Migration Guide

SpeechWorks® solutions from ScanSoft®

OpenSpeech Recognizer 2.0

Document History

Date	Release Name
April 2004	First Edition, Update 3 – for OpenSpeech Recognizer 2.0.6
December 2003	First Edition, Update 2 – for OpenSpeech Recognizer 2.0.3
November 2003	First Edition, Update – for OpenSpeech Recognizer 2.0.2
July 2003	First Edition – for OpenSpeech Recognizer 2.0

Notice

Published by:

ScanSoft, Inc.
Worldwide Headquarters
9 Centennial Drive
Peabody, MA 01960
United States

Table of Contents

Migrating systems from OSR 1.*n* to 2.0

Overview 1
New and changed information in this guide 2
New features and technical details 2
Platform tasks versus Application Development Tasks 2
  Required migration tasks 2
  Required in some instances 3
  Recommended migration tasks 3

1. Required migration tasks 5

Remove previous OSR versions 5
Install at least one language 5
Recompile large grammars 5

2. Required in some instances 6

Move configuration variables to SpeechWorks.cfg 6
Recompile binary grammars 6
Change encoding of string grammars 7
Update handling of error codes 8
Update event log scripts 9
Update integrations of audio write functions 9
  Changes required for linear audio 9
  Changes recommended for lost or suppressed samples 10
Update grammar assignments of all URI variables 10
Update grammar preload configuration 11
Convert romanized grammars and dictionaries 11
Update EOS handling 12

3. Recommended migration tasks 13

Test your applications 13
Recalibrate your use of confidence scores 13
Use the new SWIepRead( ) function 14
Use SWIrecGetXMLResult( ) to get recognition results 15
Update use of SWIrecGrammarLoad( ) 15
Move configuration changes to user-defined files 16
Implement the confidencelevel parameter 16

Update all dictionaries to new format 17
Load user dictionaries with <lexicon> instead of <meta>....................18
Set the default language 19
Add language IDs to old dictionaries 19
Test grammar loading and update grammars as needed....................20
Update SWIrecResourceAllocate( ) and SWIrecResourceFree( ) 21
Change swirec_max_speech_duration to maxspeechtimeout....................22
Use new location for the SpeechWorks.cfg file 22

4. Miscellaneous migration issues 23

New format for log files 23
Waveform logging enabled by default 23
New RIFF format for waveform output....................24
New format for parseTool output....................24
New acoustic adaptation (LEARN)....................24
SWIrecRecognizerCompute( ) – status code change....................25
Removed configuration parameters....................25
Activating duplicate grammars....................26
Pronunciation dictionary precedence 26
Merged OSR client-server architecture....................26
Using OSR 1.*n* acoustic models with OSR 2.0 27

Migrating systems from OSR 1.*n* to 2.0

This document describes topics for platform and application developers who migrate existing OSR 1.*n* system to the 2.0 release. The topics are described from several perspectives:

- Required tasks – issues that *must be addressed*. See *page 5*.
- Possibly required tasks – issues that *might need to be addressed*. See *page 6*.
- Recommended tasks – issues for review and consideration. See *page 13*.
- General issues – issues that typically require no action. See *page 23*.

In addition, the index of this guide groups issues by category. For example, "Audio issues," "Configuration issues," "Dictionary issues," "Installation issues," and so on.

Overview

OSR 2.0 adds new features and simplifies existing features without requiring code changes to OSR 1.*n* systems. However, there are required and recommended migration tasks.

In general, coding is only needed when taking advantage of new features such as magic word endpointing and Unicode support.

New and changed information in this guide

This edition has minor copyright and titlepage changes to reflect the acquisition of SpeechWorks International, Inc., by ScanSoft, Inc.

New features and technical details

Because this guide is focused on migration issues, it limits the discussion of new features to the migration tasks to be performed. For an explicit list of new features, see the release notes or consult the OSR 2.0 marketing literature.

The OSR product documentation contains hyperlinks to every topic discussed in this migration guide. See the “New and changed information” in the preface of each book:

- *OSR Client/Server Operator’s Guide*
- *OSR Developer’s Guide*
- *OSR Licensing Handbook*
- *OSR Platform Integration Manual*
- *OSR Reference Manual*
- Language supplements for the supported languages

Platform tasks versus Application Development Tasks

Each migration task described in this guide indicates whether it should be performed by a platform developer, a system operator, or an application developer. To use the guide, review each task individually and determine whether it is appropriate for you. The following tables describe the tasks by audience type:

Required migration tasks

Tasks for platform developers (or system operators):

- “Remove previous OSR versions” on page 5
- “Install at least one language” on page 5
- “Recompile large grammars” on page 5

Tasks for application developers:

- ☐ “Install at least one language” on page 5
- ☐ “Recompile large grammars” on page 5

Required in some instances

Tasks for platform developers (or system operators):

- ☐ “Move configuration variables to SpeechWorks.cfg” on page 6
- ☐ “Recompile binary grammars” on page 6
- ☐ “Change encoding of string grammars” on page 7
- ☐ “Update handling of error codes” on page 8
- ☐ “Update event log scripts” on page 9
- ☐ “Update integrations of audio write functions” on page 9
- ☐ “Update EOS handling” on page 12

