Migration Guide

SpeechWorks

OpenSpeech Browser 3.0

Document History

Date	Release Name
August 2003	Draft First Edition, for OpenSpeech Browser 3.0 Beta

Published by:

SpeechWorks International, Inc.

695 Atlantic Avenue

Boston, MA 02111 USA

Table of Contents

1 Introduction 1
Migrating from OSB 1.2 1
Migrating from OSB 2.0 Alpha 1
2 Platform Migration From OSB 2.0 Alpha 2
Changes to Installation Layout 2
Interface Changes 2
3 Application Migration From OSB 1.2 3
VoiceXML 2.0 spec, 10/2001 to 1/2003 change summary 4
VoiceXML 2.0 Optional Features 9
Grammar related optional features 9
Recognition related optional features 10
Miscellaneous optional features 11
4 Platform Migration From OSB 1.2 13
VXIinterpreter 13
Additions 13
Modifications 14
Deletions 16
VXIrec 17
Additions 17
Modifications 19
Deletions 19
VXIprompt 20
Modifications 21
Deletions 22
VXItel 23
Modifications 23

1 Introduction

The OpenSpeech Browser version 3.0 (OSB 3.0) product targets the following features:

- Updating the interpreter to permit execution of documents conforming to the Voice Extensible Markup Language (VoiceXML) February 2003 Candidate Recommendation. (OSB 1.2 supported the October 2001 CR.)
- Platform component API changes to:
    - Simplify results gathering
    - Insert events into VXI interpreter
    - Stop method
    - Recognition during transfer
    - Recognition during recording
    - Hotword support

Migrating from OSB 1.2

OSB 3.0 represents a comprehensive upgrade of the interpreter from the OSB 1.2 release. Due to internal overhauls, new features introduced, and incompatible changes made to the VoiceXML 2.0 Candidate Recommendation since the version supported by OSB PIK 1.2, upgrading to OSB 3.0 will require some modifications to your platform integration code. VoiceXML documents written to work with OSB PIK 1.2 **must** be modified to conform to the later version of the CR that OSB 3.0 supports. In addition, there are significant changes to the interpreter interface.

See Chapter 3, *Application Migration From OSB 1.2,* for details on migrating your OSB 1.2 VoiceXML applications to work with OSB 3.0.

See Chapter 4, *Platform Migration From OSB 1.2*, for details on migrating your OSB 1.2 platform to work with OSB 3.0.

Migrating from OSB 2.0 Alpha

Migrating from OSB 2.0 Alpha to OSB 3.0 requires some minor changes to the platform integration. No application migration is anticipated. See Chapter 2, *Platform Migration From OSB 2.0 Alpha*, for details.

2 Platform Migration From OSB 2.0 Alpha

Migrating from OSB 2.0 Alpha involves a change to the install layout and two function interface changes.

Changes to Installation Layout

In **OSB 2.0 Alpha**, the install location was incorrectly omitted from the zip file / tar file packages. In **OSB 3.0 Beta**, this has been corrected. The file paths within the zip/gz files now contain the path `SpeechWorks/OpenSpeech_Browser_PIK`, as was done for OSB 1.2.

This change to the product install location will affect your build environments and runtime environments.

Interface Changes

Changes to VXIinterpreterInterface

- The `Run()` function's signature has changed. The third argument, `sessionScript`, has changed from (`VXImap` *) to `(const VXIchar *)`. This argument must now point to a string which is formatted as valid ECMAScript.

Changes to VXIrecInterface

- The LoadGrammarOption() function has a new additional argument, called grammarAcceptance, of type VXIVector. This argument corresponds to the "accept" attribute. Possible values are: 0 means "exact"; otherwise means "approximate".

3 Application Migration From OSB 1.2

Significant changes have been made to the W3C VoiceXML draft between the release of the October 2001 draft, to which OSB 1.2 was implemented, and the February 2003 Candidate Recommendation, to which OSB 3.0 is implemented.

Most of these changes are largely semantic in nature (as opposed to syntactic). While simple syntactic changes could be dealt with through an automated conversion tool, the semantic changes are likely to impact each application differently, depending on the application's specific use of the element(s) affected.

This will require applications developed against the earlier, October 2001 draft to be evaluated against the February 2003 Candidate Recommendation. In order to guide application evaluation, we are providing the following table, which summarizes changes made to the W3C specification between the October 2001 and February 2003 Candidate Recommendations. Note that the full range of changes from VoiceXML 1.0 to VoiceXML 2.0 are summarized in Appendix J of the February 2003 Candidate Recommendation.

