



**ASR1500/ASR1600**  
**Automatic Speech Recognition**

**Software Development Kit**  
**for Windows 95/98/NT**

Version 3.22

**Development Tools User's Guide**







---

## EVALUATION AGREEMENT

READ THE FOLLOWING CAREFULLY BEFORE USING THE SOFTWARE.

PRIOR TO RECEIVING THE SOFTWARE, YOU HAVE SIGNED EITHER A LICENSE AGREEMENT OR A NON-DISCLOSURE AGREEMENT WITH LERNOUT & HAUSPIE SPEECH PRODUCTS (L&H), CONCERNING THE L&H ASR1500/ASR1600 SOFTWARE DEVELOPMENT KIT. THEREFORE, REFERENCE IS MADE TO THESE AND AS SUCH THE CONTRACTUAL TERMS AND CONDITIONS SHALL APPLY TO THE SOFTWARE.

IF YOU HAVE NOT SIGNED SUCH AN AGREEMENT AND IF YOU USE THE SOFTWARE, L&H WILL ASSUME THAT YOU AGREED TO BE BOUND BY THE EVALUATION AGREEMENT SPECIFIED HEREUNDER. IF YOU DO NOT ACCEPT THE TERMS OF THIS EVALUATION AGREEMENT, YOU MUST RETURN THE PACKAGE UNUSED TO L&H WITHIN SEVEN (7) DAYS AFTER RECEIPT.

### 1. Grant of Rights

In consideration of a possible commercial relationship, L&H hereby grants to you, the LICENSEE, who accepts, a non-exclusive right to internally evaluate and test the software program ("the Software").

### 2. Ownership of Software

L&H retains title, interests and ownership of the Software recorded on the original disk(s) and all subsequent copies of the Software and Documentation, regardless of the form or media in or on which the original and other copies may exist. L&H reserves all rights not expressly granted to LICENSEE.

### 3. Copy Restrictions

This Software and the accompanying documentation are copyrighted. Unauthorized copying of the Software, including Software that has been merged or included with other software, or of the documentation is expressly forbidden. LICENSEE may be held legally responsible for any intellectual property infringement that is caused or encouraged by his failure to abide by the terms of this agreement. LICENSEE is allowed to make two (2) copies of the Software solely for backup purposes, provided that the copyright notice is included on the backup copy.

### 4. Use Restrictions

LICENSEE agrees not to use the Software for any other purpose than internally evaluating the Software. LICENSEE may physically transfer the Software from one computer to another, provided that the Software is used on only one computer at a time. LICENSEE may not modify, adapt, translate, reverse engineer, decompile, disassemble or create derivative works based on the Software.



---

LICENSEE may not modify, adapt, translate or create derivative works based on the documentation provided by L&H.

The Software may not be transferred to anyone without the prior written consent of L&H. In no event may LICENSEE transfer, assign, lease, sell or otherwise dispose of the Software and Documentation on a temporary or permanent basis except as expressly provided herein.

## **5. Warranty**

THE SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. L&H shall have no liability to LICENSEE or any third party for any claim, loss or damage of any kind, including but not limited to lost profits, punitive, incidental, consequential or special damages, arising out of or in connection with the use or performance of the Software and accompanying documentation.

## **6. Termination**

This agreement is effective until terminated. L&H reserves the right to terminate this agreement automatically if any provision of this agreement is violated. LICENSEE may terminate this agreement by returning the Software and the accompanying documentation to L&H, along with a written warranty stating that all copies have been returned.

## **Copyright**

© Lernout & Hauspie Speech Products N.V.

Document No. D90931-21-08 L&H ASR1500/ASR1600 SOFTWARE DEVELOPMENT KIT

Development Tools User's Guide

V3.22 © - October 1999

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy, recording or any information retrieval system, without the written permission of L&H.

## **Trademarks**

MS-DOS®, WINDOWS®, MICROSOFT® VISUAL C++, BORLAND C++ and Sound Blaster are registered trademarks of their respective owners.

Lernout & Hauspie Speech Products is a registered trademark.

All rights reserved.



---

---

# Contents

<b>INTRODUCTION .....</b>	<b>1</b>
<b>THE ASR EVALUATOR .....</b>	<b>3</b>
Introducing the ASR Evaluator .....	3
Starting the Evaluator .....	5
Menu Commands .....	6
File Menu.....	6
Recognizer Menu .....	10
Tools Menu.....	22
Help Menu .....	24
The Evaluator Program Window.....	25
Utterance Field .....	25
N-Best alternatives Field .....	25
Recognizer Activity Field .....	26
Active Contexts Field.....	27
Active Words Field .....	27
Recognizer Management Buttons .....	27
Start Button.....	27
Stop Button.....	27
Break Button.....	28
Recognition Operating Modes .....	28
Push to talk.....	28
Automatic stop.....	29
Open microphone.....	29
Continuous Operation .....	31
Recognition from File.....	31
<b>THE LEXICON TOOLKIT.....</b>	<b>33</b>
Introducing the Lexicon Toolkit (LexTool) .....	33
Starting the Lexicon Toolkit .....	35
Menu Commands .....	36
Context menu .....	36
User menu.....	44
Dictionary menu .....	49
Help menu .....	54



---

---

<b>THE BNF GRAMMAR LANGUAGE .....</b>	<b>57</b>
Introducing the L&H BNF Grammar Language .....	57
Grammar essentials.....	58
Preliminary.....	58
Notation .....	58
Statements, Rules and Symbols .....	59
Classes.....	61
Repetitious Recognition.....	63
Optional Recognition .....	66
Unnotified Recognition .....	67
Non-Recognitions .....	68
Importing.....	70
Any Speech .....	71
Pronunciations.....	72
Spelling.....	74
Formal Description.....	75
Pre-processing Grammars .....	76
Comment Notation.....	76
Identification Section .....	77
Interface Section.....	78
Class Section.....	79
Pronounce Section .....	81
Rules Section.....	82
Isolated Word Recognition.....	85
Key Word Spotting.....	86
Connected Digit Recognition .....	87
L&H BNF Grammar Language Formal Specification.....	87
Terminal Symbol Notation, Word, Token .....	87
Non-terminal Symbol Notation, Symbol .....	88
Garbage Class.....	88
Compiler Keywords Specification .....	88
Formal Language Specification .....	89
Design Tips .....	91
Content .....	91
Limit the use of 'special features' .....	93
Vocabulary Management.....	95
<b>THE L&amp;H+ PHONETIC TRANSCRIPTION SYSTEM .....</b>	<b>97</b>
Introduction .....	97
Multiple Pronunciations.....	98

---



---

---

Phoneme Table American English.....	101
Phoneme Table Spanish.....	103
Phoneme Table Italian .....	105
Phoneme Table German.....	109
Phoneme Table British English .....	111
Phoneme Table French.....	113
Phoneme Table Dutch.....	115
Phoneme Table Japanese .....	117
Phoneme Table Korean .....	121
Phoneme Mexican Spanish.....	125



---

---



# INTRODUCTION

The ASR1500/ASR1600 Software Development Kit provides a set of development tools that you can resort to at different stages of designing, implementing, evaluating and testing the speech-recognition features of your application(s). Those tools include the Automatic Speech Recognition Evaluator, the Lexicon Toolkit, the L&H BNF Grammar Language and the L&H+ Phonetic Transcription System.

The different tools are installed automatically by the ASR SDK installation program.

If you are not familiar with ASR programming or with the SDK and its features, read the Introduction and the ASR Primer in the *Getting Started* manual to learn about the basic concepts.

This manual is organized into the following chapters:

**The ASR Evaluator** describes the graphic user interface of the ASR Evaluator and provides details about its features.

**The Lexicon Toolkit** details the interface of the Lexicon Toolkit (LexTool) and explains how the LexTool can help you to perform several context management tasks.

**The L&H BNF Grammar Language** provides a comprehensive description of the L&H BNF grammar language, the essential tool to writing speech recognition grammars.

**The L&H+ Phonetic Transcription System** contains some information about phonetic input in general, together with phoneme tables for the different languages supported by the SDK.



---

---



# THE ASR EVALUATOR

## Introducing the ASR Evaluator

The ASR Evaluator is a Windows program, based on the L&H Automatic Speech Recognition (ASR1500/ASR1600) Software Development Kit. The program serves a twofold purpose. For people who are new to the SDK, it constitutes a very straightforward tool to explore and evaluate the state-of-the-art L&H Automatic Speech Recognition technology. For people who are already in the stage of designing a speech-enabled application, it offers a quick and easy way to test the speech recognition contexts they may have created.

As it has primarily been designed to evaluate the ASR SDK, the Evaluator incorporates most of the relevant features the SDK has to offer. Thus, it helps you get a very good idea of the performance of the L&H continuous speech engine for the environment(s) and the language(s) you have purchased and for the equipment that you are using. It also illustrates the possibilities and the flexibility obtained from the Recognizer Management Service functions in the SDK.

In addition to the interactive evaluation facilities, the ASR Evaluator is equipped with Windows Drag & Drop functionality for off-line evaluations with prerecorded sample files. You select the prerecorded wave files in the Windows Explorer and drag them onto the ASR Evaluator program Window. The ASR Evaluator sequentially recognizes the selected wave files and shows the recognition results on the screen. By submitting the same files repeatedly, you can investigate the effect of the various recognizer parameters.

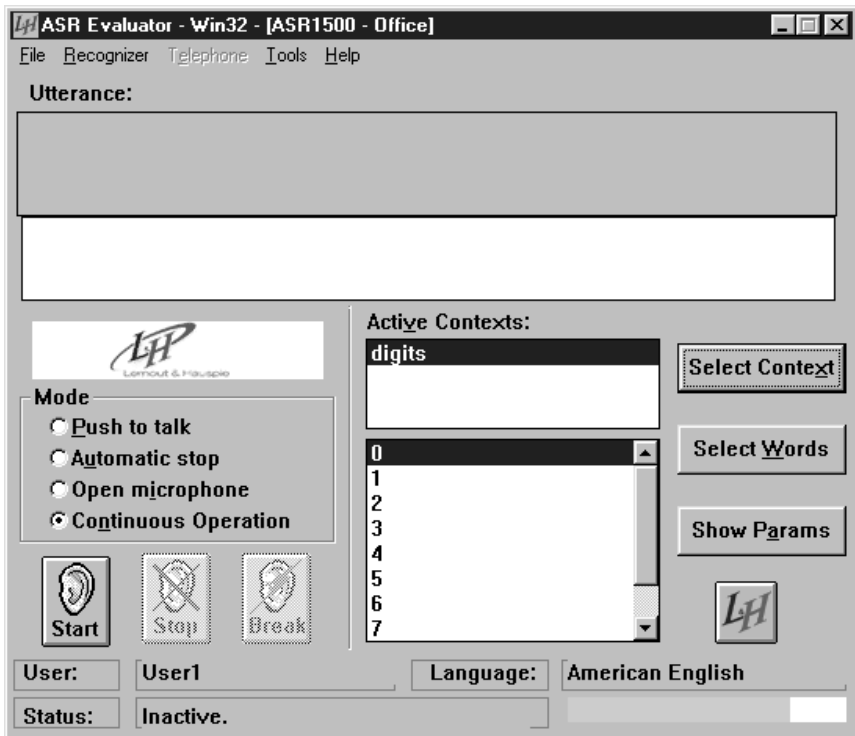


Another interesting feature of the Evaluator is the data logging of all the recognition results and user actions taking place during an evaluation session. In combination with the off-line evaluation of speech wave files, this logfile provides the basic information for possible statistical analysis of the performance of the recognition engine during a specific speech recognition task.



## Starting the Evaluator

The installation of the ASR1500/ASR1600 SDK automatically adds a program group to the Windows Programs menu. The Evaluator is one of the items in this group. When you click its name in the list, the program starts up and the main window appears on the screen.



The title bar of the program window displays information about the ASR database, i.e. the selected engine and environment.



**Note:**

Before you can start up and use the Evaluator, you have to create a user and a context and you have to register the user for that context and for a language.

For detailed instructions on how to do this refer to the chapter about the Lexicon Toolkit.

The following sections provide a systematic description of the different items in the program window.

## Menu Commands

### File Menu

#### Open Logfile

This command allows you to view the logfile you can have the Evaluator record. It opens a new window with the Notepad text editor.

The ASR Evaluator can provide data logging of all user actions and recognition results that occur during an evaluation session. To enable the logging, you must mark the Log to File checkbox in the Configuration dialog box which is accessible via the File Menu.

