">
        <audio src="prelist.wav">When you hear the name of the movie
you want, just say it.</audio>
        <foreach item="thePrompt" array="prompts">
          <audio expr="thePrompt.audio"><value
expr="thePrompt.tts"/></audio>
          <break time="300ms"/>
        </foreach>
      </prompt>

      <noinput>
        I'm sorry. I didn't hear you.
        <reprompt/>
      </noinput>

      <nomatch>
        I'm sorry. I didn't get that.
        <reprompt/>
      </nomatch>
```

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

The following is a contrived implementation of the user-defined GetMovieList function:

```
function GetMovieList()
{
  var movies = new Array(3);
  movies[0] = new Object();
  movies[0].audio = "godfather.wav"; movies[0].tts = "the godfather";
  movies[1] = new Object();
  movies[1].audio = "high_fidelity.wav"; movies[1].tts = "high
fidelity";
  movies[2] = new Object();
  movies[2].audio = "raiders.wav"; movies[2].tts = "raiders of the
lost ark";

  return movies;
}
```

When the interpreter queues the second <prompt>, it expands the <foreach> element in the previous example to the following:

```
<audio src="godfather.wav">the godfather</audio>
<break time="300ms"/>
<audio src="high fidelity.wav">high fidelity</audio>
<break time="300ms"/>
<audio src="raiders.wav">raiders of the lost ark</audio>
<break time="300ms"/>
```

The following example combines the use of the <mark> and <foreach> elements to more precisely identify which item in a list of movies the user has selected. During each iteration of the <foreach>, the interpreter stores the current item in the array to the variable *movie*. Upon execution of the <mark> element, the name of the <mark> is set to the current value of the variable movie_idx which is incremented during each iteration of the loop .

In the <filled>, if the form item variable *mov* is set to the value "more", indicating the user's desire to hear more detail about the movie name that was played when the user barged in, the code retrieves the index of the desired movie in the *movies* array from *mov$.markname*. If barge-in occurred within twenty milliseconds of the mark's execution, the code retrieves the index of the preceding movie under the assumption that the user was slow to react.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
  version="2.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
```

```
  http://www.w3.org/TR/voicexml21/vxml.xsd">

  <script src="movies.js"/>

  <form id="list_movies">
    <!-- fetch the current list of movies, and transform them into an
array -->
    <data name="domMovies" src="getmovies.cgi"/>
    <var name="movies" expr="GetMovieList(domMovies)"/>
    <var name="movie_idx" expr="0"/>
    <var name="movie_id"/>

    <field name="mov">
      <prompt>
        Say the name of the movie.
      </prompt>

      <prompt count="2">
        Here's the list of movies.
        To hear more about a movie, say 'tell me more'.
        <break time="500ms"/>
        <foreach item="movie" array="movies">
          <mark nameexpr="movie_idx++"/>
          <audio expr="movie.audio"><value expr="movie.tts"/></audio>
          <break time="500ms"/>
        </foreach>
      </prompt>

      <grammar type="application/srgs+xml" src="more.grxml"/>
      <grammar type="application/srgs+xml" src="movies.grxml"/>

      <catch event="nomatch">
        Sorry. I didn't get that.
        <reprompt/>
      </catch>

      <catch event="noinput">
        <reprompt/>
      </catch>

      <filled>
        <if cond="'more' == mov">
          <!-- user wants more detail -->
          <!-- assume no mark was executed -->
          <assign name="movie idx" expr="0"/>
          <if cond="mov$.markname != undefined">
            <!-- alas, a mark was executed, so adjust movie_idx -->
            <if cond="mov$.marktime &lt;= 20">
              <!-- returns the id of the previous movie (or the
first) -->
              <assign name="movie_idx"
                expr="(mov$.markname &lt;= 1 ? 0 : mov$.markname-
1)"/>
            <else/>
              <assign name="movie_idx" expr="mov$.markname"/>
            </if>
          </if>
```

```
          <assign name="movie id" expr="movies[movie idx].id"/>
        <else/>
          <!-- user said a specific movie -->
          <assign name="movie_id" expr="mov"/>
        </if>
      </filled>
    </field>

    <!--
     Given a movie id, fetch detail, and play it back.
     detail could be in a single wav file or in multiple files.
    -->
    <block name="play detail">
      <!-- GetMovieDetail returns an array of objects with properties
audio and tts. -->
      <var name="details" expr="GetMovieDetail(domMovies,
movie_id)"/>
      <foreach item="chunk" array="details">
        <audio expr="chunk.audio"><value expr="chunk.tts"/></audio>
      </foreach>
    </block>

  </form>
</vxml>
```

The following is a contrived XML document returned by getmovies.cgi. The user-defined GetMovieList and GetMovieDetail functions below are implemented to manipulate the DOM representation of its structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<movies xmlns="http://www.example.org">
  <movie id="m0010">
    <title>
      <tts>the godfather</tts>
      <wav>godfather.wav</wav>
    </title>
    <details>
    <detail>
      <wav>directors/directed by.wav</wav>
      <tts>directed by</tts>
    </detail>
    <detail>
      <wav>directors/coppola.wav</wav>
      <tts>francis ford coppola</tts>
    </detail>
    <detail>
      <wav>ratings/rated.wav</wav>
      <tts>rated</tts>
    </detail>
    <detail>
      <wav>ratings/r.wav</wav>
      <tts>r</tts></detail>
    <detail>
      <wav>pauses/300.wav</wav>
      <tts/>
    </detail>
```

```
    <detail>
      <wav>synopsis/m0010.wav</wav>
      <tts>a coming of age story about a nice italian boy</tts>
    </detail>
    </details>
  </movie>

  <movie id="m0052">
    <title>
      <tts>high fidelity</tts>
      <wav>high_fidelity.wav</wav>
    </title>
    <details>
      <!-- details omitted for the sake of brevity -->
    </details>
  </movie>

  <movie id="m0027">
   <title>
     <tts>raiders of the lost ark</tts>
     <wav>raiders.wav</wav>
   </title>
    <details>
      <!-- details omitted for the sake of brevity -->
    </details>
  </movie>
</movies>
```

The following is an implementation of the user-defined GetMovieList function:

```
// return an array of user-defined movie objects retrieved from the
DOM
function GetMovieList(domMovies) {
  var movies = new Array();
  try {
    var nodeRoot = domMovies.documentElement;
    for (var i = 0; i < nodeRoot.childNodes.length; i++) {
      var nodeChild = nodeRoot.childNodes.item(i);
      if ("movie" != nodeChild.nodeName) {
        continue;
      }
      else {
        var objMovie = new Object();
        objMovie.id =
nodeChild.getAttributeNS("http://www.example.org" "id");
        var nodeTitle = GetTitle(nodeChild);
        objMovie.audio = GetWav(nodeTitle);
        objMovie.tts = GetTTS(nodeTitle);
        movies.push(objMovie);
      }
    }
  }
  catch(e) {
    // unable to build movie list
    var objError = new Object();
    objError.audio = "nomovies.wav";
```