Tasks for application developers:

- ☐ “Recompile binary grammars” on page 6
- ☐ “Update event log scripts” on page 9
- ☐ “Update grammar assignments of all URI variables” on page 10
- ☐ “Update grammar preload configuration” on page 11
- ☐ “Convert romanized grammars and dictionaries” on page 11

Recommended migration tasks

Tasks for platform developers (or system operators):

- ☐ “Use the new SWIepRead( ) function” on page 14
- ☐ “Use SWIrecGetXMLResult( ) to get recognition results” on page 15
- ☐ “Update use of SWIrecGrammarLoad( )” on page 15
- ☐ “Move configuration changes to user-defined files” on page 16
- ☐ “Use new location for the SpeechWorks.cfg file” on page 22
- ☐ “Implement the confidencelevel parameter” on page 16
- ☐ “Set the default language” on page 19
- ☐ “Update SWIrecResourceAllocate( ) and SWIrecResourceFree( )” on page 21
- ☐ “Use new location for the SpeechWorks.cfg file” on page 22

Tasks for application developers:

- ☐ “Test your applications” on page 13
- ☐ “Recalibrate your use of confidence scores” on page 13
- ☐ “Move configuration changes to user-defined files” on page 16
- ☐ “Update all dictionaries to new format” on page 17
- ☐ “Load user dictionaries with <lexicon> instead of <meta>” on page 18

- ☐ "Add language IDs to old dictionaries" on page 19
- ☐ "Test grammar loading and update grammars as needed" on page 20
- ☐ "Change swirec_max_speech_duration to maxspeechtimeout" on page 22

CHAPTER 2

Required migration tasks

Remove previous OSR versions

OSR 2.0 cannot be installed on a system where a previous OSR 1.*n* is already installed. The old version of OSR must be un-installed first.

Install at least one language

In OSR 2.0, languages are installed separately from the recognizer itself. (Previously, US English was installed by default.) The purpose of the change is to reduce storage space and installation time for sites that do not require en-US.

To have a functioning system, you must install the core OSR 2.0 software plus at least one language.

Recompile large grammars

Although OSR 2.0 accepts binary grammars that were compiled under OSR 1.*n*, you must recompile large or complex grammars (e.g. stocks) to ensure fastest performance and highest accuracy. See the additional explanation in “Recompile binary grammars” on page 6 in the “Required in some instances” chapter.

CHAPTER 3

Required in some instances

Many of the following tasks are required for migration depending on which OSR 1.*n* features are used on your system.

Move configuration variables to SpeechWorks.cfg

For OSR systems running on Windows, the registry is no longer used for event log and TRC keys. Instead, the SpeechWorks.cfg is used (this configuration file was not previously used on Windows).

If you previously used registry settings,[1] you must move them to SpeechWorks.cfg.

For an overview and details (such as file locations), see the configuration chapter in the *OSR Reference Manual*.

Recompile binary grammars

ScanSoft strongly recommends that you recompile all 1.*n* binary grammars with the 2.0 compiler to ensure fastest performance and highest accuracy. Recompilation is required for large or complex grammars (e.g. stocks).

1. In OSR 1.*n*, the following registry key was used: HKEY_LOCAL_MACHINE\SOFTWARE\SpeechWorks International\ OpenSpeech Recognizer\<version>\<variable>.

When you load or activate binary grammar files that were compiled with an OSR 1.*n* compiler, OSR 2.0 performs some internal re-compilation on those grammars. The extra CPU cycles needed for the compilation can affect system performance during initialization or at runtime (depending when the binary grammars are loaded). The performance implications are amplified for large and complex grammars (e.g. stocks).

Precompiled OSR 1.*n* grammars might get poor recognition results depending on the setting of various “compile-time” parameters (in the Baseline.xml configuration file) when the grammars were compiled. For example, a setting of swirec_wtw=250 (common in OSR 1.*n*) is not optimal for recognition in OSR 2.0 and could increase errors by more than 30%. If you recompile the grammars with default OSR 2.0 settings, the problem disappears. (For a definition and list of compile-time parameters, see the parameter chapter of the *OSR Reference Manual*.)

Change encoding of string grammars

If you previously passed “string” grammars to the SWIrecGrammarLoad( ) function in OSR 1.*n*, you must ensure that the grammar encoding is compatible with the function. If a grammar encoding is not compatible, the grammar fails to load and the function returns SWIrec_ERROR_INVALID_DATA.

Previously, OSR 1.*n* did not strictly enforce grammar encoding because the software did not yet handle Unicode. You could encode string grammars in narrow-character formats such as UTF-8 or ISO-8859-1 even though the API function expects a wide character encoding. OSR 2.0 does support Unicode, and the grammar encoding is enforced.

To make these grammars compatible, you must do either of the following:

- Convert the encoding of the string grammar to UTF-16 (or some other wide character format).
- Update your use of the SWIrecGrammarData structure by specifying the *type* as “string/2.0” and using the *binary_data* field to pass the grammar in the original narrow-character format.

Update handling of error codes

In OSR 2.0, there are new error return values, and existing codes have been made more precise. For example, in OSR 1.*n*, various situations could cause the URI_NOT_FOUND error. In OSR 2.0, each of those situations causes an explicit error: URI_FETCH_ERROR, URI_NOT_FOUND, and URI_TIMEOUT.

