XHTML+Voice Profile 1.1

28 January 2003

This version:
http://www.ibm.com/pvc/multimodal/x+v/11/spec.html
Latest version:
http://www.ibm.com/pvc/multimodal/x+v/11/spec.html
Previous versions:
http://www.w3.org/TR/2001/NOTE-xhtml+voice-20011221
Editors:
Chris Cross, IBM <xcross@us.ibm.com>
Jonny Axelsson, Opera Software <jax@opera.no>
Gerald McCobb, IBM <mccobb@us.ibm.com>
T. V. Raman, IBM <tvraman@us.ibm.com>
Les Wilson, IBM <lesw@us.ibm.com>

Abstract

The XHTML+Voice profile brings spoken interaction to standard web content by integrating the mature XHTML and XML-Events technologies with XML vocabularies developed as part of the W3C Speech Interface Framework. The profile includes voice modules that support speech synthesis, speech dialogs, command and control, and speech grammars. Voice handlers can be attached to XHTML elements and respond to specific DOM events, thereby reusing the event model familiar to web developers. Voice interaction features are integrated with XHTML and CSS and can consequently be used directly within XHTML content.

Status of this Document

This section describes the status of this document at the time of its publication. Other documents may supersede this document.

Note that the language profile described in this specification re-uses W3C working drafts that are likely to change. This integration profile will be updated as needed to use the final stable versions of these specifications. This profile is an update to the XHTML+Voice 1.0 profile. XHTML+Voice 1.1 is current with the

VoiceXML 2.0 Candidate Recommendation.

Table of Contents

Appendices

1 Introduction

This section is *informative.*

This document defines version 1.1 of the XHTML+Voice profile. XHTML+Voice 1.1 is a member of the XHTML family of document types, as specified by XHTML Modularization [XHTML Modularization]. XHTML is extended with a modularized subset of VoiceXML 2.0, the XML-Events module, and a module containing a small number of attribute extensions to both XHTML and VoiceXML. The latter module facilitates the sharing of multimodal input data between the VoiceXML dialog and XHTML input and text elements.

The XML-Events module [XML Events] provides XML host languages the ability to uniformly integrate event listeners and associated event handlers with Document Object Model (DOM) Level 2 [DOM2 Events] event interfaces. The result is an event syntax for XHTML-based languages that enables an interoperable way of associating behaviors with document-level markup.

VoiceXML [VoiceXML 2.0] has been designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed-initiative conversations. In this document, VoiceXML 2.0 is modularized to prepare it for integration into the XHTML family of languages using the XHTML modularization framework. The modules that combine to support speech dialogs for updating

XHTML forms and form elements are selected to be added to XHTML. The modules are described as well as the integration issues. The modularization of VoiceXML 2.0 also specifies DOM event types specific to voice interaction for use with the XHTML Events module. Speech dialogs authored in VoiceXML 2.0 can then be treated as *event handlers* to add voice-interaction specific behaviors to XHTML documents. The language integration supports all of the modules defined in XHTML Modularization, and adds speech interaction functionality to XHTML elements to enable multimodal applications. The document type defined by the XHTML+Voice profile is *XHTML Host language document type* conformant.

1.1 Motivation And Applications

Two mature technologies, XHTML 1.1 [XHTML 1.1] and VoiceXML 2.0 [VoiceXML 2.0] are integrated using [XHTML Modularization] to bring spoken interaction to the visual web. The design leverages open industry APIs like the W3C DOM to create interoperable web content that can be deployed across a variety of end-user devices. Multiple modes of interaction are synchronized and integrated using the DOM 2 Events model [DOM2 Events] and exposed to the content author via XML Events [XML Events].

Today, web applications are authored in XHTML with user interaction created via XHTML form elements. The W3C is presently working on XForms [XForms], the next generation of web forms that bring the power of XML to web application development. The combination of XHTML and Voice described in this document can leverage the semantic richness of web applications created using XForms, while providing a smooth transition for today's developers wishing to deploy multimodal applications by adding spoken interaction to present-day web content. Integrating the work of the W3C voice browser working group into mainstream XHTML content has the advantage of ensuring that future enhancements to the voice browser component such as natural language understanding will be incorporated. Thus, a smooth transition path for web developers wishing to deliver increasingly smart user interaction for their web applications is provided. Building on XHTML Basic [XHTML Basic] and XHTML modularization, content developers will be able to deploy their content to a wide variety of end-user clients ranging from mobile phones and small PDAs to desktop browsers.

1.2 Design Principles

XHTML+Voice is an XML application [XML 1.0].

1. XHTML is the host language.

2. XHTML+Voice extends XHTML Basic with a subset of VoiceXML 2.0, as well as XML-Events and a small extension module.
3. XHTML+Voice makes authoring easy for common types of multimodal interactions.
4. VoiceXML is modularized to permit the creation of profiles that match different application deployment environments.
5. Those parts of VoiceXML that relate to the VoiceXML document being a stand-alone speech application are omitted from the XHTML+Voice profile.
6. VoiceXML modularization does not alter the VoiceXML execution model. Specifically, a speech dialog is run as specified by the VoiceXML form interpretation algorithm.
7. VoiceXML modularization does not modify the function of the VoiceXML 2.0 elements and attributes that are part of the profile.

1.3 XHTML+Voice Processing Model

XHTML+Voice is designed for creating multimodal dialogs that combine in a straightforward way the visual input mode represented by XHTML, and speech input and output, as represented by VoiceXML. Here is a "Hello World" example of XHTML+Voice:

```
<?xml version="1.0"?>
<html
xmlns="http://www.w3.org/1999/xhtml"
xmlns:vxml="http://www.w3.org/2001/vxml"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
>
  <head>
    <title>XHTML+Voice Example</title>
    <!-- voice handler -->
    <vxml:form id="sayHello">
      <vxml:block><vxml:prompt xv:src="#hello"/>
      </vxml:block>
    </vxml:form>
  </head>
  <body>
    <h1>XHTML+Voice Example</h1>
    <p id="hello" ev:event="click" ev:handler="#sayHello">
      Hello World!
    </p>
  </body>
</html>
```

The speech dialog identified by "sayHello" is activated when the user clicks anywhere on the paragraph identified by "hello." The speech dialog is a VoiceXML form that synthesizes the text obtained from the same paragraph that activated the form. The speech output is "Hello World!"

1.3.1 Processing within one Document

A speech dialog is defined within XHTML+Voice as a [VoiceXML 2.0] form with a unique ID. The VoiceXML form is activated by an XML-event with an associated handler that references the form's unique ID. The XML-event is generated from a user interaction with an XHTML element, generally a form control, or from a document event such as load or unload. Activating the VoiceXML form sets all form and field item variables to their initial values. This clears the the guard conditions on all form items that don't have an initial value set with the *expr* attribute. The form is run according to the form interpretation algorithm (FIA) specified by VoiceXML.

1.3.1.1 Language and Version

A VoiceXML form requires language and VoiceXML version information. VoiceXML 2.0 includes language and version attributes with its root <vxml> element. XHTML+Voice obtains language and VoiceXML version from XHTML as follows. Language is obtained from the HTML root element's *xml:lang* attribute, while the VoiceXML version can be derived from the value of the VoiceXML namespace. The language can be overriden by the *xml:lang* attribute on the VoiceXML grammar and prompt tags.

1.3.1.2 VoiceXML Scope within XHTML+Voice

A VoiceXML form within an XHTML+Voice document does not have the session and document scopes defined by VoiceXML. It does not have these scopes for two reasons. First, <form> is the top level VoiceXML element in an XHTML+Voice document. Second, XHTML+Voice does not allow transitions from one voice handler to another. VoiceXML 2.0 allows a form to have either dialog or document scope. If the form's scope is document, as set by the *scope* attribute, the form is active while another form in the document is running. When the speech input matches the grammar of the form with document scope, there is a transition from the currently running form to the form with the document scope. XHTML+Voice does not allow this transition. Consequently, a form's scope is limited to dialog and the *scope* attribute is omitted. The grammar *scope* attribute is also omitted for the same reason. The remaining inner VoiceXML scopes, dialog and anonymous, are the same in XHTML+Voice, as required by the VoiceXML FIA.

XHTML+Voice allows a speech dialog to be referenced as a voice handler in an external file. Because the speech dialog has no scope outside of its enclosing form, only the form in the external file is processed when the form is activated. For example, the script elements in the external file will not be processed. This is because the visual browser only executes script in the current document, and the VoiceXML <script> element is not supported. This requires the external reference to contain a fragment identifer specifying the form in addition to an absolute or relative URI. This differs from VoiceXML, which specifies that when the fragment is absent, the form "invoked is the lexically first dialog in the document" [VoiceXML 2.0]. With this restriction, the speech dialog can reside in any external XML document, including VoiceXML. Only the calling document has to be an XHTML+Voice document.

Because XHTML script placed in an external file is not processed, validation of VoiceXML results cannot be performed within an external subdialog by calling out to some ECMAScript contained within a VoiceXML script tag. ECMAScript validation of subdialog results can only be performed from the calling document. Validation methods must be included in the ECMAScript objects passed as parameters to the subdialog.