VoiceXML 2.0 spec, 10/2001 to 1/2003 change summary

The information in the following table is only a guide to the changes from the October 2001 draft to the February 2003 Candidate Recommendation. ***You will need to review the February 2003 Candidate Recommendation to ascertain the effect of the changes on your applications.***

VoiceXML Draft Section	**Summary**
1.2.4	* mapping of grammar results into VoiceXMLchanged * additional design principal added
1.2.5	* ***error.noresource*** thrown for unavailable audio input resource * ***error.noresource*** thrown for unavailable audio output resource
1.5.1	**<vxml>** *requires* ***xmlns*** attribute
1.5.2	* ***error.badfetch*** is thrown for non-existent root document * ***error.semantic*** is thrown for a root document referencing another root document * corrected examples with inline XML Form SRGS grammars without root rules
2.1.2.1	* ***error.unsupported.objectname*** thrown if implementation doesn't support a specific object * ***error.unsupported.format*** event is NOT thrown for unsupported object types
2.1.4	* corrected examples with inline XML Form SRGS grammars without root rules
2.1.6.2	* errors raised in *select* or *collect* phases of FIA generate an event, FIA moves directly to *process phase*
2.1.6.2.1	* ***error.unsupported.objectname*** thrown if implementation doesn't support a specific object * ***error.unsupported.format*** event is NOT thrown for unsupported object types * errors raised in select or collect phases of FIA generate an event, FIA moves directly to process phase
2.1.6.2.3	* ***error.unsupported.objectname*** thrown if implementation doesn't support a specific object * ***error.unsupported.format event*** is NOT thrown for unsupported object types * errors raised in select or collect phases of FIA generate an event, FIA moves directly to process phase
2.2	* phrase added to 'interpretation model'
2.2.1	* ***error.badfetch*** (was ***error.semantic***) is thrown for ***<menu dtmf="true">*** and it has ***<choice>***s with DTMF sequences other than "*", "#", or "0"
2.2.2	* ***<choice>*** may contain DTMF and speech grammars * ***<choice>*** may NOT contain grammar fragments
2.3	* all shadow variables are writeable/modifiable * ***application.lastresult$*** is now writeable and modifiable
2.3.1	* removed 'builtin grammars like boolean' from slot attribute
2.3.1.3	* value attribute description modified
2.3.4	* standalone query string is NOT a valid URI * standalone query strings require NO special handling in transitional URIs specified in ***<subdialog>*** and ***<goto>*** * ***<subdialog>*** *without* ***<return>*** continues until ***<exit>*** or no more forms for FIA selection

VoiceXML Draft Section	Summary
2.3.6	* ***<record>*** *variable implementation* is platform specific
	* ***<record>*** *behavior* specified for ***<audio>*** and ***<submit>*** MUST be supported on all platforms
	* time designator is a non-negative number, must be followed by ms or s
	* attributes that take time designators as their value (default is 0s) :
	<prompt timeout>,
	<transfer maxtime finalsilence>
	* properties that take time designators as their value :
	fetchtimeout,
	completetimeout,
	incompletetimeout,
	maxspeechtimeout,
	interdigittimeout,
	termtimeout,
	timeout,
	fetchaudiodelay,
	fetchaudiominimum,
	fetchtimeout
	* ***lastresult$value*** description
	* ***<record>*** diagram added
	* synchronized example/diagram added
2.3.7	* time designator is a non-negative number, must be followed by ms or s
	* attributes that take time designators as their value (default is 0s) :
	<prompt timeout>,
	<transfer maxtime finalsilence>
	* properties that take time designators as their value :
	fetchtimeout,
	completetimeout,
	incompletetimeout,
	maxspeechtimeout,
	interdigittimeout,
	termtimeout,
	timeout,
	fetchaudiodelay,
	fetchaudiominimum,
	fetchtimeout
	* use of ***session.connection.aai*** corrected
2.3.7.2	* bridged transfer w/ caller forces disconnect (DTMF or voice command) addressed
	* caller hangup during transfer or transfer attempt: **<transfer>** variable and *shadow variables NOT set*
	* bridged transfer w/ caller forces disconnect (DTMF or voice command) : **duration** *shadow variable* set to 0
	* bridged transfer terminated by DTMF input: **<transfer> utterance** *shadow variable* set to DTMF input
	* bridged transfer while normal bargein: ***bargeintype*** fixed to *hotword*, ***grammar activation*** is *modal,* **transferaudio** begins playing at point outgoing call *begins*
	* figure 10: ***bargein*** *only* applies to queued audio and played *before* outgoing call initiated
	* figure 10: bargeingtype is always set to *hotword*
	* more discussion on shadow variables
2.5	* <(element) ***scope=***xx> where the (element) contains ***<link>*** does NOT affect the copy of the **<link>**'s grammars