Platform developers and any application developers who access the SWIep or SWIrec API functions must update their handling of return values. You can reference SWIepAPI.h and SWIrecAPI.h for the enumerations; your code should handle every possible code (and, for future forward compatibility, should include a default case to handle unexpected return values). For details on error code meanings, see the *OSR Reference Manual*.

New codes since OSR 1.1

SWIep_ERROR_INVALID_MEDIA_TYPE
SWIep_ERROR_INVALID_SYSTEM_CONFIGURATION

SWIrec_ERROR_DUPLICATE_GRAMMAR
SWIrec_ERROR_INVALID_CHAR
SWIrec_ERROR_INVALID_MEDIA_TYPE
SWIrec_ERROR_INVALID_SYSTEM_CONFIGURATION
SWIrec_ERROR_INVALID_LANGUAGE
SWIrec_ERROR_URI_FETCH_ERROR
SWIrec_ERROR_URI_TIMEOUT

SWIrec_SERVER_ALLOCATED
SWIrec_SERVER_FREED
SWIrec_SERVER_NOT_ALLOCATED

New codes since OSR 1.0

SWIep_ERROR_LICENSE_ALLOCATED
SWIep_ERROR_LICENSE_COMPROMISE
SWIep_ERROR_LICENSE_FREED
SWIep_ERROR_NO_LICENSE

SWIrec_ERROR_LICENSE_ALLOCATED
SWIrec_ERROR_LICENSE_COMPROMISE
SWIrec_ERROR_LICENSE_FREED
SWIrec_ERROR_NO_LICENSE

Update event log scripts

This task might be required for platform or application developers who have created scripts to extract or display information from the event log.

As described in "SWIrecRecognizerCompute( ) – status code change" on page 25, the status value for a recognition failure has changed from STATUS_FAILURE to STATUS_NO_MATCH. Coinciding with the change to the status is a change to the RSTT token in the event log. In OSR 2.0, the token's value is RSTT=nomatch. Previously in OSR 1.*n*, the value was RSTT=fail.

You should investigate whether those scripts explicitly use the RSTT=fail token value, and if they do, change the scripts to handle "nomatch" instead of "fail."

Update integrations of audio write functions

This section address changes to the handling of audio by SWIepWrite( ) and SWIrecAudioWrite( ). They perform more processing on behalf of the platform, they handle new endpointer modes, they support Aurora features, and they simplify the transfer of linear audio.

Platform developers must update their integrations in some instances, and updates are strongly recommended even when not required.

These changes are backward compatible: OSR 1.*n* integrations will continue to work without change (except when using linear audio, which requires a change). Platform developers are strongly advised to upgrade their integrations to the new uses of these functions. See details below.

Changes required for linear audio

OSR 2.0 uses bytes to describe the length of audio input; previously, the length was measured in "samples." This change has no effect on platforms using u-law and a-law formats (because each sample is one byte), but changes how linear input is handled (where each sample is two bytes).

To support linear audio, platform developers must modify their integration to use the new data structures defined for SWIepWrite( ) and SWIrecAudioWrite( ), which are described in the *OSR Reference Manual* and the *OSR Platform Integration Manual*.

Related to the change from samples to bytes is the limit to the maximum size of the audio buffer. In OSR 1.*n*, the limit was 4,000 samples; in OSR 2.0 the limit is 4,000 bytes.

Changes recommended for lost or suppressed samples

This task is strongly recommended, but not immediately required.

Both OSR 1.*n* and 2.0 require platforms to indicate when audio samples contain dropped packets (lost samples) or when the samples have been silenced to eliminate noise (suppressed samples).

The mechanism for indicating lost or suppressed audio samples has changed. Previously, the samples needed to be sent as an explicit media type. In OSR 2.0, a status field is set for the sample in the calls to SWIepWrite( ) and SWIrecAudioWrite( ), and the media type is the same as the unsuppressed samples.

The changes are backward compatible: OSR 1.*n* integrations will continue to work without change. However, platform developers should update their use of the functions: *the new usage will be required in a future release*.

Update grammar assignments of all URI variables

For all grammars except built-in grammars, OSR 2.0 requires all grammar variable assignments via URIs to be prefixed with the "SWI_vars." string. This significant change enables better grammar caching, prevents namespace collisions with URI variables intended for the web server, and is more similar to other web server uses of URI modifiers.

All applications (migrating from OSR 1.1.2 and earlier) that use this variable passing mechanism *must* be modified. For OSR 1.1.3 or higher, your applications should already be using "SWI_vars."

To update you *must add* a prefix of “SWI_vars.” to each URI variable in every grammar that receives such variables. This is relevant for all API functions that manipulate grammars:

- SWIrecGrammarLoad( )
- SWIrecGrammarActivate( )
- SWIrecGrammarDeactivate( )
- SWIrecGrammarDestroy( )

To pass a variable to a grammar, the application appends the variable name and value to the grammar's URI. In turn, the grammar accesses the variable through the SWI_vars object. For example, the following URI sets the "today" variable to a value:

```
file://mygrammars/birthdate.grxml?SWI_vars.today=20010922
```

The birthdate grammar might use the variable as follows:

```
   today = SWI_vars.today ? SWI_vars.today : '20010630';
```

For details on SWI_vars, see the *OSR Developer's Guide*. In addition, all examples in the guide have been updated to show the usage of SWI_vars where appropriate.