Below you see an example of a log file with the type of data it contains.



```

old_asreval.log - Notepad
File Edit Search Help
[General information]
Date and Time = Tue Sep 08 09:43:30 1998 (1200942 ms)
[Initial parameters]
ASRREC_ACCURACY = 4939
ASRREC_SENSITIVITY = 3000
ASRREC_TS_COUNT = 314
ASRREC_NBEST = 1
ASRREC_REJECTION = 70
ASRREC_GARBAGE = 40
ASRREC_MINSPEECHDURATIONFORSTART = 60
ASRREC_ACCEPTABLE = 1
ASRREC_ACCEPTANCE = 0
ASRREC_TIMEOUT = 0
ASRREC_FARTALK = 1
ASRREC_WORDPENALTY = 500
ASRREC_NAMESEARCHACCURACY = 2000
13 = 330
14 = 400
Tue Sep 08 09:43:36 1998 (1207346 ms) Set recognizer mode: ASRMODE_AUTOMATIC_STOP
Changed the configuration to

Database Name : ASR1500 - Office
Environment Name: Win32
Language Name : American English
User Name : Luc Braems

Changed the configuration to

Database Name : ASR1500 - Office
Environment Name: Win32
Language Name : American English
User Name : Luc Braems
  
```

## Change Configuration

Use Change configuration if you want to change the global settings of the Evaluator.

When you click this function in the File menu, the Configuration dialog box appears:

**Configuration**

Database: ASR1500 - Office Env.: Win32

User: User1

Language: American English Adapted

Input Device: SB16 Wave In [220] Change

☒ Log to file: File C:\ASREVAL.LOG

OK Cancel



The settings you can adapt are the following:

- The Database

This setting determines the recognition engine and the sampling frequency the Evaluator will be using. Possible choices depend on the type and the number of ASR Evaluators or SDK options you have installed on your system. For the engine, it can be “ASR1500”, “ASR1600” or “ASR1602”; for the sampling frequency, “Office” indicates input at 11 kHz, whereas “Telephone” stands for input at 8kHz.

- The Environment

At present, this setting does not have any effect, as Win32 is the only valid environment for both Windows 95 and Windows NT.

- The User

Here you select a user as you have created it in the LexTool.

**Note:**

You can select only 1 user at the time, even if you have created several users in the LexTool.

For details on how to create users in the LexTool, refer to the next section in this manual. For more general information about users, read the ASR primer in *Getting Started*.

- The Language

Here you select the language for which you want to evaluate speech recognition. Possible choices depend on the languages you have ordered with your SDK or Evaluator

**Note:**

Before you can select a language for evaluation, you must register it to the user you have selected.



- The Input Device

With this selection, you determine the type of input handling and the wave device to be used during evaluation.

As to the input handling types, “Internal” means that the engine takes care of the sampling; “Wave” means that the application controls the sampling; “TAPI” means that extra controls are invoked to control the telephony board, the actual sampling being the same as for “Wave”.

**Note:**

You can only use the option “TAPI” if you have a telephony board with the necessary drivers to support TAPI.

As to the selection of the actual wave device, the choices depend on the hardware you have installed on your system.

When you select “Wave” or “TAPI” as the input type, you can save the speech input into a file by checking the option Save input to file. The spoken utterances are saved as a wave file *xx.wav* (*xx* represents a number) in the same directory as the one you have selected to write the logfile.

In the bottom part of the Configuration dialog box, you find the option to enable data logging. Mark the Log to file check box to activate the logging.

If you want to change the path for the logfile, press the File button and specify the appropriate directory and file name. If you specify the name of a file that already exists, the application notifies you and asks for confirmation.

## Exit

Use the option Exit from the File Menu to quit the Evaluator.



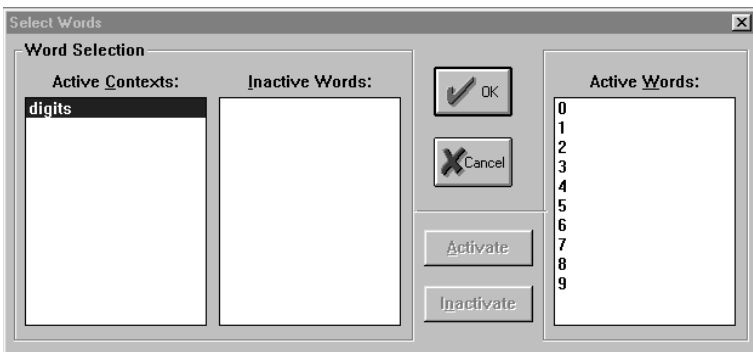
## Recognizer Menu

### Select Words

Use this function to select a set of words (a sub-vocabulary) from the vocabulary of an active context and activate them on the recognition engine.

This feature facilitates fast switching of active words in menu-based command-and-control applications. You can put all the voice commands of all the menus in one vocabulary, but activate only the ones that are needed at a certain stage in the execution of the program. This results in better and faster recognition.

To obtain the Select Words dialog box, you can click the Select Words command in the Recognizer menu or you can push the Select Words button in the main window of the Evaluator.



The dialog box contains three list boxes:

- Active Contexts

Contains the name of the context from which you want to select the sub-vocabulary. If you registered and opened several contexts for the active user, their names will be in the list box. When you click one of



the context names, the words of that context are displayed in the Inactive Words and/or in the Active Words list box, depending on their status.

- Inactive Words

Contains the list of words that are not active on the recognizer.

- Active Words

Contains the list of words that are active on the recognizer. It is only these active words that can be recognized.

To make a word active or inactive, select it from the appropriate list box and respectively push the Activate or Inactivate button or just double-click it in the list box.

**Note:**

When a context is activated on the recognition engine, all the words in that context are made active by default.

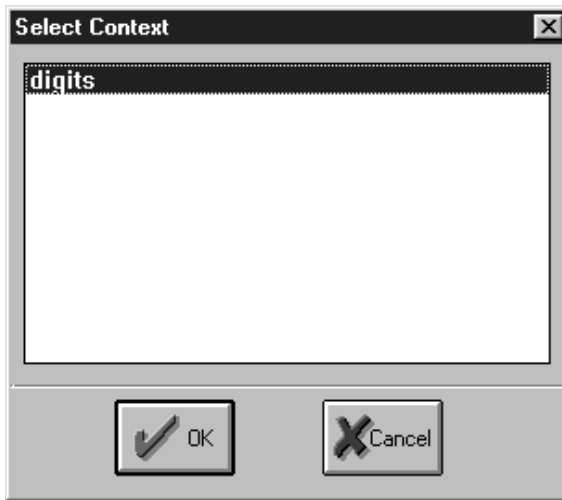
**Select Contexts**

Use this function to select a context and activate it on the recognition engine.

For the recognizer to recognize speech, you must activate one or more contexts, as these provide the necessary information to the recognizer about possible utterances to be recognized.

When you choose Select Contexts, the Select Context dialog box appears.





The dialog box contains a list of all the contexts available for the user, language and database that you have selected in the Configuration.

Select the context you wish to activate and click the OK-button. If you want to activate multiple contexts, you can select them simultaneously, as this is a multi-section list box.

You can open the Select Context dialog box also through the Select Context button in the main window of the Evaluator.

### **Show Parameters**

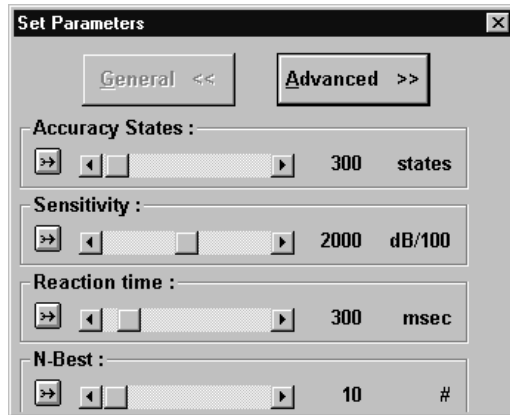
Use the option Show Parameters to view and/or change engine parameters.

By changing these parameters, you can influence the behavior of the recognition engine in order to obtain the best possible recognition results and performance. The best way to evaluate the impact of the different parameters is when using prerecorded wave files as input for the recognition. Thus, you can submit the same files a number of times with different parameter settings.



When you choose this function from the Recognizer menu, the Set Parameters dialog box appears.

You can also push the Show Params button in the Evaluator main window to access this dialog box.



For each parameter, the dialog box contains:

- a button to restore the default value
- a slider that lets you increase or decrease the value for the parameter at your discretion
- the current value for the parameter
- the unit for the parameter

The parameters are subdivided in three groups. When you open the dialog box it contains only the “General” parameters. You can extend it with the “Advanced” and the “Expert” parameters by means of the buttons at the top.

When you push the Advanced button, the dialog box is extended with a first set of parameters.

**Note:**

Depending on the selected recognition engine, the Set Parameters dialog box may contain different values or even different parameters.



The image shows a 'Set Parameters' dialog box with a title bar and a close button. At the top, there are two buttons: 'General <<' and 'Expert >>'. The 'Expert >>' button is highlighted with a dashed border. Below these buttons, there are ten parameter settings, each with a label, a slider control, and a value with a unit. The parameters are: Accuracy States (300 states), Sensitivity (2000 dB/100), Reaction time (300 msec), N-Best (10 #), Rejection penalty (75), Garbage penalty (75), Minimum speech duration for start (60 msec), Automatic Gain Control (0 boolean), Acceptance threshold (5000), Time out (0 sec), and Far Talk (1 boolean).

Parameter	Value	Unit
Accuracy States :	300	states
Sensitivity :	2000	dB/100
Reaction time :	300	msec
N-Best :	10	#
Rejection penalty :	75	
Garbage penalty :	75	
Minimum speech duration for start :	60	msec
Automatic Gain Control :	0	boolean
Acceptance threshold :	5000	
Time out :	0	sec
Far Talk :	1	boolean

The buttons at the top of the dialog box allow you to reduce the list again to only the general parameters or to extend it with the expert parameters.



**Set Parameters**

General << **Advanced <<**

**Accuracy States :**  
 300 states

**Sensitivity :**  
 2000 dB/100

**Reaction time :**  
 300 msec

**N-Best :**  
 10 #

**Rejection penalty :**  
 75

**Garbage penalty :**  
 75

**Minimum speech duration for start :**  
 60 msec

**Automatic Gain Control :**  
 0 boolean

**Acceptance threshold :**  
 5000

**Time out :**  
 0 sec

**Far Talk :**  
 1 boolean

**Name search accuracy :**  
 5000

**Partial spelling :**  
 0

The overview below briefly explains the meaning of the different parameters as they apply to the different engines and contains some interesting comments on their usage.



### **Accuracy States**

This parameter determines the maximum number of paths the recognition engine can follow while analyzing incoming speech in order to find the best possible acoustic match and come up with a recognition result for a certain utterance.

A higher value increases the recognition accuracy but also the CPU load and memory usage; a lower value reduces the recognition accuracy, but also the CPU load, the memory resources and thus results in faster recognition. The optimum value is a trade-off between the available system resources and the accuracy desired for the application.

The value is set to 8 times max branching, with a minimum of 100 and a maximum of 10000. For larger vocabularies, this default value is to be increased.

### **Sensitivity**

A speech event is defined as an energy jump of a certain amount of decibels during a minimum period (i.e. *Minimum speech duration for start* parameter). The (speech) Sensitivity parameter, expressed in hundredths of dB (decibel), sets the relative energy threshold that an input speech signal must exceed in order to be classified as valid speech.

When working in Open Microphone or Continuous operation mode, a mechanism called the Voice Activity Detector triggers the recognition engine when the signal energy observed exceeds the value for the Sensitivity parameter. This functionality is provided to off-load the CPU during silence and to yield processing time to other applications.

The parameter should be 3-6 dB above the energy level measured during silence.

### **Reaction time**

The Reaction time parameter determines the minimum trailing silence the recognition engine has to detect before it decides that an utterance is complete. This parameter



determines the minimum response time of the recognition engine in real-time operation.

The reason why the setting for the reaction time is not implemented as an internal constant of the engine is because the value required largely depends on the specific application. Recognition of isolated words typically works with 200~300ms of trailing silence, thus keeping the response time of the engine low. Continuous speech recognition grammars, to overcome the problem of the small pauses and hesitations in the speech, need 500~1000ms of trailing silence.

### **N-Best**

The N-Best parameter determines how many alternative recognition results the engine returns to the application.

During the recognition, the engine tries to match the speech input with the different phonetic models it has derived from the recognition context(s). This results in a number of possible recognition results, to which a kind of probability score is attributed, called the *confidence level*.

For the ASR1500 engine, the N-Best parameter is a boolean: with the value set to 0, the engine only returns the recognition result with the highest confidence level; with a value of 1, it returns all the available alternative results.

For the ASR1600, the parameter can have a value ranging from 1 to 100, specifying the number of alternatives you want the engine to return, ranked by order of decreasing confidence levels.

### **Rejection Penalty**

This parameter determines how quickly the recognition engine, when it is unable to recognize an utterance, decides that it is an *out-of-grammar phrase*, and rejects it.

A high penalty value results in virtually no rejection of out-of-grammar utterances; a low value, on the other hand, causes the engine to reject such utterances rapidly.



The default value is set to 75.