VoiceXML <field>, <subdialog>, and <var> elements do not have any visibility to the XHTML namespace as ECMAScript variables. Furthermore, there is no requirement to support the VoiceXML elements as nodes in the DOM object available to JavaScript. There are several problems with supporting the DOM object. Unlike XHTML form control elements, VoiceXML form item elements don't have a *value* attribute and consequently the DOM node value property is missing. A *value* attribute is necessary because the VoiceXML form item elements are their own ECMAScript variables, and they have defined values only while the enclosing form is active. At all other times their values are undefined.

1.3.1.3 Accessing Speech Dialog Results from XHTML

Speech dialog results may be accessed from XHTML in one of the following ways:

1. The VoiceXML standard application variables are available to an XHTML+Voice application as global ECMAScript variables. Each variable listed is an array of elements [0..i..n], where each element represents a possible result. See [VoiceXML 2.0] for more details:
   - application.lastresult$[i].confidence
   - application.lastresult$[i].utterance
   - application.lastresult$[i].inputmode
   - application.lastresult$[i].interpretation
2. The XHTML+Voice <sync> element is described in XHTML+Voice Extension Module.

1.3.1.4 Accessing XHTML from a Speech Dialog

The global JavaScript scope of an XHTML+Voice document is available to a speech dialog. For example, an XHTML form control element, such as input, can be accessed from within VoiceXML using the DOM object traversal notation available to JavaScript. For example, the value of an input field with name "from_city" is set from the VoiceXML assign tag as follows:

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <form id="form_id" xmlns="www.w3.org/2001/vxml">
      <field name="from_field" expr="start">
        <filled>
          <assign name="document.main.from_city.value"
                  expr="from_field"/>
        </filled>
      </field>
    </form>
  </head>
  <body>
    <form id="main" action="">
      <input name="from_city" type="text"/>
    </form>
  </body>
</html>
```

The document keyword in XHTML+Voice refers to the JavaScript DOM object. The application.lastresult$ variables are at the same scope as the DOM object, which is effectively the VoiceXML application scope.

1.3.2 Cancel

Multiple speech dialogs running simultaneously are not allowed by XHTML+Voice. A speech dialog runs in its own thread and, for many devices, the audio subsystem can be owned by only one thread at one time. Also, other resources that are guaranteed to be thread-safe may cause a voice handler to indefinitely block. Therefore, only one speech dialog can be running at one time per loaded XHTML+Voice document. If only one speech dialog can be running at one time, the activating speech dialog must cancel the currently running dialog. This is the default behavior. The running dialog should also be canceled when the current XHTML+Voice document is unloaded.

A speech dialog may be canceled for other reasons, which depend on the multimodal browser implementation and user preference. For example, A speech dialog may be canceled whenever the user leaves the current XHTML form control or clicks on another element, i.e., whenever an HTML 4.01 "onblur" event is generated. An input timeout may also cancel the current speech dialog. However, if a cancel button or some other means to cancel is supplied, then only the default behavior will in most cases be preferred. It is preferred in cases where the XHTML+Voice application is running in "voice-only" mode while the user is working in another window or application. An e-mail application, for example, should allow a "voice-only" mode to run after losing application focus. The multimodal browser may also have a user preference for cancel which would override the default behavior. A good strategy would be for the multimodal browser to cancel only upon activation of a new speech dialog by default, and provide a user-preference for cancelling upon an "onblur" event.

The document author can cancel the currently running speech dialog with the <cancel> element that can be specified by an XHTML element as a handler for an XML Event. The XHTML+Voice Extension Module section provides more details.

Cancel is a message from the visual browser that must be handled by the VoiceXML FIA. It is separate from the cancel event supported by VoiceXML that cancels the currently running prompt. The cancel message from the visual browser modifies the FIA in the sense that it must be checked throughout the FIA, and if it is received then the FIA must terminate.

1.3.3 Declarative Synchronization of Input Modes

XHTML+Voice 1.1 supports a declarative synchronization of XHTML form control elements and the VoiceXML <field> element. XHTML+Voice 1.1 introduces *sync* as a standalone element. Sync specifies the following behaviors. First, sync allows input from one speech or visual modality to set the field in the other modality. Second, setting the focus of an <input> element that is synchronized with a VoiceXML field updates the FIA to visit that VoiceXML field. This is useful when there are multiple fields within a VoiceXML form. Sync is both a message to the VoiceXML FIA from the visual browser, like cancel, and a message from the FIA to the visual browser. The XHTML+Voice Extension Module section provides more details.

1.3.4 Events and Event Handling

The nomatch, noinput, help, and error VoiceXML event types are propagated as XML-events to XHTML. They can be linked to a Javascript handler using the XML-events syntax for specifying target, observer, event, and handler. The events are propagated regardless of whether the event has already been caught and handled properly within the VoiceXML form. Within VoiceXML a chain of events can be created, where one event is caught and another event is thrown, and so on. Because the entire chain of events is propagated to XHTML, the application author should be careful not to chain multiple events of the same type. The VoiceXML error event subtypes error.semantic, error.badfetch, error.unsupport.element, etc., are propagated as the error event type to XHTML. This is in accordance with the VoiceXML specification. This allows for the user to define additional error subtypes that can be handled by the visual browser. More general user-defined event types are also supported. If a user-defined event type is defined within the VoiceXML form, such as "foo.bar", then when that event is thrown within the form, it is propagated to XHTML as an XML-event. For the example below, both the noinput and foo.bar events are handled by the visual browser via the XML-events listener tag. Note that the VoiceXML form exits because the foo.bar event is not handled within the form. Throwing an unhandled foo.bar event is like throwing an unhandled exit event, except that the foo.bar event is propagated to XHTML before the form exits.

```
<vxml:form id="ex1">
   <vxml:catch event="noinput">
      <vxml:throw event="foo.bar"/>
   </vxml:catch>

   <vxml:field name="f1">
      <vxml:grammar type="boolean"/>
      <vxml:prompt>Say yes or no</vxml:prompt>
   </vxml:field>
</vxml:form>

<ev:listener ev:target="ex1" ev:event="noinput" ev:handler="#h1"/>
<ev:listener ev:target="ex1" ev:event="foo.bar" ev:handler="#h2"/>
```

In addition to the VoiceXML event types listed above, XHTML+Voice supports the *vxmldone* event type. The vxmldone event is generated when the currently running VoiceXML form completes. All the event types that XHTML+Voice supports are listed in the XML-Events Module.

1.3.5 Document Linking with Voice

Document linking with voice is available to the author. Given an XHTML+Voice document with the following link and a tags:

```
<link rel="glossary" title="Glossary" href="glossary.html"/>
<link rel="contents" title="Contents" href="contents.html"/>
<a href="chapter3.html" title="Next Page" rel="next">Next</a>
<a href="chapter1.html" title="Previous Page" rel="previous">Previou
```

The following grammar can be produced from parsing the document. For this example the grammar is JSGF. The grammar is collected from each link and a element in the document that has a *rel* attribute. The document author uses the *rel* attribute to enable document linking for a select set of link and a elements. For each element with a *rel* attribute, the *rel* attribute value is added to the grammar. Alternatively, the *title* attribute can be used in place of *rel* for international language support:

```
#JSGF V1.0 iso-8859-1;
grammar document-links;

public <document-links> = Glossary {this.$value="glossary.html"}
             | Contents {this.$value="contents.html"}
             | Next Page {this.$value="chapter3.html"}
             | Previous Page {this.$value="chapter1.html"};
```

The grammar scope of the grammar is document so that it is always active. While XHTML+Voice does not support authoring a grammar with document scope within a form, the multimodal browser should support grammars with document scope for document linking and command and control.

1.3.6 Aural Style Sheets

With the addition of a *src* attribute to the VoiceXML <prompt> element, XHTML+Voice 1.1 is able to support Aural style sheets declared according to [CSS2]. Within XHTML, a paragraph with *id* set to "warnPara" can be styled with the CSS "warn" class:

```
<p id="warnPara" class="warn">warning</p>
```

The CSS has visual and aural rules for class "warn." When the VoiceXML<form> processes a prompt with the *src* attribute set to that paragraph, the aural style rules for "warn" are invoked. The VoiceXML Prompt SRC Attribute Section provides a complete example.

2 VoiceXML 2.0 Modules

This section first modularizes VoiceXML 2.0 and then specifies the various VoiceXML 2.0 modules used in the creation of the XHTML+VoiceXML profile.

2.1 Modularization Of VoiceXML 2.0

The files making up the modularization of the VoiceXML 2.0 SCHEMA are available as voice-xml-modules.zip and have been created to ease the process of integrating VoiceXML 2.0 and XHTML. These modules do not change the VoiceXML 2.0 language as specified by the voice browser working group of the W3C. This section gives a high-level overview of each module.

Module	Purpose	Elements	XHTML+VoiceXML?
Events	Events thrown by Voice XML processor	`catch` `help` `noinput` `nomatch` `error` `throw`	Y
Executable statements	Statements for use in voice handlers	`assign` `clear` `var` `log` `reprompt`	Y
Filled	Voice handlers invoked when a slot is filled.	`filled`	Y
Flow control	Flow control constructs from VoiceXML	`if` `else` `elseif` `return`	Y
Forms	Encapsulate voice dialogs	`form` `field` `record` `subdialog` `block` `initial` `option`	Y
Miscellaneous	Non-local transfers in VoiceXML	`exit` `goto` `link` `script` `submit`	N
Menus	VoiceXML menus	`menu` `choice` `enumerate`	N
Object	Foreign objects for VoiceXML	`object`	N
	Specifying		

Resources	resources for VoiceXML	`param` `property`	Y
Root	VoiceXML stand-alone documents	`vxml` `meta`	N
Output	Speech and audio output	`prompt` `value` `audio` `emphasis` `voice` `break` `prosody` `say-as` `phoneme` `paragraph` `p` `sentence` `s` `mark`	Y
Telephony	Telephony control	`transfer` `disconnect`	N
User Input	Speech input constructs from VoiceXML	`grammar` `count` `example` `token` `import` `item` `one-of` `rule` `ruleref`	Y
Attributes	Common attributes used in VoiceXML	NA	Y
Datatypes	Common datatypes used in VoiceXML	NA	Y
Document Model	Defines content model for VoiceXML elements	NA	N

Table 1: VoiceXML Modules

2.2 Speech Dialogs

Modules `vxml-exec-1.xsd`, `vxml-filled-1.xsd`, `vxml-resource-1.xsd` `vxml-flow-1.xsd`, and `vxml-form-1.xsd` support authoring handlers that implement speech dialogs.

2.3 Executable Content

Modules `vxml-filled-1.xsd`, `vxml-flow-1.xsd`, `vxml-exec-1.xsd`, and `vxml-resource-1.xsd` declare constructs for use within voice handlers. The semantics of these constructs are as defined in the VoiceXML 2.0 specification.

2.4 Speech Grammars

The speech grammar modules provide constructs for authoring speech grammars as specified in VoiceXML 2.0. The modules are provided by the normative VoiceXML 2.0 SCHEMA and are unchanged: `grammar-core.xsd`, `grammar.xsd`, `vxml-grammar-restriction.xsd`, and `vxml-grammar-extension.xsd`. The restriction and extension modules allow the elements and attributes normatively specified by the speech grammar specification [Speech Grammars] to be included within the VoiceXML 2.0 namespace.

2.5 Speech And Non-speech Audio Output

The speech and audio output modules define constructs for producing spoken and non-spoken audio output. The modules are provided by the normative VoiceXML SCHEMA and are unchanged: `synthesis-core.xsd`, `synthesis.xsd`, `vxml-synthesis-restriction.xsd`, and `vxml-synthesis-extension.xsd`. As with the speech grammar modules, the elements and attributes normatively defined in the SSML specification [SSML 1.0] are included within the VoiceXML 2.0 namespace.

2.6 Event Handling

Module `vxml-events-1.xsd` declares the event types defined in VoiceXML 2.0

3 XHTML Modularization

This section is *normative*.

3.1 Document Conformance

A conforming XHTML+Voice document is a document that requires only the facilities described as *mandatory* in this specification. Such a document *must* meet all of the following criteria:

1. It must validate against the XML Schema found in schema provided in this document.
2. The root element of the document must be html.
3. The name of the default namespace on the root element must be the XHTML namespace name: `http://www.w3.org/1999/xhtml`.
4. If a DOCTYPE declaration is present and includes a public identifier, the DOCTYPE declaration must reference the DTD provided in this document using its Formal Public Identifier. The system identifier may be modified appropriately.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML+Voice 1.1//EN"
"http://www.w3.org/Voice/Group/2003/xhtml+voice11.dtd">
```

3.2 User Agent Conformance

The user agent must conform to the "User Agent Conformance" section of the XHTML specification ([XHTML 1.0], section 3.2) and the conformance requirements detailed in the VoiceXML modules ([VoiceXML 2.0]) supported by the integration profile.

The user agent must conform to the following additional user agent rule:

1. When the user agent claims to support facilities defined within the VoiceXML 2.0 specifications or facilities required by this specification through normative reference, it must do so in ways consistent with the facilities' definition.

3.3 XHTML Namespace Integration

The default XML namespace of an XHTML+Voice document is XHTML. XHTML+Voice extends XHTML with VoiceXML, XML-events, and XHTML+Voice extensions. The VoiceXML, XML-events, and XHTML+Voice extension elements and attributes are included through additional namespace declarations:

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:vxml="http://www.w3.org/2001/vxml"
      xmlns:ev="http://www.w3.org/2001/xml-events"
```