Update grammar preload configuration

This task is required for some OSR 1.1.4 (and above) client-server environments. If you used the SWIsvcPreLoadGrammarsFile configuration parameter to preload grammars, you must update your configuration to use swirec_preload_file instead.

The reason for this change is that the preload feature is now available on all recognition machines (not just OSR servers).

For details, see the *OSR Reference Manual*.

Convert romanized grammars and dictionaries

This task is required for developers of Asian-language applications who previously wrote their grammars and dictionaries with Latin-1 encoding. All others can ignore this task.

OSR 2.0 supports an extensive variety of unicode encodings such as UTF-8 and UTF-16 as well as many double-byte encodings such as Shift-JIS. Developers can write grammar files and pronunciation dictionaries in their native languages. OSR 1.*n* allowed only Latin-1 encoding, and ScanSoft advised Asian developers to use a temporary solution by writing these files in a romanized form. In the case of Japanese, the romanized form is called "romaji." The temporary solution is no longer needed or supported, and the romanized files must be updated.

All grammars and dictionaries must be written in their native encodings. For example, Japanese romaji text is not allowed. For additional details, see the *OSR Language Supplement* for your language.

Update EOS handling

If your OSR 1.*n* integration used the endpointer to detect end of speech (not a recommended practice) then you must explicitly set an end-of-speech event.

OSR 2.0 supports new barge-in modes to enable new classes of application functionality:

- *begin_only* is the most commonly-used mode (and is the default); it was also supported in OSR 1.*n*. The speech detector detects the beginning of speech while the platform terminates the current prompt, and sends the speech to the recognizer.

- *selective_barge_in* and *magic_word* are new modes that allow the endpointer (speech detector) and the recognizer to be active *while the caller continues to speak*. Thus, applications can allow callers to speak without barging in until a specific phrase or command is recognized.

For details on the new modes, see the *OSR Platform Integration Manual*.

CHAPTER 4

Recommended migration tasks

Test your applications

The OSR 2.0 release is designed to be fully backward compatible. However, it is not possible to foresee all possible interactions between OSR, platform, and application code. ScanSoft recommends a complete regression test of applications after upgrading from 1.*n* to 2.0.

Recalibrate your use of confidence scores

OSR 2.0 uses newly calibrated confidence scores for recognition results; the new scale is more refined. The purpose of this change is to allow applications to treat confidence scores as probabilities. For example, applications can multiply confidence scores across multiple recognitions and recompute confidence score after eliminating some entries on an n-best list.

The following table compare the scales of confidence scores. When updating an application's use of confidenceslevel, if OSR 1.n values are between the values shown in the table, you can extrapolate linearly to determine the desired OSR 2.0 value.

New scale OSR 2.0	Old Scale OSR 1.n
0	0
86	410
149	520
198	580
281	640

New scale OSR 2.0	Old Scale OSR 1.n
330	690
477	735
662	825
840	889
931	943
966	975
1000	1000

ScanSoft recommends that platform developers and anyone who directly accesses the SWIrec API update their use of confidence scores to the new scale as soon as possible. VoiceXML developers should modify the confidence scores defined in their vxml documents. You should modify any calculations based on the scores; for example, this means updating any grammar scripts that are dynamically changed and re-used based on recognition confidence (a technique sometimes called "re-scoring").

By default, the new scale is used but you can revert to the original OSR 1.*n* scale with a configuration parameter. If you cannot recalibrate immediately, you should set the swirec_backward_compatible_confidence_scores parameter so that applications continue to receive scores with old OSR 1.*n* scale. (The compatibility parameter is described in the *OSR Reference Manual*.) This is not required, but if you do not use the parameter, borderline utterances that would have been accepted in OSR 1.*n* might be rejected in 2.0.

Use the new SWIepRead( ) function

In OSR 2.0, the new SWIepRead( ) should be used to get audio from the speech detector. Previously in OSR 1.*n*, platform developers managed all audio buffering as they wrote audio to the speech detector, received status codes from the detector, and then formatted the audio with the added information before sending it to the recognizer.

SWIepRead( ) performs all needed formatting and buffering on behalf of the platform, which can now send the audio directly to the recognizer. In addition, SWIepRead( ) supports the new magic work and selective barge-in modes; using SWIepRead( ) means that the platform does not need to handle the audio differently when the new modes are used. Note: the SWIepWrite( ) function has not changed and the behavior of OSR 1.*n* code is unchanged.

Optionally, platform integrators can use the maxspeechtimeout parameter to interact with this function. For more information, see page 22.

Use SWIrecGetXMLResult( ) to get recognition results

In OSR 2.0, the updated SWIrecGetXMLResult( ) function returns all needed recognition information (e.g. answers and confidence scores) for all entries in the n-best list. ScanSoft strongly recommends that platform developers use this function.