To decide on the appropriate value for this parameter, you have to take into account the impact of two possible side effects on your application:

- *false rejection*: an utterance that should have been recognized correctly is rejected; this can occur when the penalty is too low.
- *false acceptance*: an utterance is accepted, although it is out of grammar; this can occur when the penalty is too high.

When the recognizer rejects an utterance, the Evaluator displays ### in the Utterance field of the program window.

**Note:**

For the ASR1600 engines, this is not the result you would expect. In the SDK, the ASR1600 engine always returns an alternative with a confidence value. If you want your application to reject a result and display the ###, you have to take care of that in your program, which has been the case for the Evaluator.

**Garbage Penalty**

This parameter determines how quickly the engine decides that certain parts of the input speech are irrelevant and ignores them.

In some applications, the engine is to recognize just one specific word and ignore the rest of the input speech. When performing this kind of *keyword spotting*, the engine compares the speech to the *garbage model*, presented as <...> in the L&H BNF language.

If the penalty is set high, no sounds match the garbage model; if it is set low, large parts of the speech will match the model, at the risk even of missing the key word.

The default value is 75.



**Minimum speech duration for start**

This parameter sets the minimum duration of the expected speech before the Voice Activity Detector triggers the recognition engine.

**Automatic Gain Control**

This boolean parameter turns the Automatic Gain Control mechanism on (1) or off (0).

For optimum recognition performance it is essential that the recording level of the acquisition hardware is set correctly. Automatic Gain Control allows the recognition engine to adapt the settings if it considers the input signal too loud or too weak.

However, Automatic Gain Control (AGC) generates instantaneous fluctuations of the signal levels, which can have unwanted side effects on the quality of the recognition. Therefore, you should use AGC with the necessary caution.

**Acceptance threshold**

The Acceptance threshold parameter specifies the minimum confidence level needed for the speech recognition engine to accept an utterance and notify the application of the recognition result.

The Acceptance parameter is meaningful in the Open microphone mode only. If the value is set high enough, it allows you to have the recognition engine accept a specific utterance only when it is certain about the recognition result. The confidence level of the best recognition result is calculated and compared to the acceptance parameter. If the value is higher than the one set for the acceptance parameter, the result is accepted, else the result is rejected and the recognition engine waits for a new utterance.

For details on the Open microphone operating mode, refer to the section Recognition Operating Modes further in this chapter.



### **Time out**

The Time out parameter sets the maximum length of the time window in which the recognizer is active. The default value is set to 300 msec.

Bad environmental conditions, excessive background noise, competing signals can cause the end-of-speech detection mechanism to fail. In those cases, you can use the time-out parameter to make sure the recognition engine automatically breaks after a reasonable lapse of time.

This feature is also useful to overcome the “sleeping user” phenomenon in dialog system applications, especially over the telephone, where the user does not react to the question asked.

### **Far Talk**

This boolean parameter determines the sensitivity of the AGC by setting the auto gain algorithm mode for far-talk or close-talk.

If the parameter is set for close-talk microphones (value 0), the AGC uses smaller increments when changing the mixer settings, as the microphone remains at approximately the same distance from the speaker. If the parameter is set for far-talk microphones (value 1), larger increments are used and a wide range of signals can be captured.

### **Word Penalty**

The Word Penalty parameter influences the occurrence of *insertions* and *deletions* during recognition.

Insertions occur when more words are recognized than actually spoken; deletions, on the contrary, occur when fewer words are recognized than spoken.

The parameter is used to change the word entrance penalty. Use a higher value to cope with too many insertions and use a lower value if you have a lot of deletions.

The parameter should be set to 500 for normal recognition and to 2000 for a spelling context.



This parameter is only available for the ASR1500 engine

### **Name search accuracy**

This parameter determines the accuracy for the name search during spelling.

One of the features of the ASR1500/ASR1600 engines is that they are able to recognize words that have been spelled by the speaker.

To use this feature, you must write a recognition grammar in which you specify the words that need to be recognized.

The name-search parameter, similar to the accuracy parameter, determines the number of words that can be processed simultaneously.

As a rule, name searching is rather fast and therefore, the use of a high accuracy value is recommended.

### **Partial spelling**

This boolean parameter, when set, enables the engine to recognize a word with only a few letters spelled.

If several words start with the same letters, they are all considered as valid results and are attributed the same confidence level.

The parameter is only valid for the ASR1600 engines.

### **Show/Hide N-Best**

With this function you can tell the application to either show or hide the N-Best recognized alternatives with their corresponding confidence levels. The name of the function in the menu alternates accordingly.

Refer also to the descriptions of the Acceptance and the N-Best parameters.

The recognized alternatives are displayed in the program window underneath the Utterance field.



### **Rejection at word level**

With this function, you can have the recognition engine reject individual parts of an utterance, rather than the utterance as a whole.

You can write a grammar stating that the utterances to be recognized are a concatenation of different elements from the vocabulary. When Rejection at word level is enabled, the recognition engine considers the confidence levels of these individual elements in the phrase and it rejects only the elements with too low a confidence level.

In the recognition result displayed in the Utterance field, each rejected word is represented as --- .

## **Tools Menu**

### **Lexicon Toolkit**

By choosing this function, you start up another development tool: The Lexicon Toolkit (LexTool).

For detailed information about the different features of the LexTool, see the second chapter in this manual, The Lexicon Toolkit.

### **Add User Word**

Choose the function Add User Word if you want to add a user word to a context.

The words you add to a context via the LexTool are so-called user-independent words. The (phonetic) information necessary to recognize these words is included in the general language model of the ASR database.

User words, on the other hand, are speaker-dependent words. The information necessary to recognize these words results from a training procedure by a specific speaker.



As a rule, user words guarantee high recognition accuracy when they are pronounced by the speaker who trained them.

To add a user word to a context, proceed as follows:

- Choose Add User Word from the Tools menu.  
The Select Context dialog box appears
- In the list box, double-click the context where you want to add a user word.  
The Add User Word dialog box appears.
- Fill in the word in the User word input box and push the Add button.  
The Add user word training dialog box appears.
- Train the word by repeating it three times as requested.
- Accept the word by pushing the button if you are satisfied with the result.  
The trained word appears in the User words list box and the training dialog box is closed.
- Add the word to the context by pushing the Stop button in the Add User Word dialog box.  
You return to the Evaluator main window, where you can test the recognizer on this user word.

### **Notes:**

1. If you are not satisfied with a training utterance, you can supersede it by pushing the Redo button instead of the Accept button
2. If you decide not to add the user word to the context after you have finished training it, push the Cancel button instead of the Stop button.

### **Delete User Word**

To delete a user word from a context, proceed as follows:

- Choose Delete User Word from the Tools menu.  
The Select Context dialog box appears



- In the list box, double-click the context that contains the user word to be deleted.  
The Delete User Word dialog box appears.
- Select the user word you want to delete from the User word drop-down combo box.
- Push the Delete button.  
The word disappears from the list.
- Push the OK button to confirm your action.  
You return to the Evaluator main window.

## **Help Menu**

### **Contents**

Provides a general overview of the comprehensive on-line help system installed with the ASR Evaluator. It contains links to the different items included.

### **Search for Help On**

Provides an on-line help topic search mechanism.

### **How to Use Help**

Provides the Windows general help on the use of on-line help.

### **About**

Provides product information.



## The Evaluator Program Window

### Utterance Field

When you pronounce an utterance from the active context(s), the Evaluator displays the recognition result in the Utterance field.

This can be a word, a sentence, a digit or a digit string, if the recognition was successful or it can be ###, if the recognition engine did not recognize the utterance as a valid phrase.

If Rejection at word level is enabled, the result displayed may consist of a sentence with --- at places where individual words were not recognized.

If the recognition engine is running in Open Microphone mode, and the confidence level of the utterance does not exceed the value for the Acceptance parameter, no result is displayed; instead, the recognition engine automatically starts waiting for a new utterance.

### N-Best alternatives Field

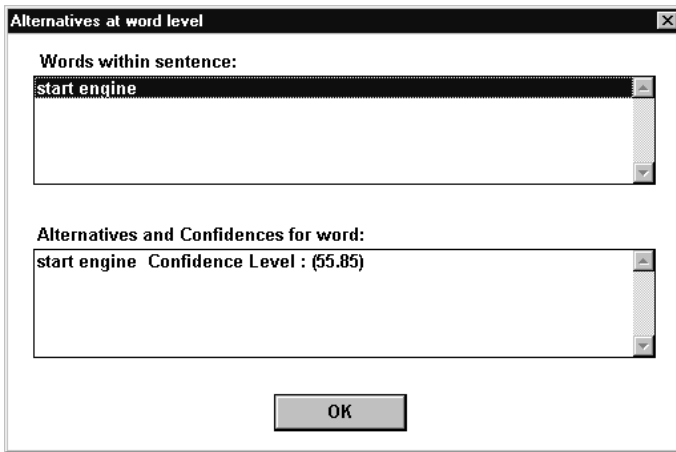
When the Show N-Best is enabled, the size of the Utterances field is reduced and the N-Best alternatives field is added underneath it.

It displays the different recognition alternatives and their confidence values in decreasing order.

If necessary, you can verify all the alternatives and confidence levels for all the words in a sentence, and for all the N-best sentences in a recognition result.

If you double-click one of the sentences in the N-Best result list box, the Alternatives at word level dialog box appears.





This dialog box contains two list boxes. The first list box shows all the words in the selected sentence. When you click one of these words, the second list box displays all the alternatives with their confidence levels.

Depending on the ASR engine, one or more alternatives with associated confidence level are available.

## Recognizer Activity Field

The Recognizer Activity field is the one that is hidden behind the L&H company logo when the recognition engine is not active.

When the recognition engine is activated in Open Microphone mode or Continuous Operation mode, it does not start full recognition until the Voice Activity Detector has detected speech. During that period, the Recognizer Activity field displays Waiting for Speech.

When the recognition engine detects an utterance, you see the message Recognizing.



This field can be very handy to adjust the Speech Sensitivity parameter. If the recognizer is constantly recognizing, you have to increase the Speech Sensitivity parameter; if it is constantly waiting for speech, you have to decrease the Speech Sensitivity parameter.

## **Active Contexts Field**

The Active Contexts field contains a list of simultaneously activated contexts.

## **Active Words Field**

The Active Words field, underneath the Active Contexts, contains a list of active words.

# **Recognizer Management Buttons**

## **Start Button**

By pushing the Start button, you start the recognition engine. Whatever recognition mode you choose, you always have to start the engine by pushing the Start button.

## **Stop Button**

When you push the Stop button, the recognition process ends and a result is returned.



## Break Button

When you push the Break button, the recognition process ends, but no result is returned.

## Recognition Operating Modes

The ASR1500 and ASR1600 recognition engines can operate in four different modes. In the Evaluator main window, you can activate them by clicking the desired option box in the Mode frame. The operating modes are:

- Push to talk
- Automatic stop
- Open microphone
- Continuous Operation

Each of these operating modes has specific fields of application. The sections below explain the essential characteristics of each mode.

### Push to talk

This operating mode is for speech applications that require the control over the *starting* and *finishing* point of the recognition time span.

Typically, applications that use this operating mode are equipped with a push-to-talk (start) and a stop button, which the user of the application has to push when he decides that the recognition is to start or to stop.

Within the context of the ASR Evaluator, you can select this mode for interactive testing, but it is also particularly suited for evaluating the recognition of prerecorded wave files on an engine. In this case, the control of the start and stop buttons is handled by the application.



## Automatic stop

This mode is best suited for speech applications that guide the user through a strict dialog.

For this type of applications, the developer can easily determine the starting point of the incoming speech (e.g. after a question). As to the finishing point of the time window, he is much less certain, because he cannot predict the length of the utterance. Therefore, he can rely on the automatic-stop mode of the engine, which is linked to a number of parameters.

The *reaction-time parameter* indicates to the recognizer how much trailing silence has to be heard before it can decide that the utterance is complete.

The *time-out* parameter sets the maximum length of the time window in which the recognizer is active. Thanks to this parameter, the recognition engine will automatically stop after a certain period even if the end-of-speech detection failed.

When working in Automatic-stop mode, you push the Start button to begin the recognition, but the engine uses end-of-speech detection to stop the recognition when the utterance is complete.

## Open microphone

This operating mode is intended for speech applications requiring control of the *starting* point of the recognition time window, even if it is not predetermined and where the *finishing* point of the time window is uncertain or cannot be determined.

As the starting point of the recognition time window is not predefined, the recognizer starts in a state with low CPU consumption monitoring the input channel. The Voice Activity Detector (a specific speech detection algorithm) and



the characteristics of the speech signal itself allow the engine to determine when real speech is coming in. Only if that is the case, it switches to full recognition.

The parameter on which this mechanism is based, is the *sensitivity* parameter. It establishes the relative energy threshold the input signal must exceed before it is classified as valid speech.