```
        xmlns:xv="http://www.voicexml.org/2002/xhtml+voice">
```

The name of the unique prefix identifier for the namespace within the document, for example, `vxml` for VoiceXML elements, is left to the document author's discretion.

3.4 XHTML+Voice Profile

The XHTML functionality in the XHTML+Voice document type is based upon the XHTML modules defined in [XHTML Modularization]. The XHTML+Voice profile includes the XHTML modules defined in [XHTML Basic], such as the basic XHTML forms and tables modules. Added to the XHTML Basic modules are the following modules:

- The XHTML scripting module.
- XML Events as defined by the XML Events module, [XML Events]. XML-events with VoiceXML event types and handlers allow the XHTML author to associate voice-interaction specific behaviors.
- A set of VoiceXML modules for speech-enabling XHTML constructs. The top level VoiceXML element for defining a voice handler is <form>.
- An XHTML+Voice Extension module for facilitating the authoring of the interaction between the visual and speech modules.

The notation, terms and document conventions used here are borrowed from [XHTML 1.1].

The profile includes the XHTML basic module defined in [XHTML Basic], the XHTML scripting module defined in [XHTML 1.1], the XML Event module defined in [XML Events], the XHTML+Voice extension module defined in the XHTML+Voice Extension Module, and the following VoiceXML 2.0 modules:

- Speech Dialogs
- Executable Content
- Speech Grammars
- Speech and non-speech audio Output
- Event Handling

3.5 XHTML+Voice Abstract Modules

The namespaces used in these modules are as follows:

XHTML:

http://www.w3.org/1999/xhtml

VoiceXML:
http://www.w3.org/2001/vxml

XML Events:
http://www.w3.org/2001/xml-events

XHTML+Voice:
http://www.voicexml.org/2002/xhtml+voice

3.5.1 Abstract Modules

Element	Content	Attributes
Base Module (XHTML)		
base	EMPTY	href* (URI)
Basic Forms Module (XHTML)		
form	Heading | Block - form	Common, action* (URI), method ("get"* | "post"), enctype (ContentType)
input	EMPTY	Common, Access, checked ("checked"), maxlength (Number), name (CDATA), size (Number), src (URI), type ("text"* | "password" | "checkbox" | "radio" | "submit" | "reset" | "hidden"), value (CDATA)
label	(PCDATA | Inline - label)*	Common, accesskey (Character), for (IDREF)
select	option+	Common, multiple ("multiple"), name (CDATA), size (Number)
option	PCDATA	Common, , selected ("selected"), value (CDATA)
textarea	PCDATA	Common, Access, cols* (Number), name (CDATA), rows* (Number)
Basic Tables Module (XHTML)		
caption	(PCDATA | Inline) *	Common

table	caption?, tr+	Common, summary (Text), width (Length)
td	(PCDATA | Flow - table)*	Common, Cell, Align
th	(PCDATA | Flow - table)*	Common, Cell, Align
tr	td+	Common, Align
Events Module (VoiceXML)		
catch	Exec	VoiceHandler, event (NMTOKENS)
help	Exec	VoiceHandler
noinput	Exec	VoiceHandler
nomatch	Exec	VoiceHandler
error	Exec	VoiceHandler
throw	EMPTY	VoiceHandler, event (NMTOKEN), eventexpr (Script), message (CDATA), messageexpr (Script)
Executable Statements Module (VoiceXML)		
assign	EMPTY	Expr
clear	EMPTY	namelist (CDATA)
var	EMPTY	Expr
log	(PCDATA | value) *	label (CDATA), expr (Script)
reprompt	EMPTY	-
Filled Module (VoiceXML)		
filled	(Exec)*	mode("any" | "all"*), namelist (CDATA)
Flow Control Module (VoiceXML)		
if	(Exec | elseif | else)*	cond (Script)
else	EMPTY	-

elseif	EMPTY	cond (Script)
return	EMPTY	namelist (CDATA), event (NMTOKEN), eventexpr (Script), message (CDATA), messageexpr (Script)
Forms Module (VoiceXML)		
form	(Form)*	id (ID)
field	(Audio | EventHandler | filled | grammar | link | vxml:option | prompt | property) *	Item, type (GrammarType), slot (NMTOKEN), modal (Boolean), xv:id (ID)
record	(Audio | EventHandler | filled | grammar | prompt | property) *	Item, type (ContentType), beep (Boolean), maxtime (Duration), modal (Boolean), dtmfterm (Boolean), finalsilence (Duration)
subdialog	(Audio | filled | param | prompt | property)*	Item, Cache, Submit, src (URI), srcexpr (Script), fetchaudio (URI)
block	Exec	Item
initial	(Audio | EventHandler | link | prompt | property)*	Item
vxml:option	PCDATA	dtmf (CDATA), value (CDATA)
Hypertext Module (XHTML)		
a	(PCDATA | Inline - a)*	Common, Access, Linking, hreflang (LanguageCode)
Image Module (XHTML)		
img	EMPTY	Common, Dim, alt* (Text), longdesc (URI), src* (URI)
Link Module (XHTML)		
link	EMPTY	Linking , media (MediaDesc)

List Module (XHTML)		
dl	(dd | dt)+	Common
dt	(PCDATA | Inline) *	Common
dd	(PCDATA | Flow)*	Common
ol	li+	Common
ul	li+	Common
li	(PCDATA |Flow)*	Common
Metainformation Module (XHTML)		
meta	EMPTY	I18N, content* (CDATA), http-equiv (NMTOKEN), name (NMTOKEN), scheme (CDATA)
Object Module (XHTML)		
object	(PCDATA | Flow | param)*	Common, Dim, archive (URI), classid (URI), codebase (URI), codetype (ContentType), data (URI), declare ("declare"), name (CDATA), standby (Text), tabindex (Number), type (ContentType)
param	EMPTY	id (IDREF), name* (CDATA), type (ContentType), value (CDATA), valuetype ("data"* | "ref" | "object")
Output Module (VoiceXML)		
prompt	(Audio | TTS)*	I18N, VoiceHandler, bargein (Boolean), bargeintype ("speech" | "hotword"), timeout (Duration), xv:src (URI)
value	EMPTY	expr (Script)
audio	(Audio | TTS)*	Cache, src (URI), expr (Script)
emphasis	SentenceContent	level ("strong" | "moderate"* | "none" | "reduced")
		I18N, gender ("male" | "female"

voice	(SentenceContent | Structure)*	| "neutral"), age (Number), variant (Number), name (CDATA)
break	EMPTY	size ("large" | "medium"* | "small" | "none"), time (Duration)
prosody	(SentenceContent | Structure)*	pitch (CDATA), contour (CDATA), range (CDATA), rate (CDATA), duration (Duration), volume (CDATA)
say-as	(PCDATA | value)*	type (SayAsType)
phoneme	PCDATA	ph (CDATA), alphabet (CDATA)
paragraph, p	(SentenceContent | Sentence)*	I18N
sentence, s	SentenceContent	I18N
mark	(SentenceContent | Sentence)*	name (IDREF)
Resources Module (VoiceXML)		
param	EMPTY	Expr, value (CDATA), valuetype ("data"* | "ref"), type (CDATA)
property	EMPTY	name (NMTOKEN), value (CDATA)
Scripting Module (XHTML)		
script	PCDATA	charset (CharSet), defer ("defer"), src (URI), type* (ContentType), xml:space="preserve"
noscript	(Heading | Block | List)+	Common
Structure Module (XHTML)		
body	(Heading | Block | List)*	Common

head	title, (meta | link | object | script | vxml:form | ev:listener | xv:sync | xv:cancel)*	I18N, profile (URI)
html	head, body	I18N, version (CDATA), xmlns (URI = "http://www.w3.org/1999/xhtml")
title	PCDATA	I18N
Text Module (XHTML)		
abbr	(PCDATA | Inline) *	Common
acronym	(PCDATA | Inline) *	Common
address	(PCDATA | Inline) *	Common
blockquote	(PCDATA | Heading | Block | List)*	Common, cite (URI)
br	EMPTY	Core
cite	(PCDATA | Inline) *	Common
code	(PCDATA | Inline) *	Common
dfn	(PCDATA | Inline) *	Common
div	(PCDATA | Flow)*	Common
em	(PCDATA | Inline) *	Common
h1	(PCDATA | Inline) *	Common
h2	(PCDATA | Inline) *	Common
	(PCDATA |	

h3	Inline)*	Common
h4	(PCDATA | Inline)*	Common
h5	(PCDATA | Inline)*	Common
h6	(PCDATA | Inline)*	Common
kbd	(PCDATA | Inline)*	Common
p	(PCDATA | Inline)*	Common
pre	(PCDATA | Inline)*	Common, xml:space="preserve"
q	(PCDATA | Inline)*	Common, cite (URI)
samp	(PCDATA | Inline)*	Common
span	(PCDATA | Inline)*	Common
strong	(PCDATA | Inline)*	Common
var	(PCDATA | Inline)*	Common
User Input Module (VoiceXML)		
grammar	(PCDATA | meta | metadata | lexicon | rule)*	Cache, I18N, version (NMTOKEN), root (IDREF), mode ("voice"* | "dtmf"), src (URI), type (ContentType), weight (CDATA), tag-format (URI)
example	PCDATA	
lexicon	EMPTY	uri (URI), type (ContentType)
tag	PCDATA	
token	PCDATA	I18N

item	(RuleExpansion)*	I18N, weight (NMTOKEN), repeat (NMTOKEN), repeat-prob (NMTOKEN)
meta	EMPTY	name (NMTOKEN), content (CDATA), http-equiv (NMTOKEN)
metadata	ANY	
one-of	(item)+	I18N
rule	(RuleExpansion | example)*	id (ID), scope ("private"* | "public")
ruleref	EMPTY	I18N, uri (URI), type (ContentType), special ("NULL" | "VOID" | "GARBAGE")
XML Events Module (XML Events)		
listener	EMPTY	XEvents
XHTML+Voice Extension Module (XHTML+Voice)		
sync	EMPTY	input (NMTOKEN), field (URI)
cancel	EMPTY	id (ID), handler (URI)
Elements	**Attributes**	
vxml:field&	id (ID)	
vxml:prompt&	src (URI)	

Table 2: XHTML+Voice Abstract Modules

3.5.2 Element content shorthands

Element Entities	Content
Audio (VoiceXML)	PCDATA | audio | value
Block (XHTML)	address | blockquote | div | p | pre
EventHandler (VoiceXML)	catch | help | noinput | nomatch | error
Exec (VoiceXML)	Audio | assign | clear | disconnect | exit | goto | if | log | prompt | reprompt | return | script | submit | throw | var

Flow (XHTML)	Heading | List | Block | Inline
Form (VoiceXML)	EventHandler | grammar | filled | initial | object | link | property | record | subdialog | Variable
Heading (XHTML)	h1 | h2 | h3 | h4 | h5 | h6
Inline (XHTML)	a | abbr | acronym | button | br | cite | code | dfn | em | img | input | kbd | label | object | q | samp | select | span | strong | textarea
RuleExpansion (VoiceXML)	PCDATA | token | ruleref | item | one-of
SentenceContent (VoiceXML)	Audio | SentenceElements
SentenceElements (VoiceXML)	break | emphasis | phoneme | mark | prosody | say-as | voice
Structure (VoiceXML)	sentence | s | paragraph | p
TTS (VoiceXML)	SentenceElements | Structure
Variable (VoiceXML)	block | field | var

Table 3: Element Entities and Content

3.5.3 Attribute list shorthands

Attribute Entities	Content
Access (XHTML)	accesskey (Character), tabindex (Number)
Align (XHTML)	align ("left" | "center" | "right"), valign ("top" | "middle" | "bottom")
Cache (VoiceXML)	fetchhint ("prefetch" | "safe"), fetchtimeout (Duration, maxage (Number), maxstale (Number)
Cell (XHTML)	abbr (Text), axis (CDATA), colspan (Number), headers (IDREFS), rowspan (Number), scope ("row" | "col")

Common (XHTML)	Core, Events, XEvents
Core (XHTML)	class (NMTOKENS), id (ID), title (CDATA)
Dim (XHTML)	height (Length), width (Length)
Events (XHTML)	MouseEvents , KeyEvents
Expr (VoiceXML)	name (VarName), expr (Script)
I18N (XML)	xml:lang (NMTOKEN)
Item (VoiceXML)	name (VarName), cond (Script), expr (Script)
KeyEvents (XHTML)	onkeypress (Script), onkeydown (Script), onkeyup (Script)
Linking (XHTML)	charset (CharSet), href (URI), hreflang (LanguageCode), rel (LinkTypes), rev (LinkTypes), type (ContentType)
MouseEvents (XHTML)	onclick (Script), ondblclick (Script), onmousedown (Script), onmouseover (Script), onmousemove (Script), onmouseout (Script)
Next (VoiceXML)	next (URI), expr (Script)
Style (XHTML)	style (CDATA)
Submit (VoiceXML)	method ("get"* | "post"), enctype (ContentType), namelist (CDATA)
VoiceHandler (VoiceXML)	count (Number), cond (Script)
XEvents (XML Events)	event, observer (IDREF), handler (URI), target (IDREF), phase ("capture" | "default"*), propagate ("stop" | "continue"*), defaultAction ("cancel" | "perform"*), id

Table 4: Attribute Entities and Content

Attribute types

Attribute	

Type	Description
Boolean	"true" | "false"
Duration	A positive real number followed by either 's' (seconds) or 'ms' (milliseconds)
GrammarType	CDATA
SayAsType	"acronym" | "spell-out" |"currency" | "measure" | "name" | "telephone" | "address" | "number" | "number:ordinal" | "number:digits" | "number:cardinal" | "date" | "date:dmy" | "date:mdy" | "date:ymd" | "date:ym" | "date:my" | "date:md" | "date:y" | "date:m" | "date:d" | "time" | "time:hms" | "time:hm" | "time:h" | "duration" | "duration:hms" | "duration:hm" | "duration:ms" | "duration:h" | "duration:m" | "duration:s" | "net" | "net:email" | "net:uri" | "vxml:date" | "vxml:boolean" | "vxml:currency" | "vxml:time" | "vxml:digits" | "vxml:number" | "vxml:phone"
VarName	NMTOKEN or NMTOKEN with "$" appended

Table 5: Attribute Types

4 XML-Events Module

This section is *normative*.

4.1 Listener

XHTML+Voice extends XHTML with the XML-Events <listener> element and its attributes. The <listener> attributes are added to XHTML elements primarily for activating voice handlers. The <listener> element and attributes belong to the XML-Events namespace:

```
xmlns:ev="http://www.w3.org/2001/xml-events"
```

4.2 Event Types

For a given XML language extended with XML Events, a set of event types must be specified independently of the [XML Events] module. The XML Event types supported by the XHTML+Voice profile include all event types defined for [HTML

4.01] intrinsic events. VoiceXML handler activation is specified by including with an XHTML element one of these event types as an XML event and an ID reference to the VoiceXML form as an XML event handler.

The XHTML+Voice profile supports the following VoiceXML 2.0 event types: nomatch, noinput, error, and help. The VoiceXML exit and cancel event types are supported within the VoiceXML form but are not propagated to the visual browser. Event types defined by the author within VoiceXML, also known as user-defined event types, are also propagated to the visual browser. However, the VoiceXML <form> element does not support adding <listener> attributes.

An additional XHTML+Voice event type, "vxmldone", is supported. The vxmldone event is generated when the voice handler completes.

The XHTML+Voice profile extends the XHTML <script> element with XML Events. The <script> element doesn't generate any events of its own, so the *target* attribute is required to specify capturing an XML event. The <script> element can target any XHTML or VoiceXML element and can specify any HTML 4.01 intrinsic event or VoiceXML event. Here is an example of how a <script> element can be a handler for a "vxmldone" event. The value of XHTML input "drink" is updated when the voice handler "fid" completes:

```
<script type="text/javascript" ev:event="vxmldone" ev:target="fid">
   document.xform.drink.value = application.lastresult$[0].utterance
</script>
<vxml:form id="fid">
   <vxml:field name="f1">
      <vxml:grammar src="drink.gram"/>
      <vxml:prompt>Coffee, tea, or milk?</vxml:prompt>
   </vxml:field>
</vxml:form>

<body>
<form id="xform" action="cgi/submit">
<input type="text" id="drink" ev:event="focus" ev:handler="#fid"/>
</form>
```

The following table matches the XHTML+VoiceXML event types with the XHTML or VoiceXML elements that support them. When the <listener> *event* attribute is added to an XHTML element, it must specify a event type supported by the element in the right-hand column. Because the HTML 4.01 event types have been translated into XML-event types, the "on" prefix for these event types have been removed.

Elements	Event Type
XHTML body	load, unload
Most XHTML elements	click, dblclick, mousedown, mouseup, mouseover, mouseout, keypress, keydown, keyup
XHTML elements: a, label, input, select, textarea, button	focus, blur
XHTML form	submit, reset
XHTML elements: input, textarea	select
XHTML elements: input, select, textarea	change
VoiceXML form	nomatch, noinput, error, help, vxmldone, "user defined"

Table 6: XHTML+VoiceXML Event Types

5 XHTML+Voice Extension Module

This section is *normative*.

The XHTML+Voice Extension module extends XHTML+Voice 1.0 with the <sync> element, <cancel> element, the *src* attribute of the VoiceXML <prompt> element, and the *id* attribute of the VoiceXML field element. The element and attributes in this module belong to their own namespace:

```
  xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
```

The value of the namespace is temporary until the XHTML+Voice specification is taken over by the W3C Voice Browser Working Group. At that time the Voice Browser Working Group will obtain from the W3C an official XHTML+Voice namespace and location for the XHTML+Voice schema.

5.1 Sync

The XHTML+Voice <sync> element adds support for synchronization of data entered via either speech or visual input. It binds the value property of the input field, or JavaScript variable, to the VoiceXML field with the given *id* attribute value. This means several things:

1. Speech dialog results are returned to both the VoiceXML field and the XHTML <input> element, or JavaScript variable.
2. Keyboard data entered into the <input> element update both the VoiceXML field and the XHTML <input> element.
3. Keyboard data entered into the <input> element satisfies the guard condition on the VoiceXML field.
4. For an active VoiceXML form with multiple fields, if the user gives focus to the input field, the FIA is instructed to visit the referenced VoiceXML field as the next item. This includes the mixed initiative case.

Sync does not activate a voice handler. This means that if the <sync> element has specified an XHTML input field but no VoiceXML form is currently active, nothing will happen when a user clicks on the input field unless an XML-event and event handler are also specified for the input field. If an event and event handler are specified, then when the user clicks on the input field the VoiceXML form is activated and the guard conditions of the VoiceXML form items are cleared. The XHTML input field is not cleared if data is already there.

The <sync> element attributes are:

input	The name of an XHTML input field or Javascript variable.
field	A URI reference to a field ID within a VoiceXML form.

Table 7: <sync> attributes

The type of the *input* attribute is NMTOKEN. The type of the *field* attribute is URI. The URI must include a fragment identifier that references a VoiceXML <field> ID. If the <field> element is in an external file, then the fragment identifier is appended to the URI.

The What You See Is What You Can Say and Mixed-initiative Conversational Interface examples both use the <sync> element to synchronize XHTML inputs and VoiceXML fields.

5.2 Cancel

The XHTML+Voice <cancel> element allows a document author to cancel a running speech dialog. It is a stand-alone element with no content that can be referenced as an XML event handler. The <cancel> element has two attributes, *id* and *handler*. The *id* attribute is required. The optional *handler* attribute references the *id* attribute of a voice handler form. If the *handler* attribute is omitted, then the currently running speech dialog is canceled. If *handler* is specified, then only the specified voice handler is canceled.

The <cancel> element attributes are:

id	Unique document identifier.
handler	A URI reference to a VoiceXML form ID.

Table 8: <cancel> attributes

The type of the *id* attribute is ID. The type of the *handler* attribute is URI. The URI must include a fragment identifier that references a VoiceXML <form> ID. If the <form> element is in an external file, then the fragment identifier is appended to the URI.

```
<head><title>Cancel Example</title>
...
<xv:cancel id="cid1" handler="#fid1"/>
<xv:cancel id="cid2"/>
</head>
<body>
...
<input type="reset" ev:event="click" ev:handler="#cid1"/>
<button ev:event="click" ev:handler="#cid2">Cancel Voice</button>
```

The example above shows how <cancel> can be used to cancel either a specific speech dialog or the currently running speech dialog. The reset button in the example cancels the speech dialog identified by "fid1." The "Cancel Voice" button cancels the currently running dialog because the *handler* attribute is omitted from the <cancel> element that is activated when the button is clicked.

5.3 VoiceXML Field ID Attribute

The VoiceXML field does not have an *id* attribute and it is required to support the XHTML+Voice <sync> element extension.

5.4 VoiceXML Prompt SRC Attribute

XHTML+Voice 1.1 extends the VoiceXML <prompt> element with a *src* attribute. The SRC attribute allows for the specification of a text source for speech output anywhere in the document or in an external document. In addition, the text source can be styled according to the aural styling rules defined in [CSS2]. For example, a style sheet may have the following styling rules for the XHTML <p> element:

```
P.romeo { voice-family: male; volume: loud; pause-before: 20ms; }
P.juliet { voice-family: female; volume: soft; }
```

A voice handler can play two prompts from two different text sources in the document, as follows:

```
<vxml:form id="sayHello">
  <vxml:block><prompt xv:src="#hello_romeo"/>
              <prompt xv:src="#hello_juliet"/>
  </vxml:block>
</vxml:form>
<body ev:event="load" ev:handler="#sayHello">
<p id="hello_romeo" class="juliet">
    Romeo, Romeo, where art thou?
</p>
<p id="hello_juliet" class="romeo">
    I am here.
</p>
</body>
```

The first prompt plays a soft female voice. The second prompt plays a loud male voice after a 20 ms pause.

A Reusable VoiceXML

This section is *informative*.

A VoiceXML form, defined here as an event handler, is more practical if it can be placed in a linked document separate from the XHTML as a reusable component. A reusable component allows for easier maintenance, and provides a default behavior that can be used as an application building-block. VoiceXML includes a subdialog construct and its calling convention is close to what is required for a reusable component. The problem is that the caller must know both the subdialog's parameters and the fields included in the ECMAScript object returned to the caller.

It is not within the scope of this profile to attempt to solve the problem of creating authentic reusable components within VoiceXML; this is the domain of the W3C Voice Working Group. Authoring conventions can, however, be suggested which should work in most cases. A VoiceXML handler can be placed in a separate file and linked from within an XHTML+VoiceXML profile document if:

- The handler is a subdialog that has a fixed number of parameters and return fields that are named according to a fixed naming convention.
- The subdialog is called by a VoiceXML form within the XHTML+VoiceXML profile document. The calling VoiceXML form is the handler activated by an XML event.
- The subdialog is referenced explicitly with a fragment identifier at the end of an absolute or relative URI. With this restriction, the subdialog can be placed in any valid XML document, including VoiceXML.

The appendix includes an example of how a subdialog can be reused by following the above authoring conventions.

B Examples

This section is *informative*.

B.1 What You See Is What You Can Say

```
<?xml version="1.0"?>
<html
xmlns="http://www.w3.org/1999/xhtml"
xmlns:vxml="http://www.w3.org/2001/vxml"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
>
  <head>
    <title>What You See Is What You Can Say</title>

      <!-- first declare the voice handlers. -->
      <!-- use vxml:form to declare a voice handler -->
      <vxml:form id="voice_city">
         <vxml:field xv:id="field_city" name="field_city">
          <vxml:grammar src="city.jsgf" type="application/x-jsgf"/>
          <vxml:prompt xv:src="#city_label"/>
          <vxml:catch event="help nomatch noinput">
            For example, say Chicago.
          </vxml:catch>
         </vxml:field>
      </vxml:form>

      <vxml:form id="voice_hotel">
         <vxml:field xv:id="field_hotel" name="field_hotel">
          <vxml:grammar src="hotel.jsgf" type="application/x-jsgf"/>
          <vxml:prompt xv:src="#hotel_label"/>
          <vxml:catch event="help nomatch noinput">
```

```
              For example, say Hilton.
            </vxml:catch>
            <vxml:filled>
                <vxml:prompt>
                    You selected <vxml:value expr="field_hotel"/>.
                 </vxml:prompt>
            </vxml:filled>
          </vxml:field>
        </vxml:form>
        <! -- done voice handlers -->

        <!-- declare inputs synchronized with VoiceXML fields -->
         <xv:sync input="city" field="#field_city"/>
        <xv:sync input="hotel" field="#field_hotel"/>
         <xv:cancel id="voice_cancel"/>
    </head>
    <body>
      <h1>What You See Is What You Can Say</h1>

      <p>This example demonstrates a simple application
        that permits the user to provide input using either
        keyboard or stylus, or speak the same information.
      </p>
      <h2>Hotel Picker</h2>
      <p>
        This voice-enabled application lets you pick a hotel.
      </p>
      <form id="hotel_query" method="post" action="cgi/hotel.pl">
        <label id="city_label">Please enter city

        <! -- input name attrib required for type "text" -->
        <input name="city" type="text"
               ev:event="focus" ev:handler="#voice_city"/>
        </label>

        <label id="hotel_label">Please enter hotel

        <input name="hotel" type="text"
               ev:event="focus" ev:handler="#voice_hotel"/>
        </label>

        <input type="submit" value="Submit"/>
        <input type="reset" value="Reset"
                ev:event="click" xv:handler="#voice_cancel"/>
      </form>
    </body>
  </html>
```

B.2 Mixed-initiative Conversational Interface

```
<?xml version="1.0"?>
<html
xmlns="http://www.w3.org/1999/xhtml"
xmlns:vxml="http://www.w3.org/2001/vxml"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
>
  <head>
    <title>Mixed Initiative Conversational Interface</title>
      <!-- first declare the voice handlers. -->
      <!-- VXML form supporting a mixed-initiative grammar -->
      <vxml:form id="voice_city_hotel">
        <vxml:grammar src="city_hotel.jsgf" type="application/x-jsgf

        <!-- Mixed initiative form begins with initial prompt -->
        <vxml:initial name="start">
           <vxml:prompt xv:src="#please_choose"/>
          <vxml:help>
            Please say the name of a city and a hotel to make
            a reservation.
          </vxml:help>
          <!-- If user is silent, reprompt once, then try
               directed prompts. -->
          <vxml:noinput count="1"><vxml:reprompt/>
          </vxml:noinput>
          <vxml:noinput count="2">
             <vxml:reprompt/>
             <vxml:assign name="start" expr="true"/>
          </vxml:noinput>
        </vxml:initial>

        <vxml:field xv:id="field_city" name="field_city">
          <vxml:grammar src="city.jsgf" type="application/x-jsgf"/>
          <vxml:prompt>Please choose a city.</vxml:prompt>
          <vxml:catch event="help nomatch noinput">
            For example, say Chicago.
          </vxml:catch>
        </vxml:field>

        <vxml:field xv:id="field_hotel" name="field_hotel">
          <vxml:grammar src="hotel.jsgf" type="application/x-jsgf"/>
          <vxml:prompt>Select your hotel.</vxml:prompt>
          <vxml:catch event="help nomatch noinput">
            For example say Hilton.
          </vxml:catch>
          <vxml:filled>
              <vxml:prompt>
                You selected <vxml:value expr="field_hotel"/>.
               </vxml:prompt>
          </vxml:filled>
        </vxml:field>
      <!-- done voice handlers -->

      <!-- declare inputs synchronized with VoiceXML fields -->
       <xv:sync input="city" field="#field_city"/>
      <xv:sync input="hotel" field="#field_hotel"/>
      <xv:cancel id="voice_cancel" handler="#voice_city_hotel"/>
  </head>
  <body>
    <h1>Mixed-Initiative Conversational Interface</h1>

    <p>In this example, we demonstrate a mixed-initiative dialog.  B
       activating a grammar capable of recognizing both cities and
       hotel names, for the entire application, the user can specify
       both hotel and city in a single utterance.  Alternatively,
       the user can fill one field at a time.
    </p>

    <h2>Hotel Picker</h2>
    <p>This voice-enabled application lets you pick a
       city and a hotel.
    </p>
    <form id="visual_city_hotel" method="post" action="cgi/hotel.pl"
             ev:event="focus" ev:handler="#voice_city_hotel" >
      <p id="please_choose">
      Please choose a city and hotel where you wish to stay.
      </p>

      <!-- input name attrib required except for type "text" -->
      <input name="city" type="text"/>
      <input name="hotel" type="text"/>

      <input type="submit" value="Submit" />
      <input type="reset" value="Reset"
              ev:event="click" ev:handler="#voice_cancel"/>
    </form>
  </body>
</html>
```

B.3 Speech-Enabled Mail Interface

This email message from the W3C voice browser working group archives has been speech-enabled to allow easy browsing of email on hand-held devices.

```
<?xml version="1.0"?>
<html
xmlns="http://www.w3.org/1999/xhtml"
```

```
xmlns:vxml="http://www.w3.org/2001/vxml"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
>
  <head>
    <title>w3c-voice-wg@w3.org from October 2001: preliminary
           draft for Multimodal Activity
    </title>
    <meta name="Author" content="Dave Raggett (dsr@w3.org)" />
    <meta name="Subject"
      content="preliminary draft for W3C multimodal activity" />
    <link rel="Stylesheet"
      href="http://www.w3.org/StyleSheets/Mail/member-message"/>

    <script language="javascript">
    // define array holding command words -> activate-id map.
    var commands = new Array();
    commands.push("__next_message");
    commands.push("__prev_message");
    commands.push("__sort_by_date");
    commands.push("__sort_by_thread");
    commands.push("__sort_by_subject");
    commands.push("__sort_by_author");
    commands.push("__more_from_this_list");
    commands.push("__other_w3c_lists");
    commands.push("__respond_to_this_message");
    commands.push("__mail_new_topic");
    commands.push("__reply_to");

    //activate takes a command word, looks it up in the commands
    // map, and activates the link.
    function activate (command) {
            var length = commands.length;
            for (var i=0;i<length;i=i++) {
                if (command == commands[i]) {
                    document.getElementById(commands[i]).click();
                    break;
                }
            }
        }
    }
    </script>

    <script ev:target="#command-and-control" ev:event="vxmldone">
        activate(application.lastresult$[0].interpretation);
    </script>

    <vxml:form id="command-and-control">
        <!-- your word is my command. -->
        <vxml:field name="word">
```

```
            <vxml:grammar jsgf="mail.jsgf"/>
            <vxml:catch event="help nomatch">
            This mail reader is speech-enabled. You can
            perform available actions via speech input.
          </vxml:catch>
      </vxml:field>
    </vxml:form>
    </head>

    <body ev:event="load" ev:handler="#command-and-control">
    <h1>preliminary draft for W3C multimodal activity</h1>