Previously, in OSR 1.*n*, the platform would typically call SWIrecGetNumAnswers( ) to determine the number of n-best answers available, and would subsequently call SWIrecGetKeyList( ) and SWIrecGetKeyValue( ) for each key to retrieve values and confidence scores.

The new function is much simpler and more efficient, especially with respect to VoiceXML and IETF specification compliance. It also correctly handles all forms of ambiguous results, including homophones. The older API function is limited in how it can return ambiguous results. ScanSoft anticipates that most platform developers (and application developers using the SWIrec API) will want to use SWIrecGetXMLResult( ) exclusively in the future. *ScanSoft also anticipates that SWIrecGetNumAnswers( ), SWIrecGetKeyList( ), and SWIrecGetKeyValue( ) will be removed from OSR in a future release.*

Update use of SWIrecGrammarLoad( )

In OSR 2.0, the data structure associated with the SWIrecGrammarLoad( ) function has new fields for various enhancements. ScanSoft recommends updating your usage of the function. However, the 2.0 changes are backward compatible and no changes are required.

Examples of the enhancements:

- You can put a grammar into a buffer, and then point to that buffer when calling the function. (For historical reasons, these grammars are called “string” grammars even though the buffer can now contain binary information.)
- There are new VXIMap properties for controlling the grammar caching mechanisms.
- You can load URI grammars that are compliant with web server media types. The following types of grammars are supported:

Grammar types
NULL

Grammar types
application/srgs+xml
application/x-swi-grammar
application/x-swi-parrameter

For more information on this function, see the *OSR Reference Manual*.

Move configuration changes to user-defined files

OSR 2.0 lets you define configuration files that reside outside of the ScanSoft directory and file system. In OSR 1.*n,* you could only change configuration settings by modifying the files provided by ScanSoft.

The new behavior has several advantages including the ability to have different configuration for different systems or applications, and the freedom of changing configuration settings without disturbing the default values from ScanSoft.

The new behavior applies to these configuration files:

- Baseline.xml
- SpeechWorks.cfg

In OSR 2.0, you can create your own configuration files to override the default parameter settings in each of the above files. Your user-defined files are not required, but if your OSR 1.*n* system has modified the default Baseline.xml or SpeechWorks.cfg, you should move those changes to user-defined files.

For details on configuration files and their storage locations, see the configuration chapter in the *OSR Reference Manual*.

Implement the confidencelevel parameter

The confidencelevel parameter is a tool for rejecting utterances with the recognizer. The confidence level is a measure of the recognizer's confidence that it has matched the correct result. The use of this parameter is optional, but recommended, in OSR 2.0.

Without this parameter in OSR 1.0, the platform or the application must evaluate every recognition result (no matter how low the confidence score) to determine whether to accept, reject, or confirm the utterance.

With the parameter in OSR 2.0, the platform or application can pre-indicate the meaning of confidence *before* the recognizer returns a result. Thus, the return status from SWIrecRecognizerCompute( ) can be used to determine acceptance or rejection.

A typical application sets this parameter via a VoiceXML property. Note that VoiceXML defines this property as a floating point number ranging from 0.0 to 1.0. Because OSR requires a range from 0 to 1000, the OSR platform is responsible for re-scaling this to the range used by OSR. The example below sets a value of 0.2, which the platform would convert to 200:

```
<property name="confidencelevel" value="0.2"/>
```

In OSR 1.*n*, this parameter was ignored if specified. This is not likely to cause any problems, but now that the parameter is used it will cause low-confidence recognitions to fail (with a no_match status) where they would have succeeded previously.

Update all dictionaries to new format

In OSR 2.0, the user dictionary format uses simple XML syntax. Previously in OSR 1.*n*, the format used a proprietary syntax. ScanSoft recommends updating to the new format to take advantage of the simplicity and clarity as well as the supported encodings and languages. The new syntax is described in detail in the Pronunciation Dictionary chapter of the *OSR Developer's Guide*.

To continue using the 1.*n* format, ScanSoft recommends adding a language ID to the dictionaries. See "Add language IDs to old dictionaries" on page 19.

Old proprietary format

```
tomato ^ t ix m aa t ow
tomato ^ t ix m ey t ow
George ^ jh ao r jh
record ^ r eh k er d
record ^ r ix k ao r d
```

New XML format

The following example contains a superset of the information shown in the old format above. Note that the XML format allows pronunciations for multiple languages in the same file and within the same <lexicon> element.

```
<?xml version="1.0" encoding="UTF-8" ?>
<lexicon xml:lang="en-US">
   <entry key="tomato">
      <definition value="t ax m ey t ow" />
      <definition value="t aa m aa t aa" xml:lang="zh-tw" />
      <definition value="t aa m aa t ow" />
   </entry>
   <entry key="George">
      <definition value="jh ao r jh" />
      <definition value="j aa j" xml:lang="zh-tw" />
   </entry>
   <entry key="record">
      <definition value="r eh k er d" part="noun" />
      <definition value="r ix k ao r d" part="verb" />
   </entry>
</lexicon>

<lexicon xml:lang="fr-CA">

   ... Canadian French entries...>

</lexicon>
```

Load user dictionaries with <lexicon> instead of <meta>

In OSR 2.0, the preferred technique for loading a user dictionary is to use the <lexicon> element with a URI pointing to the dictionary. Previously, in OSR 1.*n*, the dictionary was loaded with a <meta> that set the swirec_user_dict_name parameter, which always pointed to a file.