This mode is typically used for applications that incorporate a dialog with so-called *barge-in* functionality. In such a system, rather than having to wait until he is requested to do so, the user can start speaking at any point in the dialog. Thanks to the Open-microphone mode, the engine can determine exactly the starting point of the recognition time window. As to the finishing point, it is handled in the same way as in Automatic-stop mode.

Another possible application of this mode is the *wake-up* functionality. This feature can help to make applications for command-and-control more robust against environmental noise.

In this approach, the recognition engine listens until a particular predefined key word (e.g. "wake up!") is recognized. For every utterance, the confidence levels of the recognition result are compared to the acceptance parameter and only if the confidence level is higher than the value for the acceptance parameter, will the result be accepted and returned to the application. If not, the result is rejected and the recognition engine waits for a new utterance.

When using the Open-microphone mode, you have to push the Start button to begin. The engine, waiting for speech, activates the speech detection mechanism; when an utterance is detected, it switches to full recognition automatically; after the utterance has been recognized, the end-of-speech detection makes the engine stop automatically.



## Continuous Operation

This recognition mode is for applications that require permanent activation of the recognition engine in order to recognize one utterance after the other.

Two good examples of such applications are command-and-control applications that do not require active context changes during operation and dictation systems.

When using the Continuous Operation mode, you have to push the Start button to begin. Speech detection starts, and after speech has been detected and recognized, the engine automatically restarts speech detection. To stop the engine, you have to push the Break button.

## Recognition from File

You can evaluate the recognition performance or test the impact of different parameter settings by submitting wave files to the recognition engine. One way to record these files is by activating the option Save input to file in the Input device configuration dialog box of the Evaluator. For a detailed description of how to use this feature, refer to the section Change Configuration.

Another possibility is to record wave files by means of the Windows Sound Recorder. To record wave files of the required format, change the Recording Settings to Mono, 16 bit, 11 kHz and no compression if using an 11kHz ASR database, or Mono, 16-bit, 8 kHz and no compression if using an 8kHz ASR database.

To test the recognition with the recorded files, select the recognition mode Push to talk; from the Windows Explorer, select the wave files and drag them onto the Utterance field of the Evaluator. The files are opened one after the other and the recognized utterances appear in the Utterance field.



---

---



# THE LEXICON TOOLKIT

## Introducing the Lexicon Toolkit (LexTool)

The LexTool lets you perform a variety of tasks that are essential to ASR programming. These tasks can be the following:

- manage contexts

You can create a context, delete it, import it from a BNF grammar file and/or export it to some target platform format.

- edit the lexicon (vocabulary) of a context

You can add, delete or modify words and their phonetic transcriptions.

- manage users

You can create, delete, rename and copy a user. You can copy contexts from one user to the other. You can register or unregister a user for languages and contexts. Finally, you can export a user to install his/her speech characteristics at a later stage.

- manage phonetic exception dictionaries

The phonetic transcriptions of the user-independent words in the context can be generated by the built-in phonetic expert system or they can be taken from phonetic exception dictionaries. You can create such a phonetic dictionary, delete it, modify it and/or export it to be installed at a later stage with the application you have developed.



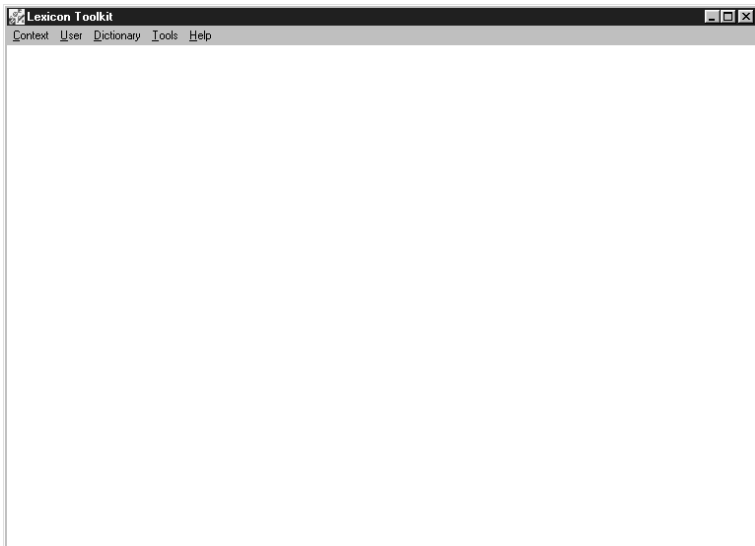
---

---



## Starting the Lexicon Toolkit

Like the ASR Evaluator, the Lexicon Toolkit is another program that is automatically installed during the installation of the ASR1500/ASR1600 SDK. It is one of the items in the program group added to the Windows Programs menu. When you click its name in the list, the program starts up and the main window appears on the screen.



### **Note:**

You can also get to the Lexicon Toolkit through the Tools menu of the ASR Evaluator.



## Menu Commands

### Context menu

An essential component to speech recognition is the recognition *context*. This binary data structure, included in the general database for a particular ASR engine, contains information about the structure and the vocabulary of the speech input in relation to a specific user and a specific language.

Basically, there are two ways to build a context. You can write and compile a BNF grammar or you can use one of the three syntaxes provided with the ASR software package.

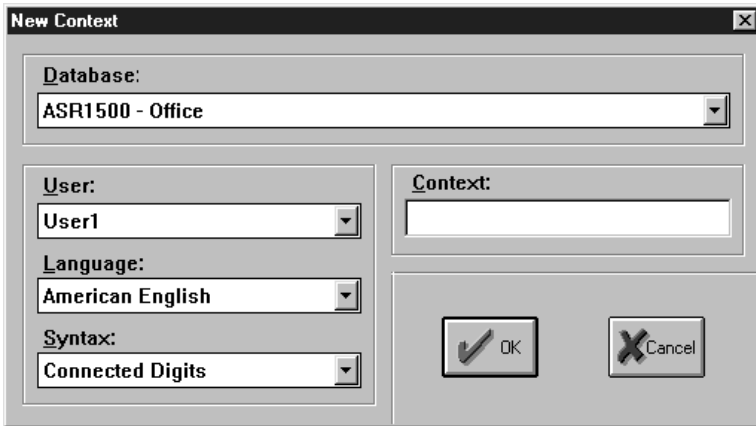
The Context menu contains functions to create, edit and delete contexts.

#### **New**

The function New lets you create a context based one of the three syntaxes.

When you choose this command, the New Context dialog box appears.





The screenshot shows a 'New Context' dialog box. It has a title bar with the text 'New Context' and a close button. The dialog is divided into several sections. The top section has a label 'Database:' followed by a dropdown menu showing 'ASR1500 - Office'. Below this, on the left, are three more dropdown menus: 'User:' showing 'User1', 'Language:' showing 'American English', and 'Syntax:' showing 'Connected Digits'. To the right of these is a text input field labeled 'Context:'. At the bottom right of the dialog are two buttons: 'OK' and 'Cancel'.

The list boxes in the dialog box let you supply the correct specifications for the context to create:

- the ASR database that you want to use to store the context
- the user for whom you want to register the context
- the language for which you want to register the context
- the syntax the context is to be based on: Connected Digits, Isolated Words or Keyword Spotting.

In the Context input box, you enter the name of the new context. You can choose the name freely, including using spaces, as long as no other context with the same name exists.

**Note:**

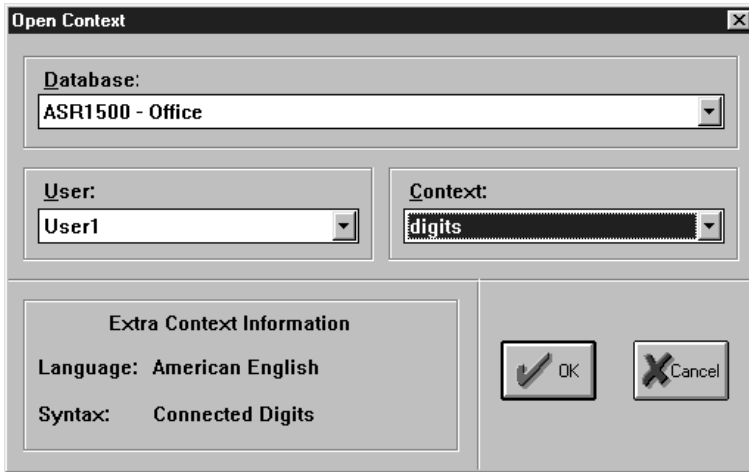
If you want to create a context and the ASR database does not contain users yet, you are prompted to create a user first.

When you click the OK button, a dialog box appears with the database, the user and the context displayed in the caption, and you can immediately start to edit the context, i.e. add words to it.



## Open

This function opens an existing context for editing. When you choose this command, the Open Context dialog box appears.



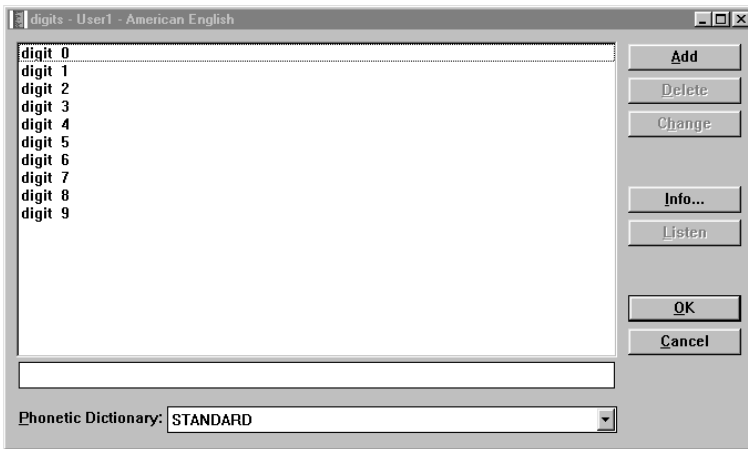
Through the list boxes in the dialog box, you select the Context itself and the Database and the User it is linked to. The Extra Context Information box automatically displays the relevant language and syntax for the selected context.

### **Note:**

The reason why you have to supply more than the mere name of the context is because there can be several contexts with the same name, but residing in different ASR databases, or registered to multiple users in a given database.

When you click the OK button, a dialog box appears with the context name, the user and the database displayed in the caption.





Editing a context can imply several tasks: adding one or more words, deleting one or more words and/or changing phonetic expressions.

In the context editing dialog box, all the words that are in the vocabulary of the context are listed. Each line of this list consists of a class name and a word. When you select a word, the phonetic transcription appears in the field underneath. In case there are several phonetic transcriptions, they are separated by semicolons. You can not change the phonetic transcription in this edit box.

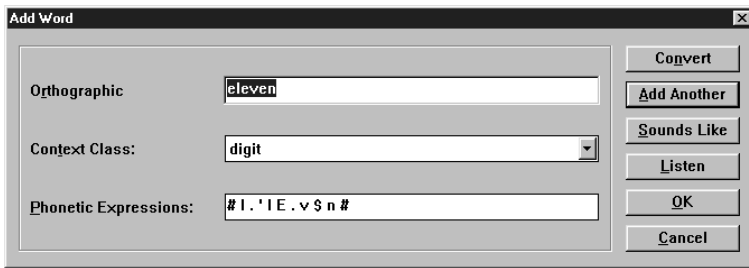
The Phonetic Dictionary field indicates the phonetic exception dictionary that is used when adding new words to the context.

If an opened context cannot be changed, the buttons Add, Delete and Change are disabled.

### **Add**

When you push the Add button, the Add Word dialog box appears.





To add a word to the context, supply its orthographic expression, choose a context class and type in the valid phonetic expression(s), separated by vertical bar(s).

If you want the system to generate the phonetic transcription automatically, use the Convert button. In that case, the transcriptions are retrieved from an exception dictionary or they are generated by a built-in phonetic expert system, the grapheme-to-phoneme system. If a word is found in the selected dictionary, the phonetic transcriptions associated with it always prevail.

If you want to add several words in a row, you can do so by using the Add Another button.

When you are finished, you click the OK button to confirm all the additions. When you click the Cancel button, only the latest addition is rejected.

Using the Sounds Like button, you can enter some other word the pronunciation of which has to be used for recognizing the current word.

When clicking the Listen button, you will hear the pronunciation of the specific word.

**Note:**

You can add numerous words quickly, taking advantage of the Convert functionality in the process, by using solely the “Enter” key on your keyboard. The first “Enter” does the conversion, while the second one



adds the word and sets you up for the next word to be added.

### **Delete**

If you want to remove a word from the context, you select it and click the Delete button. The currently selected line in the list box will be deleted.

### **Note:**

When a context is based on the Connected-digits syntax, the digits 0 through 9 are generated by the system itself. These so-called system words cannot be deleted.

### **Change**

To change the phonetic transcription of a word, you have to use the Change button or double-click a line in the list box. The Change Word dialog box appears.