VoiceXML Draft Section	Summary
3	* updated to latest SRGS spec * removed unnecessary refs to JSGF * aligned weight descriptions * *semantic interpretation* section *rewritten*
3.1.1	* SRGS ***<grammar>*** is *extended* to allow *PCDATA* for *inline grammar formats* besides the XML format of SRGS
3.1.1.1	* ABNF example corrected
3.1.1.4	* define ***xml:lang*** in terms of a *language identifier* ** SRGS <grammar> is extended to allow PCDATA for inline grammar formats besides the XML format of SRGS* * use and interpretation of ***<grammar> attributes*** clarified * inline XML SRGS grammars follow SRGS spec * inline ABNF SRGS grammar attributes MUST be ignored by the platform * extermal XML and ABNF SRGS grammar attributes MUST be ignored by the platform * all other grammar types' use and interpretation is platform-dependent * ***<grammar> base*** attribute updated to *xml:base* according to latest SRGS
3.1.2	* format of DTMF grammar example corrected
3.1.4	* ***error.semantic*** thrown when no active grammars in a ***<form>*** or ***<menu>*** when input is expected
3.1.6	* section rewritten
3.1.6.1	* matching *form-level grammars* can *override* existing *values in input items* * ***<filled>*** processing for those items is described in Section 2.4 and Appendix C
4	* updated to latest SSML spec * clarified VoiceXML addtions to SSML element defs * ***bargeintype*** defaults to '*speech*' as non platform-specific
4.1	* define ***xml:lang*** in terms of a language identifier * ***time*** designator is a *non-negative number*, must be followed by ***ms*** or ***s*** * attributes that take time designators as their value (default is 0s) : ***<prompt timeout>,*** ***<transfer maxtime finalsilence>*** * properties that take time designators as their value : ***fetchtimeout,*** ***completetimeout,*** ***incompletetimeout,*** ***maxspeechtimeout,*** ***interdigittimeout,*** ***termtimeout,*** ***timeout,*** ***fetchaudiodelay,*** ***fetchaudiominimum,*** ***fetchtimeout*** **** <prompt bargein=xx bargeintype=yy>*** default values determined by properties: *bargein* and ***bargeintype***
4.1.3	* ***<audio>***: when audio file can't be played and content of element is empty => *NO* audio played,*NO* error event thrown
4.1.5	* ***<prompt bargein="false">*** results in *no buffering of input* while prompt is playing * ***<prompt bargein="false">*** will discard any previously buffered DTMF * bargein during a sequence of prompts clarified
4.1.5.1	* ***<prompt bargeintype=xx>*** applies to both DTMF and speech input

VoiceXML Draft Section	**Summary**
4.1.8	* FIA: entry to waiting state and FIA phases clarified
5.1.2	* app and doc scoping of variables in application root doc
5.1.5	* all shadow variables are writeable/modifiable * ***application.lastresult$*** is now writeable and modifiable
5.2.2	* ***event counter*** description corrected
5.2.6	* platform specific error handling when errors in subsequent documents, which are raised in the first document in a session, and for all errors raised prior to FIA entry
5.3.2	* undeclared variable assignment throws ***error.semantic*** event * ECMAScript object properties directly assignable * declaring ECMAScript object properties as results throws ***error.semantic*** event
5.3.6	* FIA performs *normal prompt queueing* after the execution of catch elements when they end with a ***<submit>*** or ***<return>*** as well as ***<goto>***
5.3.7	* ***<goto>***: errors handled in *dialog* scope for form item transition errors
6.1.1	* ***maxage*** and ***maxstale*** modelled after *max-age* and *max-stale* in HTTP1.1 * time designator is a non-negative number, must be followed by ms or s * attributes that take time designators as their value (default is 0s) : <prompt timeout>, <transfer maxtime finalsilence> * properties that take time designators as their value : fetchtimeout, completetimeout, incompletetimeout, maxspeechtimeout, interdigittimeout, termtimeout, timeout, fetchaudiodelay, fetchaudiominimum, fetchtimeout * '*' removed from each attribtute's descrption (e.g. '*maxage' is now 'maxage')
6.2	* ***<metadata>*** is recommended over ***<meta>***
6.2.1	* removed *recommended data* from ***<meta>*** * **<meta>** types: 1st expressed by attributes ***name, content***, 2nd expressed by ***http-equiv, content***
6.2.2	* added *recommended data* to ***<metadata>*** using RDF schema and Dublin Core properties

VoiceXML Draft Section	Summary
6.3	* time designator is a non-negative number, must be followed by ms or s * attributes that take time designators as their value (default is 0s) : ***<prompt timeout>,*** ***<transfer maxtime finalsilence>*** * properties that take time designators as their value : ***fetchtimeout,*** ***completetimeout,*** ***incompletetimeout,*** ***maxspeechtimeout,*** ***interdigittimeout,*** ***termtimeout,*** ***timeout,*** ***fetchaudiodelay,*** ***fetchaudiominimum,*** ***fetchtimeout***
6.3.1	* platform specific properties reduce application portability
6.3.6	* ***confidencelevel*** property in example corrected * platform-specific ***universal command grammars*** are *OPTIONAL*
AppendixA	* define ***xml:lang*** in terms of a *language identifier*
AppendixB	* ***all*** removed as default for mode attribute of ***<filled>*** * XML prolog removed from DTD * added option <vxml ***xmlns:xsi***> attribute to DTD
AppendixC	* clarification that events are generated at more than one point in FIA processing * clarification in FIA: matching ***<link>*** grammars in current ***<form>*** or ***<menu>*** * clarification in FIA: matching ***<menu>***'s ***<choice>*** grammars outside current * collection of *active grammars* does NOT include grammars from ***<subdialog>*** call chain * clarification in FIA initialization: ***<script>*** elements an ***<form><item>***s * ***application.lastresult$*** happens after *every successful* recognition * FIA performs *normal prompt queueing* after the execution of catch elements when they end with a ***<submit>*** or ***<return>*** as well as ***<goto>*** * ***<form>*** allows multiple ***<initial>*** elements * clarification on selection of ***<initial>*** elements for execution * for field-level scoped grammars, utterance results: if top-level property matches slot n OR slot name is dot-separated path matching the result's subproperty, copy property's value into result * defintion of *utterance* and it's semantic result updated * *process phase* section of *semantic mapping* rewritten
AppendixF	* Conforming example description no longer *minimal* * VoiceXML Forum will not determine whether platform is a conforming VoiceXML Processor
AppendixJ	* reference to ***dest*** shadow variable in **<record>** removed * ***connection*** session variable description corrected
AppendixM	
AppendixN	* IETF link for VoiceXML Media type registration