ScanSoft recommends changing all these uses of <meta> to <lexicon> as described in the *OSR Developer's Guide*. This is not required immediately since the use of <meta> for this purpose is supported temporarily; however, *support will be discontinued in a future release.*

If your system uses *both* <lexicon> and swirec_user_dict_name, the value of <lexicon> is used.

Set the default language

This task is recommended for multiple-language systems (systems with more than one recognition language installed). You can ignore this task if only one language is installed on your system.

In OSR 2.0, all languages are installed separately from the recognizer itself. (Previously, US English was always installed.) During installation, OSR sets the first language installed as the default language.

If your applications predominantly use some other language than the first installed, you can improve efficiency and performance by setting the DefaultLanguage parameter to that language. (Define the parameter in SpeechWorks.cfg or as an environment variable.) See the *OSR Reference Manual* for more information.

Add language IDs to old dictionaries

This task is recommended for applications that use grammars that recognize multiple languages and continue to use pronunciation dictionaries written in the OSR 1.*n* format. Old-style dictionaries using the OSR 1.*n* format will continue to work in OSR 2.0; you can ignore this task if your grammars never mix languages or if you update dictionary formats (see above).

This optional task is to add language identifiers to every dictionary. When present, this *must be* the first line of the dictionary file (substitute the appropriate language code):

```
!language: xx-YY
```

The language ID is not required, but the absence of an ID can cause subtle recognition problems that are difficult to find when troubleshooting. The remainder of this section provides background for understanding how the language IDs are used.

Below are fragments of an example multi-language grammar; it is a Canadian French grammar that also recognizes some US English phrases. The grammar indicates the predominant language in its first lines:

```
<grammar xml:lang="fr-CA" version="1.0" root="ROOT"
      tag-format='swi-semantics/1.0'
      xmlns="http://www.w3.org/2001/06/grammar">
```

Elsewhere in the grammar, an individual list item is recognized in English:

```
<item xml:lang="en-US">flatlander</item>
```

When a dictionary that contains no language ID is loaded, OSR assumes that the dictionary's language is the same as the grammar's default language. Given the example above, OSR assumes that any old-format user dictionary is Canadian French.

During recognition, OSR uses language-specific pronunciation dictionaries in the documented order of precedence (see the *OSR Developer's Guide*). But in this example, the loaded *user dictionary* is not used for US English entries because it contains no ID and is assumed to be Canadian French. Thus, a subtle recognition problem could arise: as an application developer, you might have created a dictionary entry and correct pronunciation for "flatlander" but the entry would never be loaded into the recognizer or used.

For detailed explanations of multiple-language grammars, see the *OSR Developer's Guide*.

Test grammar loading and update grammars as needed

OSR 2.0 supports grammars written with the XML syntax specified by the W3C Candidate Recommendation June 26, 2002 "Speech Recognition Grammar Specification for the W3C Speech Interface Framework": http://www.w3.org/TR/speech-grammar/. Throughout the OSR documentation, we refer to this draft as the "SRGS grammar specification."

ScanSoft recommends updating older grammars to the most recent specification, and strongly recommends writing all new grammars with the newest syntax.

Grammars written for OSR 1.*n* load without change. OSR 2.0 analyzes the syntax of each grammar to determine which specification the grammar conforms to.

- When loading a grammar, OSR attempts to identify whether the grammar conforms to the January or June specification and then behaves accordingly.
- If OSR cannot identify January syntax in the grammar, it assumes June 2002. This could lead to some OSR 1.0 grammars not loading in OSR 2.0 without modification. The error messages in the diagnostic log file should be sufficiently descriptive to understand which changes are needed.

You can use either specification to write your grammars, but you cannot mix the specifications (a single grammar must not contain instances of syntax that are unique in both specifications).

For information on required elements for June 2002 conformance, see the *OSR Developer's Guide*. Included in the guide is a section on required header information in grammars, which uses this example:

```
<grammar xml:lang="en-US" version="1.0" root="ROOT"
      tag-format='swi-semantics/1.0'
      xmlns="http://www.w3.org/2001/06/grammar">
```

In the example, ScanSoft recommends a value of `'swi-semantics/1.0'` for tag-format instead of the `'semantics/1.0'` that is contained in the SRGS specification. The reason is to distinguish whether semantic understanding is handled by the OSR implementation or the W3C. Currently, the OSR implementation regardless of the value you specify (because the W3C specification for semantic handling has not been finalized).

Here are some differences in the June specification (they were not part of the January 2001 specification to which OSR 1.1 conforms:

- All <meta> tags must appear before any <rule> tags.
- All <meta> tags must contain both "name" and "content" attributes. (OSR 1.1 allows name withouth content.)
- DTMF grammars cannot import speech grammars; and speech grammars cannot import DTMF grammars.
- The <token> tag cannot contain double-quotation marks.
- A rule name cannot begin with the string "xml".
- A rule name cannot contain digits or hyphens (-).