This dialog box is almost the same as the Add Word dialog box, except that the orthographic expression can not be altered. This is the only way to change the phonetic transcription of a word.

### **Info**

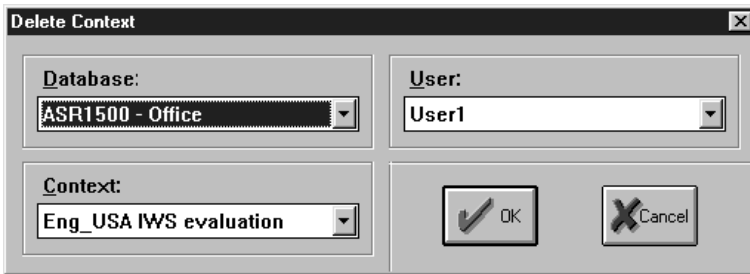
When you press the Info button in an open context, a dialog box pops up showing more information about this context.

### **Delete**

Choose this function to delete an existing context.

The Delete Context dialog box appears.





Specify the necessary information about the context that you want to delete and click the OK button.

### Import

The function Import compiles a grammar file and turns it into a context.

When you choose the Import command, the Import Context dialog box appears.



Before you compile the grammar, you have to supply some of the same information as for the New command:

- the ASR database you want to use
- the user for whom you want to register the context
- the language for which you want to register the context



- the phonetic exception dictionary from which you want the transcriptions to be retrieved

In addition, you have to supply the name of the grammar file and the format it has been written in.

The grammar file should be written in a format that is currently available. Currently the only available grammar file format is BNF Grammar Compiler.

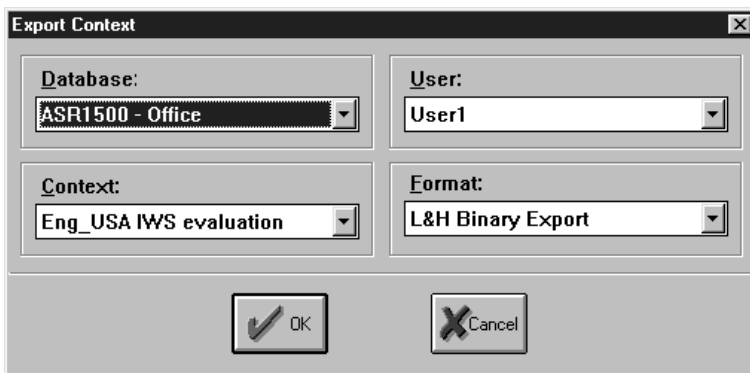
Enter the name of the new context in the appropriate input box.

When you click the OK button, the grammar file is compiled and the dialog box closes. A message box will inform you when an error occurs.

### Export

Use this function to convert the context into a format that can be used on some target platform.

When you choose this command from the Context menu, the Export Context dialog box appears.



Again, you have to supply the general information about the context concerned: database, user and context name.

In addition, you have to indicate the format into which you want to convert the context. Among the available export



formats is the SDK export format. It allows you to produce contexts that can be redistributed with the application you have developed.

When you click the OK button, you have to supply some file names. The exact number of file names depends on the selected export format. Each file name can be preceded by a drive letter and/or path specification.

When the context is exported successfully, the dialog box closes.

### **Exit**

With this function, you leave the LexTool application.

## **User menu**

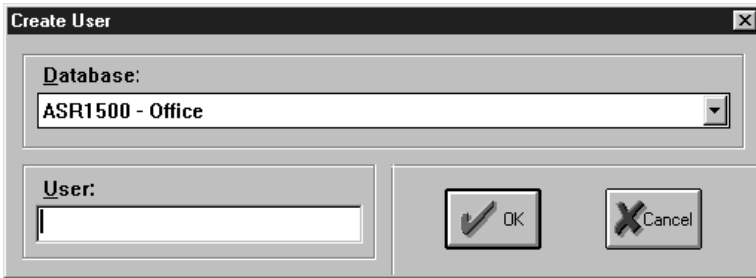
Another vital concept within the context of L&H speech recognition is the *user*. It refers to a data structure that contains all the information related to the speaker. On the one hand, it holds information for speaker-independent speech recognition in a particular language; on the other hand, it allows you to store information for speaker-dependent recognition of trained user words.

### **Create**

Use this function to create a new user

When you choose the command Create, the Create User dialog box appears.





First, you specify the ASR Database that you want to use to store the information.

Enter the name of the user in the input box. If the name already exists, you obtain an error message.

After creation of a new user, you are prompted to register this user.

### Delete

Use this function to delete an existing user.

When you choose this command, the Delete User dialog box appears.



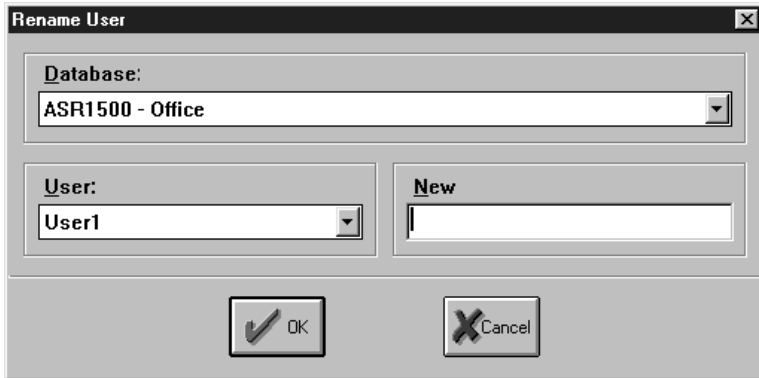
Choose the user that has to be deleted and click the OK button.

### Rename

Use this function to rename an existing user.



When you choose this command, the Rename User dialog box appears.



The 'Rename User' dialog box has a title bar with a close button. It contains three main sections: a 'Database:' section with a dropdown menu showing 'ASR1500 - Office'; a 'User:' section with a dropdown menu showing 'User1'; and a 'New' section with an empty text input field. At the bottom, there are two buttons: 'OK' with a checkmark icon and 'Cancel' with an 'X' icon.

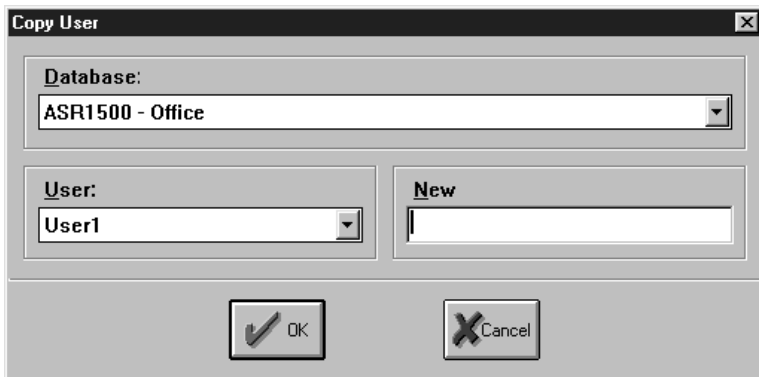
Specify the current name of the user through the User list box. Type the new name of the user in the New input field. Click OK to proceed.

A user can not be renamed to an already existing user.

### Copy

Use this command to create a new user by copying the data from an existing user.

When you choose this command, the Copy User dialog box appears.



The 'Copy User' dialog box has a title bar with a close button. It contains three main sections: a 'Database:' section with a dropdown menu showing 'ASR1500 - Office'; a 'User:' section with a dropdown menu showing 'User1'; and a 'New' section with an empty text input field. At the bottom, there are two buttons: 'OK' with a checkmark icon and 'Cancel' with an 'X' icon.



Enter the name of the new user in the appropriate input box.

The name of the new user must be different from all existing users.

### **Copy Context**

This command allows you to copy the context registered for another user to the one that is currently selected.

When you choose Copy Context from the Context menu, the Copy User Context appears.

A screenshot of a Windows-style dialog box titled "Copy User Context". The dialog has a standard title bar with a close button. It contains four input fields with dropdown arrows: "Database:" (showing "ASR1500 - Office"), "User:" (showing "User1"), "Context:" (showing "Eng\_USA IWS evaluation"), and "To" (showing "User1"). There is also a "New Context:" field which is currently empty. At the bottom, there are two buttons: "OK" with a checkmark icon and "Cancel" with an 'X' icon.

<b>Database:</b> ASR1500 - Office	
<b>User:</b> User1	<b>Context:</b> Eng_USA IWS evaluation
<b>To</b> User1	<b>New Context:</b> 
OK Cancel	

Enter the context name for the target user in the New Context input box.

### **Notes:**

1. A context can not be copied to a context with an existing name.
2. The target user must be registered for the language of the source context to complete this action successfully.



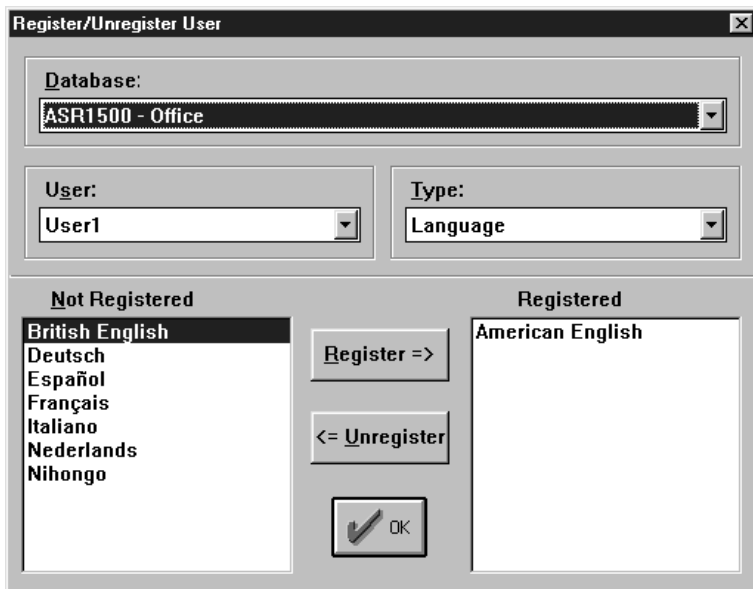
## Register/Unregister

With this function, you can register and unregister languages and contexts for a certain user.

Registering a *language* for a user implies that this user will be able to use that language. Unregistering a language means that he will not be able to use that language.

Registering a *context* for a user means that this user gets a private copy of that system context. Unregistering a context for a user means deleting that context for this user.

When you choose this command, the Register/Unregister dialog box appears.



The dialog box is titled "Register/Unregister User". It contains the following elements:

- Database:** A list box showing "ASR1500 - Office".
- User:** A list box showing "User1".
- Type:** A list box showing "Language".
- Not Registered:** A list box containing "British English", "Deutsch", "Español", "Français", "Italiano", "Nederlands", and "Nihongo".
- Registered:** A list box containing "American English".
- Buttons:** "Register =>", "<= Unregister", and "OK" (with a checkmark icon).

Through the list boxes, you indicate the database and the user.

In the Type list box, you specify whether you want to register/unregister a language or a context. The list boxes at the bottom reflect the situation of registered and



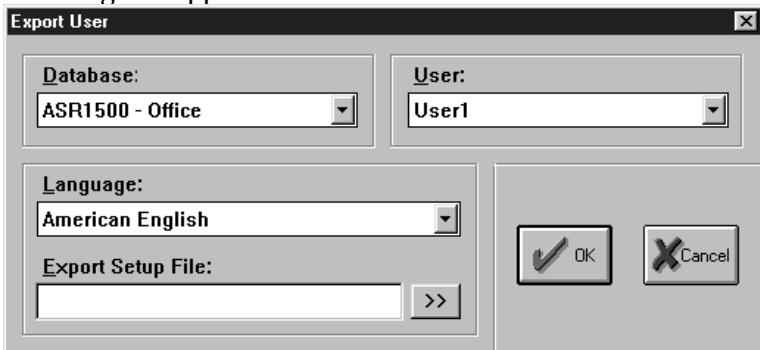
unregistered items. By means of the Register and Unregister buttons you can adjust the situation.

The OK button is only used to close the dialog box. The registering and unregistering take place immediately.

## Export

With this command, you can export a user and his speech characteristics in order to install it at a later stage.

When you choose the Export command, the Export User dialog box appears.



Next to the database and the user, you have to type in the name of the export setup file or locate and select it via the browse button (“>>”).

When you click the OK button, the user is exported and the dialog box closes.

## Dictionary menu

### New

The function New in the Dictionary Menu creates a new exception dictionary, which can be edited immediately.



When you choose this command, the New Dictionary dialog box appears.



Enter the name of the new exception dictionary. A dictionary name already used is not accepted.

When you click the OK button, a dictionary editing dialog box appears. The initial list box in this dialog box is always empty. Clicking the OK button saves all additional changes after creating the new dictionary. Clicking the Cancel button discards all additional changes after creating the new dictionary.