```
    objError.tts = "sorry. no movies are available."
    details.push(objError);
  }
  return movies;
}

function GetTitle(nodeMovie) {
  return GetChild(nodeMovie, "title");
}

function GetTTS(node) {
  return GetInnerText(GetChild(node, "tts"));
}

function GetWav(node) {
  return GetInnerText(GetChild(node, "wav"));
}

// perform a shallow traversal of the node referenced by parent
// looking for the named node
function GetChild(parent, name)
{
  var target = null;
  for (var i = 0; i < parent.childNodes.length; i++) {
    var child = parent.childNodes.item(i);
    if (child.nodeName == name) {
      target = child;
      break;
    }
  }
  return target;
}

// Given a node, dig the text nodes out and return them as a string
function GetInnerText(node) {
  var s = "";
  if (null == node) {
    return s;
  }

  for (var i = 0; i < node.childNodes.length; i++) {
    var child = node.childNodes.item(i);
    if (Node.TEXT_NODE == child.nodeType || Node.CDATA_SECTION_NODE
== child.nodeType) {
      s += child.data;
    }
    else if (Node.ELEMENT_NODE == child.nodeType) {
      s += GetInnerText(child);
    }
  }
  return s;
}
```

The following is an implementation of the user-defined GetMovieDetail function:

```
// get the details about the movie specified by id
```

```
function GetMovieDetail(domMovies, id) {
  var details = new Array();
  try {
    var nodeMovie = domMovies.getElementById(id);
    var nodeTitle = GetTitle(nodeMovie);
    var objTitle = new Object();
    objTitle.audio = GetWav(nodeTitle);
    objTitle.tts = GetTTS(nodeTitle);
    details.push(objTitle);
    var nodeDetails = GetChild(nodeMovie, "details");
    for (var i = 0; i < nodeDetails.childNodes.length; i++) {
      var nodeChild = nodeDetails.childNodes.item(i);
      if ("detail" == nodeChild.nodeName) {
        var objDetail = new Object();
        objDetail.audio = GetWav(nodeChild);
        objDetail.tts = GetTTS(nodeChild);
        details.push(objDetail);
      }
    }
  }
  catch(e) {
    // couldn't get movie details
    var objError = new Object();
    objError.audio = "nomovie.wav";
    objError.tts = "sorry. details for that movie are unavailable."
    details.push(objError);
  }

  return details;
}
```

When the interpreter queues the second <prompt> in the field named "mov", it expands the <foreach> element to the following:

```
<mark name="0"/>
<audio src="godfather.wav">the godfather</audio>
<break time="300ms"/>
<mark name="1"/>
<audio src="high_fidelity.wav">high fidelity</audio>
<break time="300ms"/>
<mark name="2"/>
<audio src="raiders.wav">raiders of the lost ark</audio>
<break time="300ms"/>
```

If the user chooses "The Godfather", when the interpreter executes the "play_detail" <block>, it expands the <foreach> element to the following:

```
<audio src="directors/directed_by.wav">directed by</audio>
<audio src="directors/coppola.wav">francis ford coppola</audio>
<audio src="ratings/rated.wav">rated</audio>
<audio src="ratings/r.wav">r</audio>
<audio src="pauses/300.wav"/>
<audio src="synopsis/m0010.wav">a coming of age story about a nice
italian boy</audio>
```

# 7 Recording User Utterances While Attempting Recognition

Several elements defined in [VXML2] can instruct the interpreter to accept user input during execution. These elements include <field>, <initial>, <link>, <menu>, <record>, and <transfer>. VoiceXML 2.1 extends these elements to allow the interpreter to conditionally enable recording while simultaneously gathering input from the user.

To enable recording during recognition, set the value of the *recordutterance* property to true. If the *recordutterance* property is set to true in the current scope, the following three shadow variables are set on the application.lastresult$ object whenever the application.lastresult$ object is assigned (e.g. when a <link> is matched):

| | |
|---|---|
| **recording** | The variable that stores a reference to the recording, or undefined if no audio is collected. Like the input item variable associated with a <record> element as described in section 2.3.6 of [VXML2], the implementation of this variable may vary between platforms. |
| **recordingsize** | The size of the recording in bytes, or undefined if no audio is collected. |
| **recordingduration** | The duration of the recording in milliseconds, or undefined if no audio is collected. |

Table 8: recordutterance-related shadow variables

When these properties are set on the application.lastresult$ object, if an input item (as defined in section 2.3 of [VXML2]) has also been filled and has its shadow variables assigned, the interpreter must also assign *recording*, *recordingsize*, and *recordingduration* shadow variables for these input items, the values of which equal the corresponding properties of the application.lastresult$ object. For example, in the case of <link> and <menu>, since no input item has its shadow variables set, the interpreter only sets the application.lastresult$ properties.

Support for this feature is optional on <record>, and <transfer>. Platforms that support it set the aforementioned shadow variables on the associated form item variable and the corresponding properties on the application.lastresult$ object when the *recordutterance* property is set to true in an encompassing scope.

Like recordings created using the <record> tag, utterance recordings can be played back using the expr attribute on <audio>.

Like recordings created using the <record> tag, utterance recordings can be submitted to a document server via HTTP POST using the namelist attribute of the <submit>, <data>, and <subdialog> elements. The enctype attribute must be set to "multipart/form-data" [RFC2388], and the method attribute must be set to "post". To provide flexibility in the naming of the variable that is submitted to the document server, the interpreter must allow the utterance recording to be assigned to and posted via any valid ECMAScript variable.

In the following example, the dialog requests a city and state from the user. On the third recognition failure, the recording of the user's utterance is submitted to a document server.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml"
  version="2.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
  http://www.w3.org/TR/voicexml21/vxml.xsd">
  <form>
    <property name="recordutterance" value="true"/>

    <field name="city state">
     <prompt>
      Say a city and state.
     </prompt>

     <grammar type="application/srgs+xml" src="citystate.grxml"/>

     <nomatch>
      I'm sorry. I didn't get that.
      <reprompt/>
     </nomatch>

     <nomatch count="3">
        <var name="the recording"
          expr="application.lastresult$.recording"/>
        <submit method="post"
          enctype="multipart/form-data"
          next="upload.cgi"
          namelist="the recording"/>
     </nomatch>
    </field>
  </form>
</vxml>
```

## 7.1 Specifying the Media Format of Utterance Recordings

To specify the media format of the resulting recording, set the *recordutterancetype* property. Platforms must support the audio file formats specified in Appendix E of [VXML2]. Other formats may also be supported. The *recordutterancetype* property defaults to a platform-specific format which should be one of the required formats. If an unsupported media format is encountered

during recognition, the platform throws an *error.unsupported.format* event which specifies the unsupported media format in its message variable. Note that the *recordutterancetype* property does not affect the <record> element.

# 8 Adding namelist to <disconnect>

As described in section 5.3.11 of [VXML2], the <disconnect> element causes the interpreter context to disconnect from the user. VoiceXML 2.1 extends the <disconnect> element to support the following attribute:

| | |
|---|---|
| **namelist** | Variable names to be returned to the interpreter context. The default is to return no variables; this means the interpreter context will receive an empty ECMAScript object. If an undeclared variable is referenced in the namelist, then an *error.semantic* is thrown (5.1.1 of [VXML2]). |

Table 9: <disconnect> Attributes

The <disconnect> namelist and the <exit> namelist are processed independently. If the interpreter executes both a <disconnect> namelist and an <exit> namelist, both sets of variables are available to the interpreter context. The precise mechanism by which these variables are made available to the interpreter context is platform specific.