VoiceXML Draft Section	**Summary**
AppendixO	* ***all*** removed as default for mode attribute of ***<filled>*** * moved ***audio*** and ***say-as extensions*** from ***vxml-synthesis-extension*** to ***vxml-synthesis-restriction*** * schema errors corrected * ***field.name*** renamed to ***variable.name*** * ***field.names*** renamed to ***vaiable.names*** * EventNames as NMTOKENS [catch event="" is illegal] * added vxml *builtin types* to ***say-as***
AppendixP	* speech or DTMF ***<grammar>*** in a ***<field>*** w/*builtin* type do NOT override builtin grammars (they are in addition to builtin grammars) * digit grammars can be parameterized * boolean grammars can be parameterized * *builtins* description modified * *builtins* not as strongly discouraged

VoiceXML 2.0 Optional Features

Several features are classified as optional by the Candidate Recommendation. Unless otherwise noted, OSB 3.0 will not support these features.

As of this writing (8/6/2003), the list of these features to be implemented in OSB 3.0 is not yet finalized.

Grammar related optional features

SRGS ABNF grammars VXML 2.0 section 3.1.1.1

The Speech Recognition Grammar Specification defines an XML-based grammar format which is required for all VoiceXML platforms. SRGS also defines a text-based ABNF form. The ABNF representation offers less expressive power than the XML-based format, but may be more familiar to grammar authors with a speech science background.

Non-SRGS grammars VXML 2.0 section 3.1.1.4

A platform may allow grammars in a format other than SRGS to be inlined in VoiceXML documents. These may not use XML tags unless enclosed in a CDATA declaration.

Approximate grammar matching VXML 2.0 section 2.2.5

Some automatically generated grammars associated with <menu>, <choice>, and <option> may loosely match the actual grammar text. This is controlled by an attribute named 'accept' with values 'exact' or 'approximate'. In the 'approximate' case, this transfers control of the actual grammar from the document author to the platform.

Standard grammar library VXML 2.0 section Appendix P

Platforms may provide some 'built-in' grammars for common tasks such as collection yes/no responses, dates, digit strings, currency amounts, etc. These grammars may be very useful for rapid prototyping, but are not standardized and reduce portability.

Universal platform-specific grammars VXML 2.0 section 6.3.6

Platforms may define standard grammars which generate VoiceXML events such as 'help', 'cancel', and 'exit'. If available on a platform and in the current language, application writers would be able to control which grammars are active by using the 'universal' property. Application writers in VoiceXML are unable to control the content of these grammars.

Recognition related optional features

Recognition during <record> VXML 2.0 section 2.3.6

Commonly, recordings are terminated by an interval of silence, by a DTMF keypress, or by a maximum duration being reached. Some platforms may allow the user to terminate the recording by matching a speech or DTMF grammar. For speech, this requires that a recognition engine capable of hotword bargein be active for the duration of the <record>ing.

Recognition during <transfer> VXML 2.0 section 2.3.7

Commonly, a bridge transfer is terminated by a far end disconnect or by the maximum call duration being reached. Some platforms may allow the user to terminate the tranfer by matching a speech or DTMF grammar. For speech, this requires that a recognition engine capable of hotword bargein be active for the duration of the <transfer>ing.

Speech detection barge-in VXML 2.0 section 4.1.5.1

Barge-in specifies whether a user can interrupt a playing prompt by speaking or pressing a DTMF key. The prompt will stop as soon as user input is detected. While not strictly required, this is very useful for getting expert users quickly through an application.

Hotword detection barge-in VXML 2.0 section 4.1.5.1

Barge-in specifies whether a user can interrupt a playing prompt by speaking or pressing a DTMF key. The prompt will only be stopped if the user's input matched an active grammar with good confidence.

Multiple Recognition Results VXML 2.0 section 6.3.6

A successful recognition must return one result and may return up to 'maxnbest'. If available, this information may be accessed by the application writers via the 'application.lastresult$' array.

Progress-based Recognition Timeout VXML 2.0 section 6.3.2

Seperate timeout values may be specified depending on whether the user's speech constitutes a valid matches against active grammars (i.e. 'completetimeout') or not (i.e. 'incompletetimeout'). While collecting a string of five digits, for instance, the 'incompletetimeout' might be used until the user has said the fifth digit whereupon the 'completetimeout' would be used. Platforms unable to support both timeouts must use the 'incompletetimeout' for both cases.

Miscellaneous optional features

Platform-specific extensions via <object> VXML 2.0 section 2.3.5

This mechanism in VoiceXML allows for platform-specific extensions. <object>s might provide database access, perform user verification, invoke telephony features, or any number of other functions.

Application requested prefetching VXML 2.0 section 6.1.1

VoiceXML allows 'fetchhint's which the platform may use to load links on document load (i.e. 'prefetch') or when the content is required during execution (i.e. 'safe'). Any specification of 'prefetch' is merely a hint which a platform may ignore.