    <strong>From:</strong> Dave Raggett (
<a id="__reply_to"
href="mailto:dsr@w3.org?Subject=Re:%20preliminary%20draft%20for\
%20W3C%20multimodal%20activity&amp;In-Reply-To=&lt;Pine.WNT.4.10.10\
110301232270.-1031403-100000@hazel&gt;&amp;References=&lt;Pine.WNT.\
4.10.10110301232270.-1031403-100000@hazel&gt;">
      <em>dsr@w3.org</em>
</a>)<br />
    <strong>Date:</strong> Tue, Oct 30 2001

    <p><!-- next="start" --></p>
    <ul class="noindent">
      <li><strong>Next message:</strong>
          <a id="__next_message" href="0093.html">
 mxd@cisco.com: "Re: [dialog] &lt;record&gt;'s dest attribute"
          </a>
      </li>
    </ul>
    <ul>
      <li><strong>Previous message:</strong>
          <a id="__previous_message" href="0091.html">
 Harish Varanasi: "RE: [ dialog ] &lt;record&gt;'s dest attribute"
          </a>
        <!-- nextthread="start" -->
        <!-- reply="end" -->
      </li>
      <li><strong>Messages sorted by:</strong>
          <a id="__sort_by_date" href="index.html#92">
              [ date ]</a>
          <a id="__sort_by_thread" href="thread.html#92">
              [ thread ]</a>
          <a id="__sort_by_subject" href="subject.html#92">
              [ subject ]</a>
          <a id="__sort_by_author" href="author.html#92">
               [ author ]</a>
      </li>
      <li><strong>Other mail archives:</strong>
        <a id="__more_from_this_list"  href="../">
```

```
            [ this mailing list ]</a>
        <a id="__other_w3c_lists"  href="../../">
            [ other W3C mailing lists ]</a>
      </li>

      <li><strong>Mail actions:</strong>
 <a id="__respond_to_this_message"
    href="mailto:w3c-voice-wg@w3.org?Subject=Re:%20preliminary\
%20draft%20for%20W3C%20multimodal%20activity&amp;In-Reply-To=\
&lt;Pine.WNT.4.10.10110301232270.-1031403-100000@hazel&gt;&amp;\
References=&lt;Pine.WNT.4.10.10110301232270.-1031403-100000@hazel&gt
          [ respond to this message ]</a>
 <a id="__mail_new_topic" href="mailto:w3c-voice-wg@w3.org">
     [ mail a new topic ]</a></li>
    </ul>

    <hr noshade="noshade" />
    <p><!-- body="start" --></p>
    <pre>
    Message body was here.
    </pre>
    <p><!-- body="end" --></p>

    <hr noshade="noshade" />
    <ul><!-- next="start" -->
     </ul>
    <!-- trailer="footer" -->
  </body>
</html>
```

B.4 Reusable VoiceXML Subdialogs

A flight query is processed with two reusable VoiceXML subdialogs. One subdialog processes the departure city or airport, the other the departure date.

```
<?xml version="1.0"?>
<html
xmlns="http://www.w3.org/1999/xhtml"
xmlns:vxml="http://www.w3.org/2001/vxml"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
>
  <head>
    <title>Flight Query</title>
    <link type="text/css" rel="stylesheet" href="style.css" />
    <script src="cityorairport.es">
      var objCityOrAirport = new CityOrAirport();
    </script>
    <script src="dateinfo.es">
      <var objDateInfo = new DateInfo();
    </script>

    <vxml:form id="voice_city_from">
      <vxml:subdialog name="cityorairport"
                     src="cityorairport.vxml#cityorairportform">
        <vxml:param name="paramSubdialogObj" expr="objCityOrAirport"
        <vxml:param name="paramPromptQuestion"
              expr="'What city or airport are you departing from?'"/
        <vxml:filled>
          <vxml:prompt>
          You are departing from
                   <value expr="cityorairport.returnCityOrAirport"/
          </vxml:prompt>
           <vxml:assign name="document.from"
                       expr="cityorairport.returnCityOrAirport"/>
        </vxml:filled>
      </vxml:subdialog>
    </vxml:form>

    <vxml:form id="voice_date_from">
      <vxml:subdialog name="dateinfo" src="dateinfo.vxml#dateform">
        <vxml:param name="paramSubdialogObj" expr="objDateInfo"/>
        <vxml:param name="paramPromptQuestion"
              expr="'What day, month, and year are you leaving?'"/>
        <vxml:filled>
          <vxml:prompt>
            You are departing on
                   <value expr="dateinfo.returnDateInfo"/>.
          </vxml:prompt>
             <vxml:assign name="document.fromDate"
                        expr="dateinfo.returnDateInfo"/>
          </vxml:filled>
        </vxml:subdialog>
    </vxml:form>
    <xv:cancel id="voice_cancel"/>

  </head>
  <body>
    <h1>Multimodal Flight Query</h1>

    <form method="post" action="/servlet/flightServlet">

      <table border="0"
              summary="Departure airport, date, and time">
        <tr>
          <td width="15%">
          <label for="from">Leaving From:</label>
          </td>
          <td colspan="2">
            <input type="text" id="from" size="20"
                   ev:event="click"
                   ev:handler="#voice_city_from" />
           </td>
        </tr>

        <tr>
          <td width="15%">
            <label for="fromDate">Travel Date:</label>
          </td>
          <td width="35%">
            <input type="text" id="fromDate" size="20"
                   ev:event="click"
                   ev:handler="voice_date_from"/>
          </td>
          <td width="50%">
            <div class="c1"><label>Time of Day:</label>
            <br />

              <table width="100%" border="0"
                               summary="leave am or pm">
                <tr>
                  <td align="left">
                    <input type="checkbox" id="departam"
                           value="checkbox"/>
                    <label for="departam">am</label> </td>
                  <td align="left">
                    <input type="checkbox" id="departpm"
                           value="checkbox"/>
                    <label for="departpm">pm</label></td>
                </tr>
              </table>
            </div>
          </td>
        </tr>
      </table>
      <br />
      <table align="center">
        <tr>
          <td align="center" width="80%">
            <input type="submit" value="Submit"/>
          </td>
          <td>
            <input type="reset" value="Reset"
                 ev:event="click" ev:handler="#voice_cancel"/>
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

C FIA for XHTML+Voice

This section is *informative*.

An XHTML+Voice voice handler is processed according to the VoiceXML 2.0 form interpretation algorithm (FIA). Because many of the VoiceXML tags, as well as forms with document grammar scope, are not supported, XHTML+Voice simplifies the FIA. Below is the FIA pseudo-code taken from Appendix C of [VoiceXML 2.0]. Comments are placed above the sections that are not supported and the sections are crossed-out. For example, the comment "no script tag" is above the crossed-out section that processes the VoiceXML 2.0 <script> element.