# 9 Adding type to <transfer>

As described in section 2.3.7 of [VXML2], the <transfer> element directs the interpreter to connect the caller to another entity. VoiceXML 2.1 extends the <transfer> element to support the following additional attribute:

| | |
|---|---|
| **type** | The type of transfer. The value can be "bridge", "blind", or "consultation". |

Table 10: <transfer> Attributes

Exactly one of "bridge" or "type" may be specified; otherwise an *error.badfetch* event is thrown.

As specified in 2.3.7 of [VXML2], the <transfer> element is optional, though platforms should support it. Platforms that support <transfer> may support any combination of bridge, blind, or consultation transfer types. Platforms supporting consultation transfer may optionally support bargein input modes of DTMF, speech recognition, or both, to cancel the call transfer attempt before the outgoing call is connected; bargein properties apply as normal. In addition, platforms supporting consultation transfer may optionally support the transferaudio attribute to specify the URI of an audio source to play while the transfer attempt is in progress and before far-end answer If the transferaudio attribute is not supported the behavior is the same as if it were not specified.

If the value of the type attribute is set to "bridge", the interpreter's behavior must be identical to its behavior when the value of the bridge attribute is set to "true". If the value of the type attribute is set to "blind", the interpreter's behavior must be identical to its behavior when the bridge attribute is set to "false". The behavior of the bridge attribute is fully specified in section 2.3.7 of [VXML2]. If the type attribute is specified and the bridge attribute is absent, the value of the type attribute takes precedence over the default value of the bridge attribute.

The bridge attribute is maintained for backwards compatiblity with [VXML2]. Since all of the functionality of the bridge attribute has been incorporated into the type attribute, developers are encouraged to use the type attribute on platforms that implement it.

The connecttimeout attribute of <transfer> applies if the type attribute is set to "bridge" or "consultation".

The maxtime attribute of <transfer> applies if the type attribute is set to "bridge".

## 9.1 Consultation Transfer

The consultation transfer is similar to a blind transfer except that the outcome of the transfer call setup is known and the caller is not dropped as a result of an unsuccessful transfer attempt. When performing a consultation transfer, the platform monitors the progress of the transfer until the connection is established between caller and callee. If the connection cannot be established (e.g. no answer, line busy, etc.), the session remains active and returns control to the application. As in the case of a blind transfer, if the connection is established, the interpreter disconnects from the session, connection.disconnect.transfer is thrown, and document interpretation continues normally. Any connection between the caller and the callee remains in place regardless of document execution. For additional information on call transfers with consultation-like functionality see AT&T Toll Free Transfer Connect [ATT_50075], MCI Enhanced Call Routing [MCI_ECR], ETSI Explicit Call Transfer [ETSI300_369] , and SIP Attended Transfer [SIP_EX].

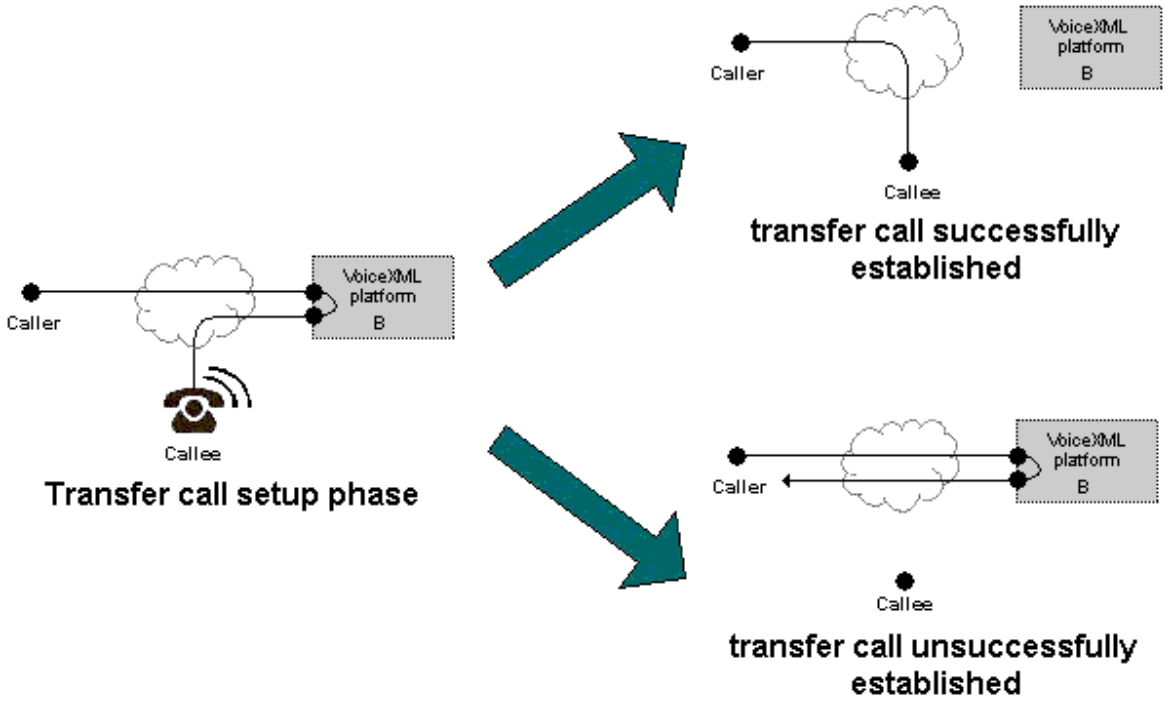


Figure 1: Audio Connections during a consultation transfer: <transfer type="consultation"/>

The possible outcomes for a consultation transfer before the connection to the callee is established are:

| Action | Value of form item variable | Event | Reason |
|---|---|---|---|
| caller disconnects | | connection.disconnect.hangup | The caller hung up. |
| caller cancels transfer before outgoing call begins | near_end_disconnect | | The caller cancelled the transfer attempt via a DTMF or voice command before the outgoing call begins (during playback of queued audio). |
| callee busy | busy | | The callee was busy. |
| network | network_busy | | An |

| | | | |
|---|---|---|---|
| busy | | | intermediate network refused the call. |
| callee does not answer | noanswer | | There was no answer within the time specified by the connecttimeout attribute. |
| --- | unknown | | The transfer ended but the reason is not known. |

Table 11: Consultation Transfer Outcomes Prior to Connection Being Established

The possible outcomes for a consultation transfer once the connection to the callee is established are:

| Action | Value of form item variable | Event | Reason |
|---|---|---|---|
| **transfer begins** | undefined | connection.disconnect.transfer | The caller was transferred to another line and will not return. |

Table 12: Consultation Transfer Outcome After Connection Has Been Established

## 9.2 Consultation Transfer Errors and Events

One of the following events may be thrown during a consultation transfer:

| Event | Reason |
|---|---|
| **connection.disconnect.hangup** | The caller hung up. |
| **connection.disconnect.transfer** | The caller was transferred to another line and will not return. |