Support for Western European Character Encodings

VXML 2.0 section Appendix F1

All VoiceXML browsers must be capable of reading UTF-8 and UTF-16 contents. Support is optional for other character sets such as ASCII and Latin-1 (ISO8859-1).

Support for Asian Character Encodings

VXML 2.0 section Appendix F1

All VoiceXML browsers must be capable of reading UTF-8 and UTF-16 contents. Support is optional for other character sets such as Big5, Shift-JIS, and EUC-JP.

Secure HTTP

VXML 2.0 section 6.1.4

Secure Hypertext Transfer Protocol (HTTPS) support is recommended but not required.

4 Platform Migration From OSB 1.2

Changes have been made to the following interfaces:

- VXIinterpreter
- VXIrec
- VXIprompt
- VXItel

VXIinterpreter

Additions

New Error codes

```
/** Run call aborted */
VXIinterp_RESULT_STOPPED              =    3,
/** Document has syntax errors     */
VXIinterp_RESULT_SYNTAX_ERROR         =   54,
/** Uncaught fatal VoiceXML event  */
VXIinterp_RESULT_UNCAUGHT_FATAL_EVENT =   55,
/** ECMAScript syntax error        */
VXIinterp_RESULT_SCRIPT_SYNTAX_ERROR  =   56,
/** ECMAScript exception throw     */
VXIinterp_RESULT_SCRIPT_EXCEPTION     =   57,
```

New Functions

RequestStop()

Enables the platform to stop a running interpreter.

```
/**
 * In the interpreter is running and doStop == TRUE, this will cause the
 * in progress Run to return as soon as possible with
 * VXIinterp_RESULT_STOPPED.  If doStop == FALSE, this clears any pending
 * stop request.
 *
 * NOTE: if the interpreter encounters an error before noticing the
 * or while servicing the request, the actual return value from Run may
 * not be something other than VXIinterp_RESULT_STOPPED.
 *
 * @return         VXIinterp_RESULT_SUCCESS on success
 *                 VXIinterp_RESULT_INVALID_ARGUMENT
 */
VXIinterpreterResult (*RequestStop)(struct VXIinterpreterInterface  *pThis,
                                    VXIbool doStop);
```

InsertEvent()

Enables the platform to insert a run-time event into a running interpreter. The event is handled at the same level as any VoiceXML event that might be generated during document processing. Note that this facility is only meant to allow a single event to be reliably inserted. Subsequent events could override the inserted event, or vice-versa.

```
/**
 * Trigger an event
 *
 * @param event    [IN] VoiceXML event to generate during Run.
 * @param message  [IN] Corresponding message string; may be NULL.
 *
 * @return         VXIinterp_RESULT_SUCCESS on success
 *                 VXIinterp_RESULT_INVALID_ARGUMENT
 */
VXIinterpreterResult (*InsertEvent)(struct VXIinterpreterInterface *pThis,
                                    const VXIchar                  *event,
                                    const VXIchar                  *message);
```

Modifications

Modified Error codes

```
/** Unable to open or read from URI */
VXIinterp_RESULT_FETCH_ERROR          =   52,
```

Modified Functions

VXIinterpreter::Run()

More explicit return codes have been implemented.

```
/**
   * Run a VoiceXML document and optionally return the result
   *
   * @param name    [IN] Name of the VoiceXML document to fetch and
   *                  execute, may be a URL or a platform dependant path.
   *                  See the Open() method in VXIinet.h for details
   *                  about supported names, however for URLs this
   *                  must always be an absolute URL and any query arguments
   *                  must be embedded.
   * @param sessionArgs [IN] Any arguments to be passed to the VXI to populate
   *                  the session scope in ECMAScript.
   * @param result  [OUT] (Optional, pass NULL if not desired.) Return
   *                  value for the VoiceXML document (from <exit/>), this
   *                  is allocated on success and when there is an
   *                  exit value (a NULL pointer is returned otherwise),
   *                  the caller is responsible for destroying the returned
   *                  value by calling VXIValueDestroy().
   *
   * @return         [From normal operation]
   *                 VXIinterp_RESULT_SUCCESS on success
   *                 VXIinterp_RESULT_FAILURE if normal error occured
   *                 VXIinterp_RESULT_STOPPED if aborted by Stop
   *                 [During initialization from defaults]
   *                 VXIinterp_RESULT_FETCH_TIMEOUT
   *                 VXIinterp_RESULT_FETCH_ERROR
   *                 VXIinterp_RESULT_INVALID_DOCUMENT
   *                 [Serious errors]
   *                 VXIinterp_RESULT_FATAL_ERROR
   *                 VXIinterp_RESULT_OUT_OF_MEMORY
   *                 VXIinterp_RESULT_PLATFORM_ERROR
   *                 VXIinterp_RESULT_INVALID_ARGUMENT
   */
  VXIinterpreterResult (*Run)(struct VXIinterpreterInterface  *pThis,
                              const VXIchar                   *name,
                              const VXIMap                    *sessionArgs,
                              VXIValue                       **result);
```