Update SWIrecResourceAllocate( ) and SWIrecResourceFree( )

This is a low-priority migration task that can be performed as a secondary task by platform developers who are updating their integrations or by application developers who are accessing the SWIrec API directly:

- In OSR 1.*n*, the third argument (reserved), was a void *.
- In OSR 2.0, you should update the argument to a wchar_t *.

If you do not perform this update, you might see compiler warnings associated with calls to these functions.

Change swirec_max_speech_duration to maxspeechtimeout

This is a low-priority migration task that can be performed as a secondary task by platform developers.

The swirec_max_speech_duration configuration parameter, which was available in OSR 1.*n*, should be replaced by the maxspeechtimeout timer. (The name can be changed without modifying the value.)

One difference between the parameters is that maxspeechtimeout can be set for the endpointer. (The swirec_max_speech_duration parameter could be set only for the recognizer.)

Use new location for the SpeechWorks.cfg file

In OSR 1.1.*n*, the SpeechWorks.cfg file was used on Linux systems only. If you modifed the file for your platform, you must re-create those modifications in a user-defined Speechify.cfg.

In OSR 1.1.*n* on Linux, the SpeechWorks.cfg file was stored in this location:

```
$SWISRSDK/SpeechWorks.cfg
```

In OSR 2.0, the default location is:

```
$SWISRSDK/config/SpeechWorks.cfg
```

If you used the file for OSR 1.1.*n*, you would have modified the version located in the installation area. This is no longer the case in OSR 2.0. Instead, you should replace the installed version with your own version located in a platform- or application- specific location. Use the SW_CONFIG_FILE environment variable to point to that location. See the configuration chapter of the *OSR Reference Manual* for details.

CHAPTER 5

Miscellaneous migration issues

New format for log files

OSR 2.0 log files are UTF-8 encoded; previously, they were Latin-1. This change applies to both the event logs and the diagnostic logs.

If you have tools that do not accept UTF-8, you can convert the new files to Latin-1 (or any other encoding) with the UCONV tools that is supplied with OSR 2.0. For details, see the file conversion appendix in the *OSR Reference Manual*.

If you prefer Latin-1 output, you can use the the swirec_default_log_encoding parameter to control the log file format.

Waveform logging enabled by default

In OSR 2.0, the default value of the swirec_suppress_waveform_logging parameter is 0. This means that waveforms will be logged on all channels by default. This is the reverse of OSR 1.0, where waveforms were logged only if specifically enabled on a channel.

Platforms that wish to limit waveform saving, for performance or disk space conservation reasons, need to implement a mechanism that suppresses waveform logging on a subset of active channels.

New RIFF format for waveform output

The swirec_suppress_waveform_logging parameter control the saving of audio files that contain the utterances of callers to applications. In OSR 2.0, these audio files are saved in RIFF format (Resource Interchange File Format). Previously in OSR 1.*n*, the files were saved in raw µlaw format.

RIFF adds header information that was not present in OSR 1.*n* audio files. If you have custom tools to play the old-format audio files, you will need to modify them.

The audio returned by the SWIrecGetWaveform( ) function has not changed; it does not add the RIFF header to returned waveforms.

New format for parseTool output

The parseTool program in OSR 2.0 writes output in the same XML result format returned by SWIrecGetXMLResults( ). The old output format of parseTool is not supported.

Application and platform developers that have written tools which depend on the output of parseTool will need to modify them to handle the new output format.

New acoustic adaptation (LEARN)

The OSR 1.*n* LEARN feature is changed, and the name LEARN is no longer used to describe the self-learning feature. Previously, the self-learning was performed by operators as a batch operation. In OSR 2.0, the feature is significantly enhanced.

In most OSR installations, the self-learning feature is nearly invisible to platform developers and application developers. But in certain instances it is necessary to change the default configuration:

- Application or grammar developers should suppress self-learning for certain types of grammars (for example, SpeakFreely and phoneme-loop grammars).
- Platform integrators who build a multiple-process OSR architecture should allow self-learning on only one process per machine.

- Sites with more than one OSR machine that require identical installations on each machine should suppress automatic self-learning updates and perform batch updates instead.

For an overview and details, see the Architectural Overview chapter in the *OSR Platform Integration Manual.*

SWIrecRecognizerCompute( ) – status code change

SWIrecRecognizerCompute( ) returns status codes to indicate recognition progress. The OSR 1.*n* code SWIrec_STATUS_FAILURE has been changed in OSR 2.0 to:

```
SWIrec_STATUS_NO_MATCH
```

The purpose of the change is to clarify the meaning of "failure," which is that the recognition could not find a match for the spoken utterance in the active grammars.

For compatibility with OSR 1.*n*, SWIrec_STATUS_FAILURE is defined as SWIrec_STATUS_NO_MATCH and the enumeration value has not changed (see SWIrecAPI.h).

Removed configuration parameters

Several OSR 1.*n* parameters are removed from the OSR 2.0 code. These parameters were not publicly documented and their removal should have no effect for migrating platforms or applications.

However, if you copy the removed OSR 1.*n* configuration parameters to an OSR 2.0 user configuration file, your system will generate configuration errors or warnings upon startup.