Table 13: Events thrown during consultation transfer

If a consultation transfer could not be made, one of the following errors will be thrown:

| Error | Reason |
|---|---|

| | |
|---|---|
| **error.connection.noauthorization** | The caller is not allowed to call the destination. |
| **error.connection.baddestination** | The destination URI is malformed. |
| **error.connection.noroute** | The platform is not able to place a call to the destination. |
| **error.connection.noresource** | The platform cannot allocate resources to place the call. |
| **error.connection.*protocol.nnn*** | The protocol stack for this connection raised an exception that does not correspond to one of the other error.connection events. |
| **error.unsupported.transfer.consultation** | The platform does not support consultation transfer. |
| **error.unsupported.uri** | The platform does not support the URI format used. The special variable _message (section 5.2.2 of [VXML2]) will contain the string "The URI x is not a supported URI format" where x is the URI from the dest or destexpr <transfer> attributes. |

Table 14: Consultation transfer attempt error events

## 9.3 Example of a Consultation Transfer

The following example attempts to perform a consultation transfer of the caller to a another party. Prompts may be included before or within the <transfer> element. This may be used to inform the caller of what is happening, with a notice such as "Please wait while we transfer your call." The <prompt> within the <block>, and the <prompt> within <transfer> are queued and played before actually performing the transfer. After the prompt queue is flushed, the outgoing call is initiated. The "transferaudio" attribute specifies an audio file to be played to the caller in place of audio from the far-end until the far-end answers. If the audio source is longer than the connect time, the audio will stop playing immediately upon far-end answer.

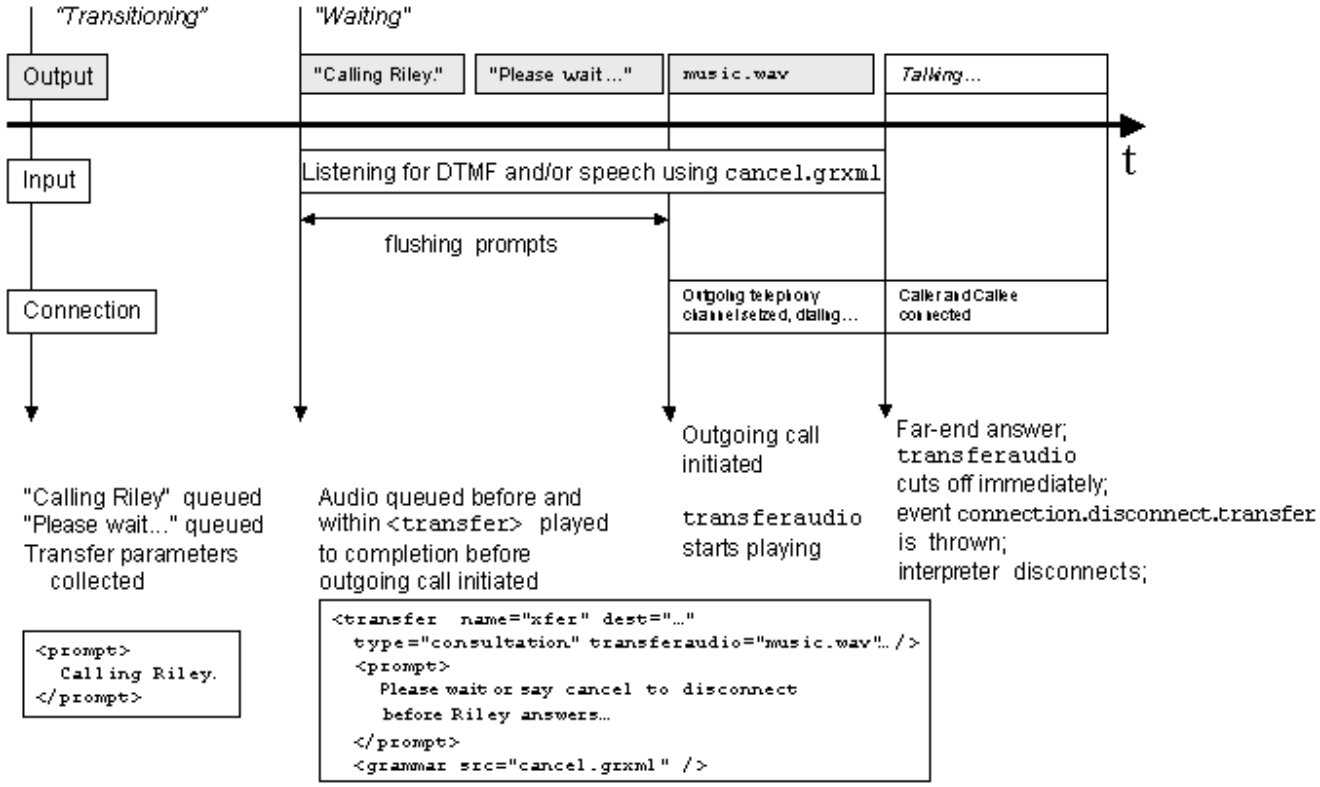


Figure 2: Sequence and timing during an example of a consultation transfer

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1"
  xmlns="http://www.w3.org/2001/vxml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
  http://www.w3.org/TR/voicexml21/vxml.xsd">
<catch event="connection.disconnect.transfer">
   <!-- far-end answered -->
   <log> Connection with the callee established: transfer
executed.</log>
</catch>

<form id="consultation_xfer">
   <block>
     <!-- queued and played before starting the transfer -->
     <prompt>
        Calling Riley.
     </prompt>
   </block>
   <!-- Play music while attempting to connect to far-end -->
   <!-- Wait up to 60 seconds for the far end to answer  -->
   <transfer name="mycall" dest="tel:+1-555-123-4567"
      transferaudio="music.wav" connecttimeout="60s"
type="consultation">
     <!-- queued and played before starting the transfer -->
     <prompt>
        Please wait...
     </prompt>
```