VXIinterpreter::SetProperties

More explicit return codes have been implemented

```
/**
   * Specify runtime properties for the VoiceXML interpreter.
   *
   * @param props   [IN] Map containing a list of properties.  Currently there
   *                  are two of interest:
   *                  * VXI_BEEP_AUDIO           URI for the beep audio
   *                  * VXI_PLATFORM_DEFAULTS    URI for the platform defaults
   *
   * @return         VXIinterp_RESULT_SUCCESS on success
   *                 VXIinterp_RESULT_INVALID_PROP_NAME
   *                 VXIinterp_RESULT_INVALID_PROP_VALUE
   *                 VXIinterp_RESULT_INVALID_ARGUMENT
   */
  VXIinterpreterResult (*SetProperties)(struct VXIinterpreterInterface *pThis,
                                      const VXIMap                   *props);
```

VXIinterpreter::Validate

More explicit return codes have been implemented

```
/**
 * Load and parse an VXML document.  This tests the validity.
 *
 * @param name    [IN] Name of the VoiceXML document to fetch and
 *                  execute, may be a URL or a platform dependant path.
 *                  See the Open( ) method in VXIinet.h for details
 *                  about supported names, however for URLs this
 *                  must always be an absolute URL and any query arguments
 *                  must be embedded.
 *
 * @return        VXIinterp_RESULT_SUCCESS if document exists and is valid
 *                VXIinterp_RESULT_FAILURE if document is invalid VXML
 *                VXIinterp_RESULT_FETCH_ERROR if document retrieval failed
 *                VXIinterp_RESULT_FETCH_TIMEOUT
 *                VXIinterp_RESULT_INVALID_ARGUMENT
 *                VXIinterp_RESULT_FATAL_ERROR
 *                VXIinterp_RESULT_OUT_OF_MEMORY
 */
VXIinterpreterResult (*Validate)(struct VXIinterpreterInterface  *pThis,
                                 const VXIchar *name);
```

VXIinterpreterCreateResource

Now also returns invalid argument

```
/**
 * @name VXIinterpreterCreateResource
 * @memo  Create an interface to the VoiceXML interpreter.
 * @doc Create a VXI interface given an interface structure that
 * contains all the resources required for the VXI.
 *
 * @param resource [IN] A pointer to a structure containing all the
 *                      interfaces requires by the VXI
 * @param pThis    [IN] A pointer to the VXI interface that is to be
 *                      allocated.  The pointer will be set if this call
 *                      is successful.
 *
 * @return     VXIinterp_RESULT_SUCCESS if interface is available for use
 *             VXIinterp_RESULT_OUT_OF_MEMORY if low memory is suspected
 *             VXIinterp_RESULT_INVALID_ARGUMENT
 */
VXI_INTERPRETER VXIinterpreterResult
VXIinterpreterCreateResource(VXIresources *resource,
                             VXIinterpreterInterface ** pThis);
```

Deletions

Deleted Keys identifying properties for SetProperties.

```
#define VXI_MAX_LOOP_ITERATIONS L"vxi.property.maxLoopIterations"/*VXIInteger*/
#define VXI_MAX_DOCUMENTS       L"vxi.property.maxDocuments"     /*VXIInteger*/
#define VXI_MAX_EXE_STACK_DEPTH L"vxi.property.maxExeStackDepth" /*VXIInteger*/
```

Deleted Error codes

```
/** I/O error                        */
VXIinterp_RESULT_IO_ERROR              =   -8,
/** System error, out of service     */
VXIinterp_RESULT_SYSTEM_ERROR          =   -6,
/** Return buffer too small          */
VXIinterp_RESULT_BUFFER_TOO_SMALL      =   -4,
/** Non-fatal non-specific error     */
VXIinterp_RESULT_NON_FATAL_ERROR       =    2,
/** Document not found               */
VXIinterp_RESULT_NOT_FOUND             =   50,
```

VXIrec

The major change to VXIrec is that recognition results are now in XML format.

Additions

Recognition during transfer (Hotword)

VXItel dependency

Now depends on VXItel for record while transfer.

```
#include "VXItel.h"
```

Transfer result structure

```
/**
 * Record results structure as returned by Record( )
 */
typedef struct VXIrecTransferResult {
  VXIContent*  xmlresult;

  /** Duration of the transfer in milliseconds */
  VXIunsigned duration;

  /** Result code for the transfer request */
  VXItelTransferStatus status;

  /*
  ** Call to release the resources stored with this result structure
  */
  void           (*Destroy)(struct VXIrecTransferResult **result);
} VXIrecTransferResult;
```

VXIrec::HotwordTransfer()

Enables transfer to be terminated using recognition of ‘hotword’

```
  /*
   * @name HotwordTransfer
   * @memo HotwordTransfer provides for recognition terminated transfer
   * @doc
   * Called by the interpreter to perform a bridged transfer.
   *
   * If a platform returns VXIrec_RESULT_UNSUPPORTED, indicating that hotword
   * recognition is not supported during this transfer, the interpreter will
   * call VXItelInterface::TransferBridge.
   *
   * @param properties     [IN] termination character, length, timeouts...
   * @param transferDest   [in] identifier of transfer location (e.g. a
   *                       phone number to dial or a SIP URL)
   * @param data           [IN] the data to be sent to the transfer target
   * @param transferResult [OUT] Newly allocated result structure containing
   *                       the result of the transfer request; see structure
   *                       definition above
   *
   * @return VXIrec_RESULT_SUCCESS on success
   */
  VXIrecResult (*HotwordTransfer)(struct VXIrecInterface * pThis,
                                  struct VXItelInterface * tel,
                                  const VXIMap *properties,
                                  const VXIchar* transferDest,
                                  const VXIMap *data,
                                      VXIrecTransferResult  ** transferResult);
```