- swi_augment_mode – replaced by SWI.type; see "Pronunciation dictionary precedence" on page 26.
- swirec_lmfloor – generates an error if used.
- swirec_lmceiling – generates an error if used.
- swirec_lmoffset – hardcoded to 0; ignored if used in a configuration file.

Activating duplicate grammars

In OSR 2.0, a grammar cannot be activated more than once for the same recognition event. Previously, OSR 1.*n* ignored these duplicate activations, but now OSR 2.0 generates an error (see “Update handling of error codes” on page 8) if SWIrecGrammarActivate( ) is called for a grammar that is already active.

This change only affects platforms or applications that inadvertently perform duplicate activations. For example, errors can occur if the platform omits balancing calls to SWIrecGrammarDeactivate( ) for every grammar used.

Pronunciation dictionary precedence

When OSR determines the pronunciations of words, it searches user and system dictionaries exclusively (if a pronunciation is found in one dictionary, the other is not consulted).

In OSR 2.0, you can now change the precedence of dictionaries with the SWI.type property. This allows you to change the order that dictionaries are searched and to use pronunciations from more than one dictionary (non-exclusively). For details, see the *OSR Developer’s Guide*.

Merged OSR client-server architecture

As of OSR 1.1.4, the client-server and all-in-one architectures are merged into a single product. You can use the same installation CD for either architecture.

Detailed information for client-server environments is merged throughout the OSR documentation set, and a new document (the *OSR Client/Server Operator’s Guide*) describes specific details for that environment.

Using OSR 1.*n* acoustic models with OSR 2.0

If you are using customized acoustic models on your OSR 1.*n* system, you can continue to use them with OSR 2.0. However, you must also use the 1.*n* PCS file (Principal Components file) that was shipped with your OSR 1.*n* system. Otherwise, the recognition accuracy of your system will drop precipitously.

To specify the file, do the following:

1. Open your 1.*n* Baseline.xml file and consult the value of the swirec_pcs_name parameter for the language you are using. This allows you to find the desired PCS file.

2. Copy the PCS file to an appropriate (platform- or application-specific) location on your system.

3. Add the swirec_pcs_name parameter to a language-specific block of a user configuration file in your OSR 2.0 installation. Set the value to the file's new location.

Index

A

API issues
- confidence scores 13
- confidencelevel 16
- error codes 8
- STATUS_FAILURE 25
- STATUS_NO_MATCH 25
- SWIrecGetXMLResult() 15
- SWIrecGrammarLoad() 15
- SWIrecRecognizerCompute() 25
- SWIrecResourceAllocate() 21
- SWIrecResourceFree() 21
- VXIMap properties 15

Audio issues
- linear audio 9
- SWIepWrite 9
- SWIrecAudioWrite 9
- waveform output 24

B

barge-in modes 12
Baseline.xml 16
begin_only endpointer mode 12
Binary grammar recompilation 6

C

confidence scoring 13
confidencelevel parameter 16
configuration files
- SpeechWorks.cfg 22

Configuration issues
- confidence scores 14
- confidencelevel 16
- removed parameters 25

Conversion utility for file encoding 23

D

Dictionary issues
- loading 18
- new XML format 18
- precedence of 20

duplicate grammars 26

E

Encoding, of log files 23
error code changes 8

G

Grammar issues
- activating duplicate grammars 26
- loading dictionaries 18
- re-compiling grammars 6
- romanized grammars 11
- string grammars 15
- SWIrecGrammarLoad() 15
- testing grammars 20
- variable assignments 10
- W3C compliance 20

I

Installation issues
- language installs 5
- set default language 19

J

Japanese romaji, in grammars 12

L

lexicon element
- in dictionaries 18
- loading dictionaries 18

Logging issues
- event log encoding 23
- RSTT token change 9
- waveform output 24

P

Parameter issues
- confidence scores 14
- confidencelevel 16
- removes params 25

Precedence, of dictionaries 20
Pronunciation dictionaries, see Dictionaries

R

RIFF waveform output 24
romaji 11
romanized grammars 11
RSTT token 9

S

Scripts, for event logs 9
SRGS grammar specification 20
STATUS_FAILURE 9
STATUS_NO_MATCH 9
string grammar encodings 7
String grammars 15
SWI_vars 11
SWI_vars prefix for variables 10
SWIepWrite() 9

swirec_backward_compatible_confidence_scores
14
SWIrec_ERROR_INVALID_DATA 7
swirec_lmceiling 25
swirec_lmfloor 25
swirec_lmoffset 25
swirec_preload_file 11
swirec_suppress_waveform_logging 24
SWIrecAudioWrite() 9
SWIrecGetKeyList() 15
SWIrecGetKeyValue() 15
SWIrecGetNumAnswers() 15
SWIrecGetWaveform() 24
SWIrecGetXMLResult() 15
SWIrecGrammarLoad() 15
SWIrecRecognizerCompute() 17, 25
SWIrecResourceAllocate() 21
SWIrecResourceFree() 21
swi-semantics 21
SWIsvcPreLoadGrammarsFile 11

U
UCONV conversion tools 23
unicode support 11
URI variables 10
UTF-8 encoding in log files 23

V
Variables, assigning 10
VXIMap 15

W
W3C grammar
  specification 20
  syntax 20

X
XML format for dictionaries 18