```
     <filled>
        <if cond="mycall == 'busy'">
           <prompt>
             Riley's line is busy. Please call again later.
           </prompt>
         <elseif cond="mycall == 'noanswer'"/>
           <prompt>
             Riley can't answer the phone now. Please call
             again later.
           </prompt>
         </if>
      </filled>
   </transfer>
   <!-- submit call statistics to server -->
   <block>
      <submit namelist="mycall" next="/cgi-bin/report"/>
   </block>
</form>
</vxml>
```

# A VoiceXML Document Type Definition

The VoiceXML DTD is located at http://www.w3.org/TR/voicexml21/vxml.dtd.

Due to DTD limitations, the VoiceXML DTD does not correctly express that the <metadata> element can contain elements from other XML namespaces.

**Note:**

The VoiceXML DTD includes modified elements from the DTDs of the Speech Recognition Grammar Specification 1.0 [SRGS] and the Speech Synthesis Markup Language 1.0 [SSML].

# B VoiceXML Schema

This section is Normative.

The VoiceXML schema is located at http://www.w3.org/TR/voicexml21/vxml.xsd.

The VoiceXML schema depends upon other schema defined in the VoiceXML namespace:

- vxml-datatypes.xsd: definition of datatypes used in VoiceXML schema
- vxml-attribs.xsd: definition of attributes and attribute groups used in VoiceXML schema
- vxml-grammar-restriction.xsd: this schema references the no-namespace schema of the Speech Recognition Grammar Specification 1.0 [SRGS] and restricts some of its definitions for embedding in the VoiceXML namespace.

- vxml-grammar-extension.xsd: this schema references vxml-grammar-restriction.xsd and extends some of its definitions for VoiceXML.
- vxml-synthesis-restriction.xsd: this schema references the no-namespace schema of the Speech Synthesis Markup Language 1.0 [SSML] and extends as well as restricts some of its definitions for embedding in the VoiceXML namespace.
- vxml-synthesis-extension.xsd: this schema references vxml-synthesis-restriction.xsd and extends some of its definitions for VoiceXML.

The complete set of Speech Interface Framework schema required for VoiceXML 2.1 is available here.

**Note:**

In order to accommodate the addition of the nameexpr attribute, the name attribute is optional on the <mark> element within VoiceXML 2.1. It is mandatory in [SSML] and [VXML2].

# C Conformance

This section is normative.

## C.1 Conforming VoiceXML 2.1 Document

A VoiceXML 2.1 document is a conforming VoiceXML 2.1 document if it adheres to the specification described in this document (Voice Extensible Markup Language (VoiceXML) 2.1 Specification) including VoiceXML 2.1's schema (see VoiceXML Schema). A conforming VoiceXML 2.1 document must meet all of the following criteria:

1. The root element of the document must be the <vxml> element.
2. The <vxml> element must include a "version" attribute with the value "2.1".
3. The <vxml> element must designate the VoiceXML namespace using the "xmlns" attribute [XMLNAMES]. The namespace for VoiceXML is defined to be http://www.w3.org/2001/vxml.
4. The document must conform to the constraints expressed in the VoiceXML 2.1 Schema.
5. The <vxml> element may also include "xmlns:xsi" and "xsi:schemaLocation" attributes to indicate the location of the schema for the VoiceXML namespace. If the "xsi:schemaLocation" attribute is present, it must include a reference to the VoiceXML 2.1 Schema:

```
6. xsi:schemaLocation="http://www.w3.org/2001/vxml
     http://www.w3.org/TR/voicexml21/vxml.xsd"
```

A document is a conforming VoiceXML 2.1 document if it satisfies the requirements of this specification (Voice Extensible Markup Language (VoiceXML) 2.1 Specification) and is valid per the normative schema identified by

```
http://www.w3.org/TR/voicexml21/vxml.xsd"
```

7. There may be a DOCTYPE declaration in the document prior to the root element. If present, the public identifier included in the DOCTYPE declaration must reference the VoiceXML 2.1 DTD using its Formal Public Identifier.

```
<!DOCTYPE vxml
  PUBLIC "-//W3C//DTD VOICEXML 2.1//EN"
   "http://www.w3.org/TR/voicexml21/vxml.dtd">
```

The system identifier may be modified appropriately. If a document contains this declaration, it must be a valid XML document. Since the DTD for a document consists of both the external and internal subset, a document conforming to this specification must not include an internal subset when referencing the VoiceXML 2.1 DTD.

Here is an example of a Conforming VoiceXML 2.1 document:

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.1" xmlns="http://www.w3.org/2001/vxml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
  http://www.w3.org/TR/voicexml21/vxml.xsd">

  <form>
    <block>hello</block>
  </form>

</vxml>
```

Note that in this example, the recommended "xmlns:xsi" and "xsi:schemaLocation" attributes are included as is an XML declaration. An XML declaration like the one above is not required in all XML documents. VoiceXML document authors are strongly encouraged to use XML declarations in all their documents. Such a declaration is required when the character encoding of the document is other than the default UTF-8 or UTF-16 and no encoding was determined by a higher-level protocol.

The VoiceXML language or these conformance criteria provide no designated size limits on any aspect of VoiceXML 2.1 documents. There are no maximum values on the number of elements, the amount of character data, or the number of characters in attribute values.

## C.2 Using VoiceXML with other namespaces

The VoiceXML namespace may be used with other XML namespaces as per [XMLNAMES], although such documents are not strictly conforming VoiceXML 2.1 documents as defined above. Future work by W3C will address ways to specify conformance for documents involving multiple namespaces.

## C.3 Conforming VoiceXML 2.1 Processors

A VoiceXML 2.1 processor is a user agent that can parse and process Conforming VoiceXML 2.1 documents.

In a Conforming VoiceXML 2.1 Processor, the XML parser must be able to parse and process all well-formed XML constructs defined within [XML] and [XMLNAMES]. It is not required that a Conforming VoiceXML 2.1 processor use a validating parser.

A Conforming VoiceXML 2.1 Processor must be a Conforming Speech Synthesis Markup Language Processor [SSML] and a Conforming XML Grammar Processor [SRGS] except for differences described in this document. If a syntax error is detected while processing a grammar document, then an *error.badfetch* event must be thrown.

A Conforming VoiceXML 2.1 Processor must support the syntax and semantics of all VoiceXML elements as described in this document and in [VXML2]. Consequently, a Conforming VoiceXML 2.1 Processor must not throw an *error.unsupported.<element>* for any VoiceXML element which must be supported when processing a Conforming VoiceXML 2.1 Document.

When a Conforming VoiceXML 2.1 Processor encounters a Conforming VoiceXML 2.1 Document with non-VoiceXML elements or attributes which are proprietary, defined only in versions of VoiceXML earlier than [VXML2], or defined in a non-VoiceXML namespace, and which cannot be processed, then it must throw an *error.badfetch* event.

When a Conforming VoiceXML 2.1 Processor encounters a document with a root element designating a namespace other than VoiceXML, its behavior is undefined.

When a conforming VoiceXML 2.1 Processor encounters a document with a root element of <vxml> with a version attribute with a value other than 2.0 or 2.1, its behavior is undefined.

There is, however, no conformance requirement with respect to performance characteristics of the VoiceXML 2.1 Processor.

While the features of VoiceXML 2.1 are orthogonal additions to [VXML2], a VoiceXML application should not reference documents of both types. The handling of a single application that references both VoiceXML 2.0 and VoiceXML 2.1 documents is platform-specific. Reasonable behavior in this case ranges the gamut from successful execution to the throwing of an error.

Interpreters that support both VoiceXML 2.0 documents and VoiceXML 2.1 documents must support the ability to transition from an application of one version to an application of another version.

**Note:**

The xsd:anyURI type and thus URI references in VoiceXML documents may contain a wide array of international characters. Implementers should reference [RFC 3987] and the [CHARMODEL] in order to provide appropriate support for these characters in VoiceXML documents and when processing values of this type or mapping them to URIs.

# D ECMAScript Language Binding for DOM

This appendix contains the ECMAScript binding for the subset of Level 2 of the Document Object Model exposed by the <data> element.