XMLRESULT Mimetype added:

```
#define VXIREC_MIMETYPE_XMLRESULT  L"application/x-vnd.speechworks.osb2"
```

Mime options added:

```
#define REC_MIME_OPTION         L"text/x-grammar-option"
#define REC_MIME_OPTION_DTMF    L"text/x-grammar-option-dtmf"
```

VXIrec::GetMatchedGrammar()

```
  /*
   * @name GetMatchedGrammar
   * @memo GetMatchedGrammar returns grammar for ID in recognition result
   * @doc
   * An XML result returned by a successful recognition contains a string
   * identifying the matched grammar.  This function is use to map that
   * string back to the corresponding VXIrecGrammar.
   *
   * @param grammarID     [IN] String identifier from the XML result.
   * @param gram          [OUT] Corresponding VXIrecGrammar.
   *
   * @return VXIrec_RESULT_SUCCESS on success
   */
  VXIrecResult (*GetMatchedGrammar)(struct VXIrecInterface * pThis,
                                    const VXIchar *grammarID,
                                    VXIrecGrammar **gram);
```

VXIrec::LoadGrammarOption()

```
  /*
   * @name LoadGrammarOption
   * @memo LoadGrammarOption may build a grammar for a VXML <option> element.
   * @doc
   * VoiceXML 2.0 requires support for the SRGS XML format but provides no
   * standard for building semantic results inside the grammar.  The <option>
   * element is unique in that it requires this mapping.  The interpreter will
   * invoke this function to build each option grammar.  For each <field>,
   * LoadGrammarOption may be called once for speech grammars and once for
   * DTMF grammars.
   *
   * @param properties  [IN] Set of properties as per LoadGrammarString
   * @param gramChoices [IN] The utterance value for this choice
   * @param gramValues  [IN] The corresponding semantic meaning
   * @param isDTMF      [IN] Is the utterance a DTMF choice?
   * @param gram        [OUT] Handle to the new grammar; possibly NULL.
   *
   * @return VXIrec_RESULT_SUCCESS on success
   */
  VXIrecResult (*LoadGrammarOption)(struct VXIrecInterface  *pThis,
                                    const VXIMap            *properties,
                                    const VXIVector         *gramChoices,
                                    const VXIVector         *gramValues,
                                    const VXIbool            isDTMF,
                                    VXIrecGrammar          **gram);
```

Modifications

Simplified recognition results

Now uses xml results rather than key value lookup. Also have done away with embedded status and input mode fields.

The new VXIrecRecognitionResult is :

```
typedef struct VXIrecRecognitionResult {
  /*
   * Waveform of the caller's utterance. Returned a VXIContent which
   * stores the length and the mime type in one unit.
   */
  VXIContent*      waveform;

  /*
   * The semantic interpretation of the recognition expressed using the
   * Natural Language Semantic Markup Language (NLSML).  The mimetype
   * of the VXIContent may allow multiple formats to be supported.
   */
  VXIContent*     xmlresult;

  /*
   * Call to release the resources stored with this result structure
   */
  void            (*Destroy)(struct VXIrecRecognitionResult **result);
} VXIrecRecognitionResult;
```

Modified record result

The record result structure no longer has as status field, and has added and xmlresult field.

```
/**
 * Record results structure as returned by Record( )
 */
typedef struct VXIrecRecordResult {
  /** The bytes of the recording */
  VXIContent * waveform;

  VXIContent*  xmlresult;

  /** Duration of the recording in milliseconds */
  VXIunsigned duration;

  /** DTMF character which terminated the recording, or 0 if none. */
  VXIbyte termchar;

  VXIbool maxtime;
  /** True if the recording was terminated because the maximum time limit was
      reached **/

  /*
  ** Call to release the resources stored with this result structure
  */
  void            (*Destroy)(struct VXIrecRecordResult **result);
} VXIrecRecordResult;
```

Deletions

NLSML mimetype deleted;

```
#define VXIREC_MIMETYPE_NLSML   L"application/grammar+nlsml"
```

Key/Value for recognition result deleted

Replaced by xml results mechanism (see Additions above)

```
/**
 * Keys identifying properties in VXIMap used to return recognition
 * result information for each n-best entry, see
 * VXIrecRecognitionResult below for details
 */
#define REC_KEYS     L"vxi.rec.keys"       /* VXIVector of VXIStrings   */
#define REC_VALUES   L"vxi.rec.values"     /* VXIVector of VXIStrings   */
#define REC_CONF     L"vxi.rec.confidence" /* VXIVector of VXIFloats    */
#define REC_RAW      L"vxi.rec.raw"        /* VXIVector of VXIStrings   */
#define REC_GRAMMAR  L"vxi.rec.grammar"    /* VXIPtr to a VXIrecGrammar */
```

Status codes deleted

```
/**
 * Status codes for recognition results
 */
typedef enum VXIrecStatus {
  REC_STATUS_SUCCESS      = 0,  /* Recognition returned a hypothesis      */
  REC_STATUS_FAILURE      = 1,  /* Speech detected, no likely hypothesis  */
  REC_STATUS_TIMEOUT      = 2,  /* No speech was detected                 */
  REC_STATUS_DISCONNECT   = 3,  /* Caller has disconnected; no hypothesis */
  REC_STATUS_ERROR        = 4,  /* An error aborted recognition           */
} VXIrecStatus;
```

Errors deleted

```
  /* Generic HW source error         */
  VXIrec_RESULT_HW_ERROR            =   55,
```

VXIprompt

This interface has shifted from the notion of prompts being a series of audio segments to prompts being a series of SSML documents. This allows platforms to more easily leverage SSML enabled Text to Speech engines, like Speechify, as well as allows a uniform approach to prompt engine development.