## Open

Use this function to open an existing exception dictionary for editing.

When you choose this function, the Open Dictionary dialog box appears.





Editing a dictionary can consist in any of the following: adding one or more words, deleting a word, changing a phonetic expression.

Clicking the OK button opens a dictionary editing dialog box with the dictionary name and the language name of the caption. This dialog box contains a list box with words. Each line of this list box contains a word. In the edit box underneath, you find the phonetic transcription.

**Notes:**

1. You can not change the phonetic transcription in this edit box.
2. When an opened dictionary cannot be changed, the editing buttons Add, Delete and Change are disabled

When clicking the Add button, the Add Word dialog box appears. Adding words to an exception dictionary is very similar to adding to a context. You can fill in an orthographic expression (a word) and you can type in a number of phonetic expressions separated by a vertical bar.

Use the Convert button to generate the phonetic transcription automatically. These transcriptions are generated using the grapheme-to-phoneme phonetic expert system and exception dictionaries.

Using the Sounds Like button, you can enter some other word, and use its pronunciation for recognizing the current word.

When clicking the Listen button, you will hear the word.

When you add a word, you can immediately add another word by clicking the Add Another button.

With the OK button, all additions are confirmed and put into the current open dictionary. With the Cancel button the



possible current addition is discarded, but all previous additions are put into the current open dictionary.

When clicking the Delete button, the currently selected line in the list box is deleted.

When clicking the Change button or double-clicking a line in the list box, a new change word dialog box opens. This dialog box resembles the add word dialog box.

### Delete

The function Delete deletes an existing exception dictionary. When you choose the Delete command, the Delete Dictionary dialog box appears.



### **Note:**

The dictionary named STANDARD can not be deleted. This is the system dictionary for the selected language.

### Rename

The function Rename renames an existing exception dictionary.

When you choose this command, the Rename Dictionary dialog box appears.





Select the Database, the Language and the dictionary; supply its new name and click the OK button.

The dictionary name STANDARD cannot be renamed.

### Copy

Use the function Copy to create a new dictionary by copying an existing one. This copies an exception dictionary to a new exception dictionary.

When you choose the Copy command, the Copy Dictionary dialog box appears.





Fill in the name of the new dictionary in the appropriate input box.

A dictionary cannot be copied to an already existing dictionary.

## Export

The function Export, in the Dictionary Menu exports an exception dictionary to be installed later.

A dialog box with the database, the language and the dictionary appears after selecting Export.



The Export setup file has to be filled in, or can be chosen via the ">>" button.

When you click the OK button, the dictionary is exported and the dialog box closes.

## Help menu

### Contents

Provides an on-line help general overview, with links to all details.



### **Search for Help On**

Provides an on-line help topic search mechanism.

### **How to Use Help**

Provides help on the use of on-line help.

### **About**

Provides product information.



---

---



# THE BNF GRAMMAR LANGUAGE

## Introducing the L&H BNF Grammar Language

This chapter describes the L&H BNF grammar language used to work out continuous speech recognition grammars.

No matter how diverse the applications can be, the phrases input to a continuous speech system, as a rule, are based on a structure of some kind, called a context. This structure can range from highly complex to as simple as a sequence of digits. The L&H BNF grammar language constitutes a powerful tool for specifying the structure of the input to a continuous speech recognizer.

When the grammar is complete, the BNF grammar compiler turns this specification into a context that can be handled by the recognition software.

The core of the L&H BNF grammar language is a collection of grammar *rules*. Each rule describes an allowable structure and gives it a name. An example of a grammar rule might be:

`<date> : the <day> of <month> <year> ;`

Here `<date>`, `<day>`, `<month>` and `<year>` represent structures of interest for the input language of the continuous speech recognizer; `<day>`, `<month>` and `<year>` are defined by other rules elsewhere. With the proper definitions, the input phrase, “*the fourth of July 1776*” will match the above rule.

For historical reasons, a structure that is handled by the recognition engine itself is called a *terminal symbol*, while a structure that is specified further by other rules in the



grammar is called a *non-terminal symbol*. To avoid confusion, terminal symbols are often referred to as *tokens* or *words* (even though they do not have to be limited to one word!).

## Grammar essentials

### Preliminary

A full-fledged BNF grammar consists of different logical sections: the pre-compiler section, the identification section, the interface section, the class section, the pronounce section and the rules section.

Rather than describing these different sections sequentially and in a theoretical way, the first part of this chapter will make abstraction of these. Starting from a very simple basic grammar, it will gradually introduce new concepts, to end with a complete and formal specification of the L&H BNF grammar.

### Notation

The L&H BNF grammar specification uses a C-like syntax:

- All 'expressions' are terminated by a semi-colon (;)
- Comments can be inserted by using the C (*/\* \*/*) or the C++ (*//*) comment conventions.
- Parentheses (*()*) can be used to group expressions together.



## Statements, Rules and Symbols

Imagine a bar, in the not so far future, operated by a computer. Naturally, the computer is to recognize the customers' orders.

A grammar describing such a situation could be the one below:

*!grammar Drinks;*

*!language "American English";*

*!export <Speech>;*

*<Speech>: lemonade / milkshake / orange juice;*

This grammar should not be too hard to understand, and it is probably obvious that we could either order a lemonade, a milkshake or an orange juice. Nevertheless some further explanation is warranted.

You will probably have noticed that the first three lines of our grammar start with an exclamation mark (!) followed by a word. These words are *reserved words*, i.e. words that have a special meaning to the grammar compiler. From now on we will call a line that start with a reserved word a *statement*. All other lines (except for the comment lines) will be called *rules*. So, we could say that the above grammar contains three statements and one rule.

This grammar also contains *symbols*. Symbols can be any combination of a subset of ASCII-characters (only the printable characters, and excluding the "larger than"-symbol (">") as well) enclosed by angular brackets (< >). Symbols are placeholders for rules that are defined elsewhere. In the above sample grammar, only one symbol can be recognized: <Speech>.

Let us now take a closer look at the first statement. This kind of statement is called a *grammar statement*. It contains the reserved word *!grammar*, followed by a *word* and is



terminated by a semi-colon. The *word* can either be a single word (like Drinks) or a string enclosed by double quotes (e.g. "The Recognition Bar").

Grammar statements are optional and only used for identifying the grammar. The use of grammar statements has no consequence on the behavior of the derived context.

The second statement is called a *language statement*. The reserved word *!language* is followed by a language name and a semi-colon. If the language contains two or more words, it should be enclosed by double quotes. For instance: "American English".

The language statement is optional and restricts the use of the grammar to a specific language. If a grammar contains the language statement "*!language* French;" it can only be compiled in a context for the French language. If we do not use the language statement, the same grammar can, for instance, be used for both "British English" and "American English".

**Note:**

If we want to import your grammar in the LexTool, and we use the language statement in your grammar, the language specified in the grammar should correspond to the language specified in the LexTool.

The third statement is an *export statement*. Export statements start with the reserved word *!export*, followed by a *symbol* and a semi-colon. Export statements provide entry points into the grammar. Thus, they define what can be recognized and what not. In this case, it is the rule identified by <Speech>.

This leaves us with the final part of our grammar: the *rule*. A rule is made up of a symbol, followed by a colon followed by the symbol *definition* terminated by a semi-colon. In this grammar, the definition consists of a disjunction of phrases, separated by the "or" operator ('|'). It indicates that the



symbol <Speech> is defined as “lemonade” **or** “milkshake” **or** “orange juice”.

Since the export statement defines speech as ‘that which is defined by <Speech>’, the conclusion is that only these three drinks can be recognized.

### **Notes:**

1. To be able to use a context all symbols have to be defined!
2. A space is not a delimiter for phrases. We cannot order an “orange” nor can we order a “juice”. What we can order is an “orange juice”.

## **Classes**

In the previous section we have defined our first grammar, allowing our bar to go into business. However, only a limited set of drinks is available. Even if we could always add drinks to our grammar, this would imply that we have to recompile the grammar whenever we decide to add a drink to the menu. The following grammar illustrates how we can bypass this problem:

```
!grammar "Class Drinks";
```

```
!language "American English";
```

```
!export <Speech>;
```

```
!class <Speech>;
```

```
<Speech>: lemonade | milkshake | orange juice;
```

At first glance, there is not much difference between this grammar and the one we have defined in the previous section. There is only one extra statement, identified by the reserved word *!class* followed by a symbol and a semi-colon. This statement is therefore called a *class statement*. If a class-statement for a symbol exists, we call that symbol a *class*.



The class statement allows us to modify the context at runtime: we can add and delete words (using the `asrCtxAddWordEx()`, `asrCtxAddUserWordEx()` or `asrCtxDeleteWord()`) after the grammar has been compiled into a context. Thus, we can update the contents of a context without having to close, recompile and reopen it.

**Note:**

Words can only be added and removed from classes, *not* regular symbols.

In order to be able to add or remove a word, the context has to be opened with the `ASRCTXATTRIB_WRITE` attribute or it has to be a temporary context. For detailed information on these issues, refer to the ASRAPI functions `asrCtxOpen()`, `asrCtxOpenTemp()` or `asrCtxImportBuf()` in the *Function Reference* manual, supplied with the SDK.

If we do not add any content (words, phrases) to a context (by changing the last line into “<Speech>;”), your context becomes *generic*. That is the structure that can be used for any language.

Classes, symbols and even words not belonging to a symbol or a class can be mixed together into a single grammar, as the below sample demonstrates:

```
!grammar "Class Drinks revised";
```

```
!language "American English";
```

```
!export <Order>;
```

```
!class <Drink>;
```

```
<Order>: <Intro><Drink> please;
```

```
<Drink>: a lemonade / a milkshake / an orange juice;
```

```
<Intro>: I would like / Can I have;
```



The possible recognition results for this grammar are:

*Can I have an orange juice please.*

*Can I have a lemonade please.*

*Can I have a milkshake please.*

*I would like an orange juice please.*

*I would like a lemonade please.*

*I would like a milkshake please.*

## Repetitious Recognition

It is not unlikely that a customer would like to order multiple drinks. To accomplish this, we could reuse the context as we built it before, and run it multiple times. However, running the same context multiple times results in multiple recognitions, which would make it impossible to keep the different orders separated from each other. Therefore, we want to recognize multiple orders within the same recognition. This could be as done as follows:

```
!grammar "3 Drinks";
```

```
!language "American English";
```

```
!export <Speech>;
```

```
<Speech>: <Drink><Drink><Drink>;
```

```
<Drink>: lemonade / milkshake / orange juice;
```

Although this works fine, there is a major drawback: the above grammar forces the customers to always order three drinks. The following grammar could help us to overcome this problem:

```
!grammar "1 to 3 Drinks";
```

```
!language "American English";
```



```
!export <Speech>;
```

```
<Speech>: <Drink> / <Drink><Drink> /  
<Drink><Drink><Drink>;
```

```
<Drink>: lemonade / milkshake / orange juice;
```

This is a good solution, but it would again give rise to problems if we wanted a range of up to 10 orders. We could write down all possible combinations, but this is a lot of work. A better way is demonstrated in the grammar below:

```
!grammar "1 to 10 Drinks Revised";
```

```
!language "American English";
```

```
!export <Speech>;
```

```
<Speech>: !repeat(<Drink>,1, 10);
```

```
<Drink>: lemonade / milkshake / orange juice;
```

We have just introduced a new reserved word: *!repeat(arg1, arg2, arg3)*. The first argument (arg1) is the expression to repeat, arg2 is the minimum number of iterations and arg3 is the maximum number.

The above grammar therefore models a situation where a customer can order up to ten drinks.

### **Note:**

The third argument can also be an asterisk (\*). This symbol is used to specify an undefined (unlimited) number of repetitions. However, if we know the maximum number of possible iterations, it is better to specify this rather than using the asterisk. There is no use in trying to recognize n+1 iterations, if the maximum number to be recognized will never exceed n.

Of course, the maximum number of repetitions should be larger or equal to the minimum number of repetitions.



0 is a valid value for the minimum number of repetitions.

The next grammar is a bit more complex, as it uses all the structures we have discussed so far. It lets the user say the number of each drink she/he would like, instead of forcing her/him to repeat the drink itself.

*!grammar "A lot of Drinks";*

*!language "American English";*

*!export <Speech>;*

*!class <Drink>;*

*!class <Drinks>;*

*<Speech>: !repeat(<Order>, 1, 3) please;*

*<Order>: <Single Order> / <Multiple Order>;*

*<Single Order>: (one / a) <Drink>;*

*<Multiple Order>: <Count> <Drinks>;*

*<Count>: 2 / 3 / 4 / 5;*

*<Drink>: lemonade / milkshake / orange juice;*

*<Drinks>: lemonades / milkshakes / orange juices;*

Valid orders would include:

*a lemonade please*

*a lemonade one orange juice please*

*3 milkshakes 5 lemonades one orange juice please*

But also the more unlikely:

*one orange juice a orange juice please*

*5 milkshakes 2 milkshakes one milkshake please*

*4 orange juices, one milkshake, one orange juice please*



The following are not valid:

*4 orange juice*

*a orange juice a milkshake a lemonade a lemonade please*

**Note:**

This grammar models orders that are very unlikely to be said. As long as there is a chance they could be ordered, they should be modeled, otherwise it should not be possible to recognize them. In such cases, this grammar would not be a good grammar, but for now and as a demonstration, it serves well.