At any time during the running of the FIA a cancel message from the visual browser may have to be processed. This will terminate the FIA. The other external message is sync for declarative synchronization of visual and speech inputs. Sync modifies the selection phase of the FIA as follows. The guard condition of the field associated with the sync is cleared and the field is selected. If the sync carries with it data entered from XHTML during processing of the FIA, then sync will update the field with the data and set its guard condition. Finally, sync returns the utterance collected in the field to the visual browser.

```
  //
  // Initialization Phase
  //

  foreach ( <var>, <script> and form item, in document order )
  {
     if ( the element is a <var> )
       Declare the variable, initializing it to the value of
       the "expr" attribute, if any, or else to undefined.

     // No script tag
```

```
     else if ( the element is a <script> ) No script tag
       Evaluate the contents of the script if inlined or else
       from the location specified by the "src" attribute.
```

```
     else if ( the element is a form item )
       Create a variable from the "name" attribute, if any, or
       else generate an internal name.  Assign to this variable
       the value of the "expr" attribute, if any, or else
```

```
          undefined.

        foreach ( input item and <initial> element )
        {
          Declare a prompt counter and set it to 1.
        }
    }

    // No document-level grammars
```

```
    if ( user entered this form by speaking to its
          grammar while in a different form)
    {
        Enter the main loop below, but start in
        the process phase, not the select phase:
        we already have a collection to process.
    }
```

```
    //
    // Main Loop: select next form item and execute it.
    //

    while ( true )
    {
        //
        // Select Phase: choose a form item to visit.
        //

        // No goto
```

```
        if ( the last main loop iteration ended
                  with a <goto nextitem> )
            Select that next form item.
```

```
        // if there is a sync event, the form item associated with the
        // sync is selected after clearing its guard condition
        // else

        if (there is a form item with an
                  unsatisfied guard condition )
            Select the first such form item in document order.

        else
```

```
            Do an <exit>
```

```
            return -- the form is full and specified no transition.

        //
        // Collect Phase: execute the selected form item.
        //
```

```
// Queue up prompts for the form item.

unless ( the last loop iteration ended with
        a catch that had no <reprompt>,
       // cannot change active dialogs
```

```
        and the active dialog was not changed
```

```
                                        )
{

    Select the appropriate prompts for an input item or <initial
    Queue the selected prompts for play prior to
    the next collect operation.

    Increment an input item's or <initial>'s prompt counter.
}

// Activate grammars for the form item.

if ( the form item is modal )
    Set the active grammar set to the form item grammars,
    if any. (Note that some form items, e.g. <block>,
    cannot have any grammars).
else
    Set the active grammar set to the form item
    grammars and any grammars scoped to the form,
    the current document, and the application root
    document.

// Execute the form item.

if ( a <field> was selected )
    Collect an utterance or an event from the user.
    // If the sync event is received set the guard condition

else if ( a <record> was chosen )
    Collect an utterance (with a name/value pair
    for the recorded bytes) or event from the user.

// no <object>
```

```
else if ( an <object> was chosen )
    Execute the object, setting the <object>'s
    form item variable to the returned ECMAScript value.
```

```
else if ( a <subdialog> was chosen )
    Execute the subdialog, setting the <subdialog>'s
    form item variable to the returned ECMAScript value.
    // no <transfer>
```

```
    else if ( a <transfer> was chosen )
        Do the transfer, and (if wait is true) set the
        <transfer> form item variable to the returned
        result status indicator.
```

```
    else if ( an <initial> was chosen )
        Collect an utterance or an event from the user.
    else if ( a <block> was chosen )
    {
        Set the block's form item variable to a defined value.
        Execute the block's executable context.
    }

    //
    // Process Phase: process the resulting utterance or event.
    //

    Assign the utterance and other information about the last
    recognition to application.lastresult$. // Must have an utteranc

    // no link
```

```
    if ( the utterance matched a grammar belonging to a <link> )
      If the link specifies an "next" or "expr" attribute,
      transition to that location.  Else if the link specifies an
      "event" or "eventexpr" attribute, generate that event.
```

```
    // no choice
```

```
    else if ( the utterance matched a grammar belonging to a <choice
      If the choice specifies an "next" or "expr" attribute,
      transition to that location.  Else if the choice specifies
      an "event" or "eventexpr" attribute, generate that event.
```

```
    // no grammar outside the current <form> except command & contro
    if ( the utterance matched a grammar from outside the current <f
    {
      Transition to the command & control handler for the utterance.
    }
```

```
    else if ( the utterance matched a grammar from outside the curre
              <form> or <menu> )
    {
      Transition to that <form> or <menu>, carrying the utterance
      to the new FIA.
    }
```

```
    // Process an utterance spoken to a grammar from this form.
    // First copy utterance result property values into correspondin
```

```
// form item variables.

Clear all "just_filled" flags.

if ( the grammar is scoped to the field-level ) {
   // This grammar must be enclosed in an input item.  The input
   // has an associated ECMAScript variable (referred to here as
   // item variable) and slot name.

   if ( the result is not a structure )
     Copy the result into the input item variable.
   elseif ( a top-level property in the result matches the slot
            or the slot name is a dot-separated path matching a
            subproperty in the result )
     Copy the value of that property into the input item variabl
   else
     Copy the entire result into the input item variable

   Set this input item's "just_filled" flag.
}
else {
   foreach ( property in the user's utterance )
   {
      if ( the property matches an input item's slot name )
      {
         Copy the value of that property into the input item's f
         item variable.

         Set the input item's "just_filled" flag.
      }
   }
}

// Set all <initial> form item variables if any input items are

if ( any input item variable is set as a result of the user utte
    Set all <initial> form item variables to true.

// Next execute any <filled> actions triggered by this utterance

foreach ( <filled> action in document order )
{
    // Determine the input item variables the <filled> applies t

    N = the <filled>'s "namelist" attribute.

    if ( N equals "" )
    {
       if ( the <filled> is a child of an input item )
         N = the input item's form item variable name.
```

```
              else if ( the <filled> is a child of a form )
                N = the form item variable names of all the input
                    items in that form.
          }

          // Is the <filled> triggered?

          if ( any input item variable in the set N was "just_filled"
                 AND  (  the <filled> mode is "all"
                             AND all variables in N are filled
                         OR the <filled> mode is "any"
                             AND any variables in N are filled) )
                Execute the <filled> action.

           If an event is thrown during the execution of a <filledgt;,
                event handler selection starts in the scope of the <fil
           which could be an input item or the form itself.
      }
      // If no input item is filled, just continue.
  }
```

D DTD

This section defines the DTD used to formally define the XHTML+Voice 1.1 integration profile. This section is *normative*.

D.1 xhtml+voice11.dtd

The individual modules making up the DTD for profile `xhtml+voice11` along with the top-level driver file are packaged together and available with this document. Note that use of the DTD in place of the SCHEMA requires the elements and attributes specified by both [Speech Grammars] and [SSML 1.0] to be placed within their respective namespaces. The speech grammar namespace is "http://www.w3.org/2001/06/grammar" and the SSML namespace is "http://www.w3.org/2001/10/synthesis."

E Schema

This section defines the formal XML Schema used to define the XHTML+Voice 1.1 profile. This section is *normative*.

E.1 xhtml+voice11.xsd

The individual modules making up the SCHEMA for the XHTML+Voice 1.1 profile along with the top-level driver file are packaged together and available with this document.

F References

F.1 Normative References

XHTML Basic
XHTML Basic , 19 December 2000, Mark Baker, Masayasu Ishikawa, Shinichi Matsui, Peter Stark, Ted Wugofski, Toshihiko Yamakami

CSS2
Cascading Style Sheets, level 2 (CSS2) Specification, Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs, 1998. W3C Recommendation available at: http://www.w3.org/TR/REC-CSS2/.

DOM2 Events
Document Object Model (DOM) Level 2 Events Specification, Tom Pixley, 2000. W3C Recommendation available at: http://www.w3.org/TR/DOM-Level-2-Events/.

HTML 4.01
HTML 4.01 Specification, Dave Raggett, Arnaud le Hors, Ian Jacobs, 1999. W3C Recommendation available at: http://www.w3.org/TR/html4/.

RFC 2396
RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax., Tim Berners-Lee, et. al, 1998. Available at: http://www.ietf.org/rfc/rfc2396.txt.

XML Events
xml Events - An events syntax for XML, Steven Pemberton, T. V. Raman and Shane P McCarron, 2001. W3C Working Draft available at: http://www.w3.org/TR/xml-events/.

Speech Grammars
Speech Recognition Grammar Specification Version 1.0, Andrew Hunt and Scott McGlashan. W3C Candidate Recommendation, June, 2002 available at: http://www.w3.org/TR/speech-grammar/.

SSML 1.0
Speech Synthesis Markup Language Specification, Mark Walker, Dan Burnett, and Andrew Hunt. W3C Working Draft, December, 2002 available at: http://www.w3.org/TR/speech-synthesis/.

VoiceXML 2.0
Voice Extensible Markup Language (VoiceXML) , Scott McGlashan et al, W3C Candidate Recommendation available at: http://www.w3.org/tr/voicexml20/.

XHTML Modularization
Modularization of XHTML Murray Altheim, Frank Boumphrey, Sam Dooley, Shane McCarron, Sebastian Schnitzenbaumer, Ted Wugofski available at: http://www.w3.org/TR/xhtml-modularization/.

XHTML 1.1
XHTML 1.1 - Module-based XHTML Murray Altheim, Shane McCarron available at: http://www.w3.org/TR/xhtml11/.

XML 1.0
Extensible Markup Language (XML) 1.0 (Second Edition), Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, 2000. W3C Recommendation: available at: http://www.w3.org/TR/REC-xml.

XML Names
Namespaces in XML, Tim Bray, Dave Hollander, Andrew Layman, 1999. W3C Recommendation available at: http://www.w3.org/TR/REC-xml-names/.

XSchema-1
XML Schema Part 1: Structures, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, 2001. W3C Recommendation available at: http://www.w3.org/TR/xmlschema-1/.

XSchema-2
XML Schema Part 2: Datatypes, Paul V. Biron, Ashok Malhotra, 2001. W3C Recommendation available at: http://www.w3.org/TR/xmlschema-2/.

XHTML 1.0
XHTML 1.0: The Extensible HyperText Markup Language - A Reformulation of HTML 4 in XML 1.0, Steven Pemberton, et. al, 2000. W3C Recommendation available at: http://www.w3.org/TR/xhtml1/.

F.2 Informative References

ECMA 262
ECMA-262: ECMAScript Language Specification, European Computer Manufacturers' Association (ECMA), 1999. Available at ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf.

RFC 2141
RFC 2141: URN Syntax, R. Moats, 1997. Available at: http://www.ietf.org/rfc/rfc2141.txt.

XForms
XForms 1.0 , Micah Dubinko, Josef Dietl, Roland Merrick,Dave Raggett, T. V. Raman, Linda Bucsay Welsh 2001. W3C Candidate Recommendation available at: http://www.w3.org/TR/xforms/.

XSchema-0
XML Schema Part 0: Primer, David C. Fallside, 2001. W3C Recommendation available at: http://www.w3.org/TR/xmlschema-0/.

XSLT
XSL Transformations (XSLT) Version 1.0, James Clark, 1999. W3C Recommendation available at: http://www.w3.org/TR/xslt.