Prefetching, caching, and audio streaming continue to be done by this interface to optimize CPU and network overhead.

VoiceXML allows audio collected during the <record> element to be played in the middle of SSML documents. These recordings are passed by the VXI to the VXIprompt implementation inside the properties VXIMap. The PROMPT_AUDIO_REFS points to a second VXIMap containing pairs of identifiers (VXIStrings) and their associated recording (VXIContent).

Each identifier is preceded by PROMPT_AUDIO_REFS_SCHEME.

Within the SSML document, the audio recording is replaced by a mark element whose name attribute is an identifier inside the PROMPT_AUDIO_REFS map.

Modifications

VXIprompt::PlayFiller()

Clarification in the header comment that it 'Queues and possibly starts' special play of a filler segment, rather than simply 'Starts' the special play.

The *src* param no longer supports the PROMPTS_AUDIO_REF_SCHEME for playing in memory binary audio.

The *text* param format for TTS may be W3C SSML (type set to VXI_MIME_SSML) or simple wchar_t text (type set to VXI_MIME_UNICODE_TEXT). The implementation may also support other formats.

VXIprompt::PreFetch()

Criticality for calling Prefetch() prior to Queue() has been removed.

The *src* param no longer supports the PROMPTS_AUDIO_REF_SCHEME for playing in memory binary audio.

The *text* param format for TTS may be W3C SSML (type set to VXI_MIME_SSML) or simple wchar_t text (type set to VXI_MIME_UNICODE_TEXT). The implementation may also support other formats.

VXIprompt::Queue()

The *src* param has been replace by the *content* param:

```
/**
 * @name Queue
 * @memo Queue a segment for playing, blocking
 * @doc
 * The segment does not start playing until the Play() method is
 * called. This call blocks until the prompt's data is retrieved
 * (for streaming plays until the data stream starts arriving) so
 * that the caller can be assured the segment is available for
 * playback (important for supporting play with fallback).
 *
 * @param type       [IN] Type of segment, either a MIME content type,
 *                   a sayas class name, or NULL to automatically detect
 *                   a MIME content type (only valid when src is
 *                   non-NULL). The supported MIME content types and
 *                   sayas class names are implementation dependant.
 * @param content    [IN] URI or platform dependant path to the content or
 *                   NULL when specifying in-memory text.
 * @param text       [IN] Text (possibly with markup) to play via TTS
 *                   or sayas classes, pass NULL when src is non-NULL.
 *                   The format of text for sayas class playback is
 *                   determined by each class implementation. The
 *                   format of text for TTS playback may be W3C
 *                   SSML (type set to VXI_MIME_SSML) or simple wchar_t
 *                   text (type set to VXI_MIME_UNICODE_TEXT).  The
 *                   implementation may also support other formats.
 * @param properties [IN] Properties to control the fetch, queue, and
 *                   play, as specified above. May be NULL.
 *
 * @return VXIprompt_RESULT_SUCCESS on success
 */
VXIpromptResult (*Queue)(struct VXIpromptInterface *pThis,
                         const VXIchar             *type,
                         const VXIchar             *content,
                         const VXIchar            *text,
                         const VXIMap             *properties);
```

Deletions

Property Keys Deleted

```
#define PROMPT_LANGUAGE          L"vxi.prompt.language"        /* VXIString  */
#define PROMPT_RECORDING_SOURCE  L"vxi.prompt.recordingSource" /* VXIString  */
#define PROMPT_TTS_VOICE_NAME    L"vxi.prompt.tts.voiceName"   /* VXIString  */

/* Property defaults */
#define PROMPT_AUDIO_REFS_DEFAULT             NULL /* No default */
#define PROMPT_LANGUAGE_DEFAULT               L"en-US" /* US English */
#define PROMPT_RECORDING_SOURCE_DEFAULT       L"" /* use platform default */
#define PROMPT_TTS_VOICE_NAME_DEFAULT         L"" /* use TTS engine default */
```

VXItel

Modifications

Result codes renamed for consistency

In general, result codes have been clean up ala:

```
* @return VXItel_SUCCESS if operation succeeds,
*         VXItel_INVALID_ARGUMENT if pThis is NULL,
*         VXItel_HW_ERROR if an hardware error occurs,
*         VXItel_NOT_INITIALIZED if VXItel was not properly initialized.
*         VXItel_UNEXPECTED_RESULT if any other error occurs.
```

VXItel::TransferBridge()

TransferBridge is called only if VXIrecInterface::HotwordTransfer returns VXIrec_RESULT_UNSUPPORTED.