The only other alternative may be to completely write out all possible utterances. This may be laborious, but gives us the best results!

## Optional Recognition

In some of the above grammars we have included “please” and in others we have not. Since not all people use this word, there is a problem. Should we add it at the expense of poorer recognition for people who do not use it, or should we leave it out, thus penalizing people who do use it?

The obvious answer to this question is to allow both. However, just as we did with the repetitions, we do not completely write down all possible recognitions, but rather use a shortcut. This is demonstrated below:

*!grammar “Possibly Polite Drinks”;*

*!language “American English”;*

*!export <Order>;*

*<Order>: <Drink> !optional(please);*

*<Drink>: a lemonade / a milkshake / an orange juice;*



The above grammar has the same result as the one below:

*!grammar "Possible Polite Drinks Revised";*

*!language "American English";*

*!export <Order>;*

*<Order>: <Drink> / <Drink> please;*

*<Drink>: a lemonade / a milkshake / an orange juice;*

It may seem a bit useless to introduce a whole new concept to the grammar-specification language, when such a trivial workaround is available. In complex grammars, however, we may be glad to have this option at our disposal.

### **Note:**

Note that "optional( x )" has the same effect as "( x | )".

## **Unnotified Recognition**

The word "please" in the above example does not have any meaning to our computer. Indeed, it does not affect the client's order. Consequently, there is no need to notify our system of it, when speech is recognized. This can be accomplished by using the *!ignore(expression)* function. Its use is very similar to that of *!optional(expression)*:

*!grammar "Ignore Politeness In Drinks";*

*!language "American English";*

*!export <Order>;*

*<Order>: <Drink> !ignore(please);*

*<Drink>: a lemonade / a milkshake / an orange juice;*



**Note:**

Notice the difference between *!optional()* and *!ignore()*: *!optional* does not require the user to actually say the word, but if he does, the system is notified of it; *!ignore()* requires the word to be spoken, but the system is not notified of it. The table below illustrates this. We will use the above grammars as samples.

Possible Utterances	Possible Recognitions	
	<i>!optional(please)</i>	<i>!ignore(please)</i>
a lemonade	a lemonade	###
a milkshake	a milkshake	###
an orange juice	an orange juice	###
a lemonade please	a lemonade please	a lemonade
a milkshake please	a milkshake please	a milkshake
an orange juice please	an orange juice please	an orange juice

**Note:**

*!ignore()* can only be used with single words *not* strings nor symbols (classes).

## Non-Recognitions

Closely related to the *!ignore()* function is the *!action()* function. As a matter of fact, they are each other's opposites. The function *!action* will return a word/phrase even though it has not been (and may not be) spoken. To see how this function is used, take a look at the grammar below:

*!grammar "Action Linked Drinks";*



*!language "American English";*

*!export <Order>;*

*<Order>: a lemonade !action("2 \$") / a milkshake !action("4 \$") / an orange juice !action("3 \$");*

It may be important to first point out that it is possible to retrieve only the result, the result and the associated action or the associated action only. You will have noticed that in the above grammar we have included actions containing the price for each drink. This facilitates the calculation of the bill, as the charged amount for each drink is returned together with the recognized drink itself. So, we do not have to look it up in a table.

**Note:**

1. `!actions()` can also be used to make your application language-independent. Without actions, we have to create a new context for each localized version of your application, as words differ from language to language. When we add the same actions to the corresponding words in each language, our application can be language-independent, if it processes the actions rather than the words themselves. The application does not have to know in which language a command was uttered. For instance:

*!action("STOP\_COMMAND") stop;*

*versus*

*!action("STOP\_COMMAND") arrête;*

2. A third use of actions can be to process similar words in a uniform matter. For instance all of the words below have the same meaning, and therefore it is logical to process them in the same manner:

*!action("START\_COMMAND") begin /*



```
!action("START_COMMAND") start |  
!action("START_COMMAND") initiate;
```

## Importing

Previously, we have stated that each symbol has to be defined, in order to be able to use the context. More precisely, all symbols have to be defined prior to activating the context on an engine. Practically this means that all symbols should be defined in the grammar in order for it to compile.

There is one exception though: the *importing* of symbols. By importing a symbol, we tell the compiler that the definition of that particular symbol will be imported from another context. To use a context containing imported symbols, it has to be linked with the context *exporting* that symbol prior to activating it on the engine. See also the Function Reference manual for explanations on the following ASRAPI functions: `asrCtxLibrary()` and `asrCtxLink()` or `asrLib...()` and `asrRecCtxActivateEx()`

Here is an example of how we can use this feature:

```
!grammar "Food stuff";  
  
!language "American English";  
  
!export <Food>;  
  
<Food>; pizza | hamburger | hot dog;  
  
  
!grammar "Food & Drink";  
  
!language "American English";
```



*!export <Order>;*

*!import <Food>;*

*<Order>: <Food> / <Drink>;*

*<Drink>: a lemonade / a milkshake / an orange juice;*

The first grammar models a restaurant that serves hamburgers, hot dogs and pizzas. In the second grammar we reuse this model in our bar-grammar. Our bar can now serve food as well.

### **Note:**

Unlike the first grammar, which can be used as a standalone, the second grammar needs to be linked to the other one.

Importing can be useful when user-dependent words are used. The structure can be a general context importing all user-trained words from other contexts. As this is done at runtime, it is possible for several users to share the same general context, each with their own context for trained words.

## **Any Speech**

In our examples, we have modeled different ways to take an order. But sometimes we do not know what to expect exactly. We only know that our customer will order a drink from our menu, but we have no idea on how he is going to phrase this.

In such cases, it would be more efficient to check only for specific words in a larger phrase. For doing so, we need the *garbage model* ('<...>'). The garbage model models any speech.

Below you can see an example of its use:



*/grammar "Drink Spotting";*

*/language "American English";*

*/export <Order>;*

*<Order>: <...><Drink><...>;*

*<Drink>: lemonade / milkshake / orange juice;*

Following the above grammar, the customer can say anything. As long as the utterance contains the name of one of the drinks, it is recognized. Yet, this grammar will recognize only the first drink that is named in an utterance.

**Note:**

The use of the garbage model should be limited! For more information see also the description of the *garbage penalty* parameter in the first chapter of this manual, the ASR Evaluator or refer to the section on the engine parameters in the *Getting Started* manual, supplied with the ASR SDK.

## Pronunciations

Imagine we want to expand our list of drinks with coffee. This is a bit of a problem, because there are three different pronunciations for this word: a British English one ("kɒfi" or in L&H+ format: " 'kA+.fi ") and two American English ones(" 'kɒfi " or in L&H+ format " 'kO.fi " and " 'kɑfi " or in L&H+ format " 'kA.fi "). Normally, there should not be a problem because the phonetic expert system, the Grapheme-to-Phoneme module, together with the standard dictionary will automatically generate the most commonly used transcriptions for a word. (For instance 0 will be transcribed as "zero" and "oh".) But for coffee (and for some other words) this is not the case. Only the most general transcription will



be used. If we want to use the alternative or maybe both pronunciations, the sample grammar below provides a solution:

```
!grammar "Pronounced Drinks";
```

```
!language "American English";
```

```
!export <Drink>;
```

```
<Drink>: lemonade | milkshake | orange juice | coffee;
```

```
!pronounce L&H coffee "([T=\ "Pm(L&H+)\ "]#( 'kO.fi ) | ('kA.fi)#)";
```

### **Note:**

We can not use the transcription “ ‘kA+fi ” as the phoneme “A+” is not supported in American English, but only in British English.

The *pronounce statement* in the above grammar tells the engine what phonetic transcription(s) have to be used. This statement is a bit complicated. Therefore, some more explanation may be appropriate. First there is the reserved word *!pronounce*. It should always be followed by the *vendor mark* (“L&H”). The second argument is the word for which you wish to provide the transcription. The third argument is the transcription itself.

### **Note:**

Use the or-operator (“|”) to enumerate alternative transcriptions. Use the back-slash (“\”) operator to define the succeeding quote as part of the string.

The actual transcription is preceded by a header, and enclosed in parentheses and quotes.

As to the headers, currently two different types of headers are used: phoneme-based and allophone-based.



The first one uses the [T="Pm(L&H+)"] header, the second the [T="A(L&H+)"].

All languages currently available, except for *Japanese*, use the phoneme-based transcription and hence the [T="Pm(L&H+)"] header. Japanese uses the allophone-based transcriptions (the [T="A(L&H+)"] header.)

## Spelling

The last feature to discuss is *spelling*.

The recognition engine allows you to spell words using the *!spelling* feature. Although you could treat each letter as a separate word, this would yield very poor recognition results. Indeed, letters are very small words with a great resemblance and they are therefore very hard to recognize. Even humans have problems differentiating between for instance "m" and "n".

This feature simply tells the engine that the specified words have to be spelled. It is important to realize that it is necessary to know these words in advance, i.e. prior to the recognition. It is *not possible to recognize unknown words*, i.e. words that are not in the context, just by spelling them.

This is an example:

```
!grammar "Spelled Drinks";  
!language "American English";  
!export <Drink>;  
!class <Drink> !spelling <spell-syntax>;  
<Drink>: lemonade / milkshake / "orange juice" / coffee;  
<spell-syntax>:!repeat(<letters>, 1, *);  
<letters>: l / e / m / o / n / a / d / i / k / s / h / r / g / j / u / c / f;
```



The above grammar specifies that words belonging to the class <Drink> have to be recognized and that these words are structured as specified in the <spell-syntax>. This syntax specifies that the words are concatenations of the <letters> enumerated. In simpler terms, the words in <Drink> have to be spelled using the letters in <letters>.

**Notes:**

1. The symbol containing the spelled words has to be a class.
2. The characters making up the spelling syntax can only consist of one symbol.
3. Although adding only those characters that make up the words suffices, this imposes limitations on the words that can be added: they have to be made up of the same letters. To avoid this problem, you can define the symbol that contains the letters as a class, which allows you to add letters at runtime.
4. If you want to spell a string it has to be enclosed in double quotes.
5. It is not possible to mix up normal (continuous) recognition with spelling! (For instance it is not possible to recognize something like "I have a c a t".)

## Formal Description

In this section, the various features introduced above will be discussed, in a more formal and structural way.



## Pre-processing Grammars

A C-style preprocessor is used for compiling BNF grammar files.

Currently only the following preprocessor statement is supported:

**#include filename**

filename is a valid file name enclosed in quotation marks.

Example:

```
# include "numbers.bnf"
```

```
!grammar Stocks;  
!language "American English";  
!export <speech>;
```

```
<speech> : <buy command> / <sell command>;
```

```
<buy command> : !optional(please) buy !optional (<number>  
!optional(shares) / share) of !optional(the)  
<company name>;
```

```
<sell command> : !optional(please) sell !optional (<number>  
!optional(shares) / share) of !optional(the)  
<company name>;
```

## Comment Notation

C/C++-like comments `/* */` and `/**` can be used in BNF grammar files.



## Identification Section

The identification section consists of two optional statements where you can specify the grammar name and the language used. The language to be used must always be specified when starting the grammar compiler. If it is also specified in the file, both languages must match.

### Example

```
!grammar German_Grammar;  
!language Deutsch;
```

### **!grammar statement**

#### **Format**

**!grammar** word ;

#### **Description**

The **!grammar** statement specifies the name of the grammar.

#### **Example**

```
!grammar TestGrammar;
```

### **!language statement**

#### **Format**

**!language** word ;

#### **Description**

The **!language** statement specifies the language of the grammar.

#### **Example**

```
!language "American English";
```

#### **Comments**

This is an optional statement: the language will be specified anyway as a parameter to the grammar compiler.



## Interface Section

The interface section consists of a number of export and import statements that specify which symbols are available to other grammars and the recognizer software, and which symbols are used from other grammars.

With the `!export` statement, you can specify the non-terminal symbols to be exported, so that they can be used by other grammars. Exported symbols also serve as the top-level entry point for the compiler.

With the `!import` statement you can declare the non-terminal symbols, which are used in the grammar but defined and exported in other grammar files. All the separately compiled grammars (contexts) have to be linked, so that the imported non-terminal symbols are resolved.

### Example

```
!import <Digit> ;  
!export <Number> ;
```

The non-terminal symbol `<Digit>` is defined in another grammar file and is imported here. The non-terminal symbol `<Number>` is defined in this grammar file and is exported, so that it can be used in other grammar files. The syntax starting with the non-terminal symbol `<Number>` can be activated for recognition.