**Prototype Object DOMException**

**The DOMException class has the following constants:**

**DOMException.INDEX_SIZE_ERR**
This constant is of type **Number** and its value is **1**.

**DOMException.DOMSTRING_SIZE_ERR**
This constant is of type **Number** and its value is **2**.

**DOMException.HIERARCHY_REQUEST_ERR**
This constant is of type **Number** and its value is **3**.

**DOMException.WRONG_DOCUMENT_ERR**
This constant is of type **Number** and its value is **4**.

**DOMException.INVALID_CHARACTER_ERR**
This constant is of type **Number** and its value is **5**.

**DOMException.NO_DATA_ALLOWED_ERR**
This constant is of type **Number** and its value is **6**.

**DOMException.NO_MODIFICATION_ALLOWED_ERR**
This constant is of type **Number** and its value is **7**.

**DOMException.NOT_FOUND_ERR**
This constant is of type **Number** and its value is **8**.
**DOMException.NOT_SUPPORTED_ERR**
This constant is of type **Number** and its value is **9**.
**DOMException.INUSE_ATTRIBUTE_ERR**
This constant is of type **Number** and its value is **10**.
**DOMException.INVALID_STATE_ERR**
This constant is of type **Number** and its value is **11**.
**DOMException.SYNTAX_ERR**
This constant is of type **Number** and its value is **12**.
**DOMException.INVALID_MODIFICATION_ERR**
This constant is of type **Number** and its value is **13**.
**DOMException.NAMESPACE_ERR**
This constant is of type **Number** and its value is **14**.
**DOMException.INVALID_ACCESS_ERR**
This constant is of type **Number** and its value is **15**.
**The DOMException object has the following properties:**
**code**
This property is of type **Number**.

**Object DOMImplementation**

**The DOMImplementation object has the following methods:**
**hasFeature(feature, version)**
This method returns a **Boolean**.
The **feature** parameter is of type **String** .
The **version** parameter is of type **String** .
**createDocumentType(qualifiedName, publicId, systemId)**
This method returns a **DocumentType** object.
The **qualifiedName** parameter is of type **String** .
The **publicId** parameter is of type **String** .
The **systemId** parameter is of type **String** .
This method can raise a **DOMException** object.
**createDocument(namespaceURI, qualifiedName, doctype)**
This method returns a **Document** object.
The **namespaceURI** parameter is of type **String** .
The **qualifiedName** parameter is of type **String** .
The **doctype** parameter is a **DocumentType** object .
This method can raise a **DOMException** object.

**Object DocumentFragment**

**DocumentFragment has all the properties and methods of the Node object .**

**Object Document**

**Document has all the properties and methods of the Node object as well as the properties and methods defined below.**

**The Document object has the following properties:**
**doctype**
This read-only property is a **DocumentType** object.
**implementation**
This read-only property is a **DOMImplementation** object.
**documentElement**
This read-only property is a **Element** object.
**The Document object has the following methods:**
**createElement(tagName)**
This method returns a **Element** object.
The **tagName** parameter is of type **String** .
This method can raise a **DOMException** object.
**createDocumentFragment()**
This method returns a **DocumentFragment** object.
This method can raise a **DOMException** object.
**createTextNode(data)**
This method returns a **Text** object.
The **data** parameter is of type **String** .
**createComment(data)**
This method returns a **Comment** object.
The **data** parameter is of type **String** .
**createCDATASection(data)**
This method returns a **CDATASection** object.
The **data** parameter is of type **String** .
This method can raise a **DOMException** object.
**createProcessingInstruction(target, data)**
This method returns a **ProcessingInstruction** object.
The **target** parameter is of type **String** .
The **data** parameter is of type **String** .
This method can raise a **DOMException** object.
**createAttribute(name)**
This method returns a **Attr** object.
The **name** parameter is of type **String** .
This method can raise a **DOMException** object.
**createEntityReference(name)**
This method returns a **EntityReference** object.
The **name** parameter is of type **String** .
This method can raise a **DOMException** object.
**getElementsByTagName(tagname)**
This method returns a **NodeList** object.
The **tagname** parameter is of type **String** .
**importNode(importedNode, deep)**
This method returns a **Node** object.
The **importedNode** parameter is a **Node** object .
The **deep** parameter is of type **Boolean** .
This method can raise a **DOMException** object.

**createElementNS(namespaceURI, qualifiedName)**
This method returns a **Element** object.
The **namespaceURI** parameter is of type **String** .
The **qualifiedName** parameter is of type **String** .
This method can raise a **DOMException** object.
**createAttributeNS(namespaceURI, qualifiedName)**
This method returns a **Attr** object.
The **namespaceURI** parameter is of type **String** .
The **qualifiedName** parameter is of type **String** .
This method can raise a **DOMException** object.
**getElementsByTagNameNS(namespaceURI, localName)**
This method returns a **NodeList** object.
The **namespaceURI** parameter is of type **String** .
The **localName** parameter is of type **String** .
**getElementById(elementId)**
This method returns a **Element** object.
The **elementId** parameter is of type **String** .

**Prototype Object Node**

**The Node class has the following constants:**
**Node.ELEMENT_NODE**
This constant is of type **Number** and its value is **1**.
**Node.ATTRIBUTE_NODE**
This constant is of type **Number** and its value is **2**.
**Node.TEXT_NODE**
This constant is of type **Number** and its value is **3**.
**Node.CDATA_SECTION_NODE**
This constant is of type **Number** and its value is **4**.
**Node.ENTITY_REFERENCE_NODE**
This constant is of type **Number** and its value is **5**.
**Node.ENTITY_NODE**
This constant is of type **Number** and its value is **6**.
**Node.PROCESSING_INSTRUCTION_NODE**
This constant is of type **Number** and its value is **7**.
**Node.COMMENT_NODE**
This constant is of type **Number** and its value is **8**.
**Node.DOCUMENT_NODE**
This constant is of type **Number** and its value is **9**.
**Node.DOCUMENT_TYPE_NODE**
This constant is of type **Number** and its value is **10**.
**Node.DOCUMENT_FRAGMENT_NODE**
This constant is of type **Number** and its value is **11**.
**Node.NOTATION_NODE**
This constant is of type **Number** and its value is **12**.
**The Node object has the following properties:**
**nodeName**

This read-only property is of type **String**.
**nodeValue**
This property is of type **String**. This property can raise a **DOMException** object on retrieval.
**nodeType**
This read-only property is of type **Number**.
**parentNode**
This read-only property is a **Node** object.
**childNodes**
This read-only property is a **NodeList** object.
**firstChild**
This read-only property is a **Node** object.
**lastChild**
This read-only property is a **Node** object.
**previousSibling**
This read-only property is a **Node** object.
**nextSibling**
This read-only property is a **Node** object.
**attributes**
This read-only property is a **NamedNodeMap** object.
**ownerDocument**
This read-only property is a **Document** object.
**namespaceURI**
This read-only property is of type **String**.
**prefix**
This property is of type **String**.
**localName**
This read-only property is of type **String**.
**The Node object has the following methods:**
**insertBefore(newChild, refChild)**
This method returns a **Node** object.
The **newChild** parameter is a **Node** object .
The **refChild** parameter is a **Node** object .
This method can raise a **DOMException** object.
**replaceChild(newChild, oldChild)**
This method returns a **Node** object.
The **newChild** parameter is a **Node** object .
The **oldChild** parameter is a **Node** object .
This method can raise a **DOMException** object.
**removeChild(oldChild)**
This method returns a **Node** object.
The **oldChild** parameter is a **Node** object .
This method can raise a **DOMException** object.
**appendChild(newChild)**

This method returns a **Node** object.
The **newChild** parameter is a **Node** object .
This method can raise a **DOMException** object.
**hasChildNodes()**
This method returns a **Boolean**.
**cloneNode(deep)**
This method returns a **Node** object.
The **deep** parameter is of type **Boolean** .
**normalize()**
This method has no return value.
**isSupported(feature, version)**
This method returns a **Boolean**.
The **feature** parameter is of type **String** .
The **version** parameter is of type **String** .
**hasAttributes()**
This method returns a **Boolean**.

**Object NodeList**

**The NodeList object has the following properties:**
**length**
This read-only property is of type **Number**.
**The NodeList object has the following methods:**
**item(index)**
This method returns a **Node** object.
The **index** parameter is of type **Number** .

**Object NamedNodeMap**

**The NamedNodeMap object has the following properties:**
**length**
This read-only property is of type **Number**.
**The NamedNodeMap object has the following methods:**
**getNamedItem(name)**
This method returns a **Node** object.
The **name** parameter is of type **String** .
**setNamedItem(arg)**
This method returns a **Node** object.
The **arg** parameter is a **Node** object .
This method can raise a **DOMException** object.
**removeNamedItem(name)**
This method returns a **Node** object.
The **name** parameter is of type **String** .
This method can raise a **DOMException** object.
**item(index)**
This method returns a **Node** object.
The **index** parameter is of type **Number** .
**getNamedItemNS(namespaceURI, localName)**

This method returns a **Node** object.
The **namespaceURI** parameter is of type **String** .
The **localName** parameter is of type **String** .
**setNamedItemNS(arg)**
This method returns a **Node** object.
The **arg** parameter is a **Node** object .
This method can raise a **DOMException** object.
**removeNamedItemNS(namespaceURI, localName)**
This method returns a **Node** object.
The **namespaceURI** parameter is of type **String** .
The **localName** parameter is of type **String** .
This method can raise a **DOMException** object.

**Object CharacterData**

**CharacterData has all the properties and methods of the Node object as well as the properties and methods defined below.**
**The CharacterData object has the following properties:**
**data**
This property is of type **String**. This property can raise a **DOMException** object on retrieval.
**length**
This read-only property is of type **Number**.
**The CharacterData object has the following methods:**
**substringData(offset, count)**
This method returns a **String**.
The **offset** parameter is of type **Number** .
The **count** parameter is of type **Number** .
This method can raise a **DOMException** object.
**appendData(arg)**
This method has no return value.
The **arg** parameter is of type **String** .
This method can raise a **DOMException** object.
**insertData(offset, arg)**
This method has no return value.
The **offset** parameter is of type **Number** .
The **arg** parameter is of type **String** .
This method can raise a **DOMException** object.
**deleteData(offset, count)**
This method has no return value.
The **offset** parameter is of type **Number** .
The **count** parameter is of type **Number** .
This method can raise a **DOMException** object.
**replaceData(offset, count, arg)**
This method has no return value.
The **offset** parameter is of type **Number** .
The **count** parameter is of type **Number** .

The **arg** parameter is of type **String** .
This method can raise a **DOMException** object.

**Object Attr**

**Attr has all the properties and methods of the Node object as well as the properties and methods defined below.**
**The Attr object has the following properties:**
**name**
This read-only property is of type **String**.
**specified**
This read-only property is of type **Boolean**.
**value**
This property is of type **String**.
**ownerElement**
This read-only property is a **Element** object.

**Object Element**

**Element has all the properties and methods of the Node object as well as the properties and methods defined below.**
**The Element object has the following properties:**
**tagName**
This read-only property is of type **String**.
**The Element object has the following methods:**
**getAttribute(name)**
This method returns a **String**.
The **name** parameter is of type **String** .
**setAttribute(name, value)**
This method has no return value.
The **name** parameter is of type **String** .
The **value** parameter is of type **String** .
This method can raise a **DOMException** object.
**removeAttribute(name)**
This method has no return value.
The **name** parameter is of type **String** .
This method can raise a **DOMException** object.
**getAttributeNode(name)**
This method returns a **Attr** object.
The **name** parameter is of type **String** .
**setAttributeNode(newAttr)**
This method returns a **Attr** object.
The **newAttr** parameter is a **Attr** object .
This method can raise a **DOMException** object.
**removeAttributeNode(oldAttr)**
This method returns a **Attr** object.
The **oldAttr** parameter is a **Attr** object .
This method can raise a **DOMException** object.

**getElementsByTagName(name)**
This method returns a **NodeList** object.
The **name** parameter is of type **String** .
**getAttributeNS(namespaceURI, localName)**
This method returns a **String**.
The **namespaceURI** parameter is of type **String** .
The **localName** parameter is of type **String** .
**setAttributeNS(namespaceURI, qualifiedName, value)**
This method has no return value.
The **namespaceURI** parameter is of type **String** .
The **qualifiedName** parameter is of type **String** .
The **value** parameter is of type **String** .
This method can raise a **DOMException** object.
**removeAttributeNS(namespaceURI, localName)**
This method has no return value.
The **namespaceURI** parameter is of type **String** .
The **localName** parameter is of type **String** .
This method can raise a **DOMException** object.
**getAttributeNodeNS(namespaceURI, localName)**
This method returns a **Attr** object.
The **namespaceURI** parameter is of type **String** .
The **localName** parameter is of type **String** .
**setAttributeNodeNS(newAttr)**
This method returns a **Attr** object.
The **newAttr** parameter is a **Attr** object .
This method can raise a **DOMException** object.
**getElementsByTagNameNS(namespaceURI, localName)**
This method returns a **NodeList** object.
The **namespaceURI** parameter is of type **String** .
The **localName** parameter is of type **String** .
**hasAttribute(name)**
This method returns a **Boolean**.
The **name** parameter is of type **String** .
**hasAttributeNS(namespaceURI, localName)**
This method returns a **Boolean**.
The **namespaceURI** parameter is of type **String** .
The **localName** parameter is of type **String** .

**Object Text**

**Text has all the properties and methods of the CharacterData object .**

**Object Comment**

**Comment has all the properties and methods of the CharacterData object .**

**Object CDATASection**

**CDATASection has all the properties and methods of the Text object .**

**Object DocumentType**

**DocumentType has all the properties and methods of the Node object as well as the properties and methods defined below.**
**The DocumentType object has the following properties:**
**name**
This read-only property is of type **String**.
**entities**
This read-only property is a **NamedNodeMap** object.
**notations**
This read-only property is a **NamedNodeMap** object.
**publicId**
This read-only property is of type **String**.
**systemId**
This read-only property is of type **String**.
**internalSubset**
This read-only property is of type **String**.

**Object Notation**

**Notation has all the properties and methods of the Node object as well as the properties and methods defined below.**
**The Notation object has the following properties:**
**publicId**
This read-only property is of type **String**.
**systemId**
This read-only property is of type **String**.

**Object Entity**

**Entity has all the properties and methods of the Node object as well as the properties and methods defined below.**
**The Entity object has the following properties:**
**publicId**
This read-only property is of type **String**.
**systemId**
This read-only property is of type **String**.
**notationName**
This read-only property is of type **String**.

**Object EntityReference**

**EntityReference has all the properties and methods of the Node object .**

**Object ProcessingInstruction**

**ProcessingInstruction has all the properties and methods of the Node object as well as the properties and methods defined below. The ProcessingInstruction object has the following properties:**
**target**
This read-only property is of type **String**.
**data**
This property is of type **String**.

# E VoiceXML Media Type

This section is Normative.

The media type for VoiceXML is "application/voicexml+xml" as specified in [RFC4267].

# F References

## F.1 Normative References


**DOM2**
DOM Level 2, ed. Arnaud Le Hors et al. W3C Recommendation, November 2000. See http://www.w3.org/TR/DOM-Level-2-Core/.

**ECMA**
Standard ECMA-262 ECMAScript Language Specification, December 1999. See http://www.ecma-international.org/publications/standards/Ecma-262.htm.

**SRGS**
Speech Recognition Grammar Specification Version 1.0, ed. Andrew Hunt and Scott McGlashan W3C Recommendation, March 2004. See http://www.w3.org/TR/2004/REC-speech-grammar-20040316/.

**SSML**
Speech Synthesis Markup Language, ed. Daniel C. Burnett et al. W3C Recommendation, September 2004. See http://www.w3.org/TR/2004/REC-speech-synthesis-20040907/.

**RFC2119**
Key words for use in RFCs to Indicate Requirement Levels, ed. S. Bradner, March 1997. See http://www.ietf.org/rfc/rfc2119.txt.

**RFC2388**
Returning Values from Forms: multipart/form-data, ed. L. Masinter, August 1998. See http://www.ietf.org/rfc/rfc2388.txt.

**RFC2616**
Hypertext Transfer Protocol -- HTTP/1.1, ed. R. Fielding et al. IETF RFC 2616, June 1999. See http://www.ietf.org/rfc/rfc2616.txt.

**RFC3023**

XML Media Types, ed. M. Murata et al. IETF RFC 3023, January 2001. See http://www.ietf.org/rfc/rfc3023.txt.

**RFC4267**

The W3C Speech Interface Framework Media Types, ed. M. Froumentin. IETF RFC 4267, November 2005. See http://www.ietf.org/rfc/rfc4267.txt.

**VXML2**

VoiceXML 2.0, ed. Scott McGlashan et al. W3C Recommendation, March 2004. See http://www.w3.org/TR/2004/REC-voicexml20-20040316/.

**XML**

Extensible Markup Language (XML) 1.0, ed. Tim Bray et al. W3C Recommendation, February 2004. See http://www.w3.org/TR/2004/REC-xml-20040204/.

**XMLNAMES**

Namespaces in XML, ed. Tim Bray et al. W3C Recommendation, January 1999. See http://www.w3.org/TR/1999/REC-xml-names-19990114/.

## F.2 Informative References

**ATT_50075**

AT&T Toll Free Transfer Connect (TR 50075), September 1999. See http://www.att.com/cpetesting/pdf/tr50075.pdf.

**CHARMODEL**

Character Model for the World Wide Web 1.0: Resource Identifiers, ed. Martin Duerst et al. W3C Candidate Recommendation, November 2004. See http://www.w3.org/TR/2004/CR-charmod-resid-20041122/.

**DATA_AUTH**

Authorizing Read Access to XML Content Using the <?access-control?> Processing Instruction, ed. Matt Oshry et al. W3C Note, June 2005. See http://www.w3.org/TR/2005/NOTE-access-control-20050613/.

**ETSI300_369**

ETS 300 369-1: Integrated Services Digital Network (ISDN);Explicit Call Transfer (ECT) supplementary service;Digital Subscriber Signalling System No. one (DSS1) protocol;Part 1: Protocol specification, May 1995. See http://www.etsi.org/.

**HTML4**

HTML 4.01, ed. Dave Raggett et al. W3C Recommendation, December 1999. See http://www.w3.org/TR/1999/REC-html401-19991224/.

**MCI_ECR**

MCI Enhanced Call Routing, See http://global.mci.com/wholesale/us/voice/enhancedcallrouting/.

**RFC3987**

Internationalized Resource Identifiers (IRIs), ed. M. Duerst, January 2005. See http://www.ietf.org/rfc/rfc3987.txt.

**SIP_EX**

Session Initiation Protocol Service Examples, ed. A. Johnston et al. Internet-Draft, March 2006. See

http://www.softfront.co.jp/tech/ietfdoc/org/draft-ietf-sipping-service-examples-10.txt.

## F.3 Acknowledgements

VoiceXML 2.1 was written by the participants in the W3C Voice Browser Working Group. The following have significantly contributed to writing this specification:

- Matt Oshry, Tellme Networks
- RJ Auburn, Voxeo Corporation
- Paolo Baggia, Loquendo
- Michael Bodell, Tellme Networks
- David Burke, Voxpilot Ltd.
- Daniel C. Burnett, Invited Expert
- Emily Candell, Comverse
- Ken Davies, HeyAnita
- Jim Ferrans, Motorola
- Jeff Haynie, Vocalocity
- Matt Henry, Voxeo Corporation
- Hakan Kilic, Scansoft
- Scott McGlashan, Hewlett-Packard
- Brien Muschett, IBM
- Rob Marchand, VoiceGenie
- Jeff Kusnitz, IBM
- Brad Porter, Tellme Networks
- Ken Rehor, Invited Expert
- Laura Ricotti, Loquendo
- Davide Tosello, Loquendo
- Milan Young, Nuance

# G Change Summary

## G.1 Summary of changes since the Last Call Working Draft

The following is a summary of the major changes since the Last Call Working Draft was published.

- Added informative references to transfer specifications similar to consultation type transfer.
- Moved description of mechanism for securing access to XML data (access-control PI) to W3C NOTE.
- Made HTML4 ref informative.
- Added normative reference to RFC 2119 and described usage of 'must', 'may', 'should' in Status
- Added Elements table.

- Added datafetchhint, datamaxstale, and datamaxage properties and specified defaults for fetchtimeout, fetchaudio, fetchhint, maxstale, and maxage properties of <data>.
- Clarified lastresult$ and shadow variable assignment behavior with respect to utterance recordings.
- Clarified relationship between fetchaudio and mark-related lastresult$ properties and shadow variables.
- Clarified interaction between interpreter and interpreter context when both disconnect namelist and exit namelist occur.
- Clarified scope in which grammar srcexpr expression is evaluated.
- Replaced script srcexpr example.
- Replaced grammar srcexpr example2.
- The schema and DTD were updated to reflect that the name attribute on data is optional.
- Added note to schema and Appendix C.3 regarding support for international characters in URIs.

## G.2 Summary of changes since the Candidate Recommendation

The following is a summary of the major changes since the Candidate Recommendation was published.

- Updated RestrictedVariableName.datatype in vxml-datatypes.xsd to allow range of characters as described in ECMA-262.
- Added terminology section.
- Clarified that mark's nameexpr is evaluated when the containing prompt is queued.
- Clarified that the MIME type of the document reference by <data> is defined in RFC 3023.
- Processing of fragment identifiers on URIs referenced by the <data> element is platform-specific.
- Support for bargein and transferaudio during setup of a consultation transfer is optional.
- Miscellaneous conformance clarifications.
- Added reference to the W3C Speech Interface Framework Media Types which includes official VoiceXML media type.
- Modified <foreach> content model and clarified ECMAScript functionality.