### **!export statement**

#### **Format**

**!export** symbol;

#### **Description**

The `!export` statement specifies the top rule non-terminal symbols in the BNF grammar. It also you allows to export frequently used symbols for use by other contexts. Other contexts can import the symbols by linking the contexts.

#### **Example**



```
!export <Number>;  
!export <Integer>;
```

**Comments**

At least one `!export` statement should always be used, to indicate at least one the starting rule. If this is not the case, it will result in an empty context after grammar compilation.

The non-terminal symbols must be defined in the grammar.

**!import statement****Format**

```
!import symbol;
```

**Description**

The `!import` statement allows you to import frequently used symbols from other contexts.

**Example**

```
!import <Number>;  
!import <Month>;
```

**Comments**

The non-terminal symbols specified in the `!import` statement cannot be defined in the grammar itself.

**Class Section**

The class section consists of a number of class statements.

**Example**

```
!class <City>;  
!class <State>;  
!class <name> !spelling <letters>;
```



**!class statement**

In the !class statement you can define a non-terminal symbol that is a word class, i.e., a list of words to which words can be added or deleted afterwards. Word classes can be edited with the Lexicon Toolkit or with the ASRAPI functions `asrCtxAddWord( )`, `asrCtxAddUserWord( )` and `asrCtxDeleteWord( )`. A word class is a container of alternative words, with the same syntactical meaning. Two different styles of class descriptions exist:

**Format 1**

**!class** symbol;

**Description**

The specified non-terminals are declared as word classes. These word classes should be defined further on in the grammar, using the following syntax:

<symbol> ::

or

<symbol> : word1 | word2 | ... | wordN;

**Example**

*!class <Destination City>;*

*!class <Departure City>;*

**Format 2**

**!class** symbol1 **!spelling** symbol2;

**Description**

symbol1 is a class of words that will be spelled using a syntax described in the rule definition of symbol2. symbol1 is a regular class as described above, and can be exported or used in other rules. For recognition, the words in symbol1 should be spelled using the symbols specified in the syntax definition. Currently this syntax definition should be of the form:

<syntax> : <class>;

where <class> is a class containing the symbols to be recognized in the spelling.

**Example**



```
!export <rule>;  
!class <animal> !spelling <syntax>;  
!class <letter> : a | c | d | g | o | t;  
<animal> : cat | dog;  
<syntax> : <letter>+;  
<rule> : <animal>;
```

This example will recognize “cat” or “dog” upon the utterance of “‘c’ ‘a’ ‘t’ ” or “‘d’ ‘o’ ‘g’ ”.

## Pronounce Section

### **!pronounce statement**

#### **Format**

```
!pronounce vendor_mark word word ;
```

#### **Description**

With the **!pronounce** statement you can override the default phonetic transcriptions for certain words. When such statements are given for a certain terminal word, the given transcriptions are used instead of the default phonetic transcriptions for that word. The phonetic alphabets for specific languages can be found in the appendix.

#### **Example**

```
!pronounce L&H Dr "([T=\ "Pm(L&H+)\ "])# ' d A k . t $ R+ #)";  
!pronounce L&H St "([T=\ "Pm(L&H+)\ "])# ' s t R+ I t #)";
```

Here we define the pronunciations for the abbreviations for doctor and street.

#### **Comments**

There are 3 arguments: the vendor name, the terminal word and the phoneme string (which has to be put between double quotes and parentheses). The phonemes must be part of the L&H+ phonetic alphabet, defined for the specific language.



The specified phonetic transcription also has to **include** a header containing information on the phonetic-model used. Currently there are two models available:

- the [T="A(L&H+)"] header for use with the **Japanese** language, and
- the [T="Pm(L&H+)"] header for use with all other languages.

Tables with the available phonemes for each language can be found in the appendices.

## Rules Section

In the rules section the body of the grammar definition is given. This section is made up of one or more grammar rules. A grammar rule has the form:

<A> : expression ;

<A> represents the non-terminal name; the expression represents a regular expression of zero or more names; the colon and the semicolon are L&H BNF punctuation. Of all the non-terminals, those that are exported have a particular importance. The parser (compiler) is designed to recognize the top-level symbols by the contents of the export statements.

A *rule* specifies a set of symbol strings to be matched. It contains words (which match words to be recognized), non-terminals (which match symbols defined by rules in the grammar), operators and functions (which specify repetitions, choices, and other features).

### Sequences

#### Format

expression expression

#### Description



Sequences concatenate two or more expressions. The white space between consecutive expressions, symbols or words is the implicit concatenation operator.

**Example**

I want to go to <City>

This expression indicates that the words “I”, “want”, “to”, “go”, “to” and the symbol <city> are matched by this expression if they occur in this specific sequential order.

**Alternatives****Format**

expression1 | expression2

**Description**

The value of this sub-expression is either expression1 or expression2. One or more constituents in an alternative expression can be empty. This results in the expression possibly being nothing at all.

See also the !optional operator.

**Example**

*Boston | New York | <Foreign Cities>*

This expression indicates that the word “Boston”, the expression “New York” and the symbol <Foreign City> are all allowed by the expression.

**!repeat function****Format**

!repeat ( expression, NUMBER, number\_or\_asterisk )

**Description**

The expression can be repeated from a minimum NUMBER of times to a maximum number of times. If an asterisk is specified, it means “infinity times”. Note that NUMBER can be 0!

**!optional function****Format**

!optional ( expression )



### **Description**

The !optional function makes an expression optional in a certain rule.

### **Example**

*the !optional( <adjective> ) tree*

### **Comments**

The non-terminal symbol <adjective> is optional in this expression. There may be an adjective, but it is not necessary. Note that this in fact is another notation for:

( <adjective> | )

## **!ignore function**

### **Format**

!ignore ( expression )

### **Description**

The !ignore function makes an expression recognized, but not notified to the application.

### **Example**

*the !ignore("old") tree*

### **Comments**

The full expression (including "old") is recognized, but only the string "the tree" is notified to the application. Note that !ignore can **only** contain **one** single word enclosed by double quotes. Symbols can not be ignored!

## **!action function**

### **Format**

!action( word )

### **Description**

The !action function specifies an additional word or string to be notified to the application, without this being recognized, i.e., for recognition purposes the action string is not present. The word or phrase has to be enclosed in quotation marks.

### **Example**

*file close !action("fclose( fileHandle);");*



### Comments

Upon recognition of the expression, excluding the action string, the full expression (including the action string) is notified to the application, i.e.,

```
file close fclose( fileHandle );
```

### Precedence Levels and Associativity for Operators

The following preceding levels and associativity rules apply:

Precedence	Operator	Associativity
highest	brackets	
	functions	
	sequence	left, binary
lowest	alternative	left, binary

The functions have the highest precedence level. The binary, left associative operator ‘*sequence*’ (which does not have a symbol) has a higher precedence level than the binary, left associative operator *alternative* “|”. The implicit precedence levels can be overruled by the use of brackets.

#### Example

*New York / Amsterdam*

*New ( York / Amsterdam )*

The implicit precedence levels make the first example an alternative list of “New York” and “Amsterdam” where the second example is an alternative list of “New York” and “New Amsterdam”.

## Isolated Word Recognition

Isolated Word Recognition is treated as a special case of continuous speech recognition and is provided by a special



purpose syntax that comes standard with the ASRAPI installation. This special purpose syntax is conceptually equivalent to the following formal description in terms of the L&H continuous speech grammar language.

```
!grammar IsolatedWordRecognition;  
!export <Word>;  
!class <Word>;  
<Word> ::
```

This tiny grammar description defines the symbol <word> as a word class. The only rule in the grammar states that valid utterances are a <word> and that the symbol <word> is empty. When new contexts are created with this syntax, words can be added to the <word> word class using the `asrCtxAddWord()` function.

## Key Word Spotting

Keyword Spotting is treated as a special case of continuous speech recognition and is provided by a special purpose syntax that comes standard with the ASRAPI installation. This special purpose syntax is conceptually equivalent to the following formal description in terms of the L&H continuous speech grammar language.

```
!grammar KeyWordSpotting;  
!export <start>;  
!class <Keyword>;  
<start> : <...> <Keyword> <...>;  
<Keyword> ::
```

This grammar description defines the symbol <keyword> as a word class. The first rule states that a valid utterance is a <keyword> surrounded by garbage (noise, non-vocabulary speech, etc.). The second rule that defines the symbol <keyword> states that the class is empty. When new contexts are created with this syntax, words can be added to the <keyword> word class using the `asrCtxAddWord()` function.



## Connected Digit Recognition

Connected Digit Recognition is treated as a special case of continuous speech recognition and is provided by a special purpose syntax that comes standard with the ASRAPI installation. This special purpose syntax is conceptually equivalent to the following formal description in terms of the L&H continuous speech grammar language.

```
!grammar ConnectedDigits;
!export <start>;
!class <digit>;
<start> : !repeat(<digit>,1,*);
<digit> : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;
```

The first rule states that a valid utterance is a repetition of one or more <digit>. The second rule that defines the symbol <digit> states that the symbol is one of the digits. This grammar is specific for each language since it contains specific words.

## L&H BNF Grammar Language Formal Specification

### Terminal Symbol Notation, Word, Token

Two notations exist for terminal symbols:

- 1) any sequence of letters and digits, beginning with letter: ASRAPI, take, word, I, UB40.
- 2) any text inside double quotation marks: "Lernout & Hauspie Speech Products", "B-52", "tic-tac-toe", "&\$^%@"\*\"n\r\12kg".

Such strings can include escape sequences like:



\a	BEL	(007)	\t	TAB	(011)
\b	BS	(010)	\v	VTAB	(013)
\f	FF	(014)	\"	"	
\n	NL	(012)	\'	'	
\r	CR	(015)	\\	\	

## Non-terminal Symbol Notation, Symbol

Non-terminal symbols are of the form <any text excluding some characters>. Any text is defined here as the printable ASCII characters. Non-printing characters such as carriage return, line feed, form feed, horizontal tabulation, vertical tabulation, etc. cannot be used. Also the ">" character cannot be used.

## Garbage Class

The garbage class is a special class, written as <...>, which can be used in the grammar specification to indicate places where any phrase, any speech can be uttered. This class is used, e.g., to specify the keyword spotting syntax in this BNF language.

## Compiler Keywords Specification

Compiler keywords are differentiated from regular terminal symbols by a leading '!' character. Here is the list of keywords:

- !grammar - grammar statement



- !language - language statement
- !pronounce - pronounce statement
- !class - word class statement
- !spelling - spelling syntax identifier
- !export - non-terminals export statement
- !import - non-terminals import statement
- !repeat - repeat function (expression can be repeated)
- !optional - optional function (expression is optional)
- !ignore - pseudo-function (do not notify recognized text to the application)
- !action notified - action-string function - extra info is to the application

## Formal Language Specification

The `#include` statement and C/C++ style comments, being preprocessor statements, are not described in this specification.

```

grammar :
    sequence-of-statements
sequence-of-statements :
    <empty>
statement sequence-of-statements
statement :
    <empty> ;
    grammar-statement ;
    language-statement ;
    export-statement ;
    import-statement ;
    class-statement ;
    pronounce-statement ;
    rule-statement ;

```



```
grammar-statement :  
    !grammar word  
  
language-statement :  
    !language word  
  
export-statement :  
    !export symbol  
  
import-statement :  
    !import symbol  
  
class-statement :  
    !class symbol  
    !class symbol !spelling symbol  
  
pronounce-statement :  
    !pronounce vendor-mark word word  
  
rule-statement :  
    symbol : comp-expression  
  
comp-expression :  
    <empty>  
    comp-expression expression  
    comp-expression / comp-expression  
  
expression :  
    word  
    symbol  
    ( comp-expression )  
    operator-expression  
  
operator-expression :  
    repeat-operator  
    optional-operator  
    ignore-operator  
    action-operator  
  
repeat_operator :  
    !repeat ( comp-expression, integer,  
    integer )  
    !repeat ( comp-expression, integer, * )  
  
optional-operator:  
    !optional ( comp-expression )  
  
ignore-operator:  
    !ignore ( word )
```



```
action-operator:
    !action ( word )

symbol-list :
    ( symbol sequence-of-symbols )
    symbol sequence-of-symbols

vendor-mark :
    L&H
```

## Design Tips

The more your grammar is adjusted to the situation, the better the recognition will be. The aim is to be able to recognize all valid commands and only those. Making sure all valid commands are included is mostly not too difficult. On the other hand, making sure that only those are added may require some more effort.

Even if you have a good model of the speech that is likely to be produced, there are some other factors that may influence the quality of the recognition. Grammars with seemingly the same effect still may have different behavior. In the following sections, we will tackle some of the thorny issues.

## Content

In this section we will discuss the content of a grammar. Even if most of the comments here seem pretty obvious, some of them may prove very useful.

### Strings

The first thing you should consider is to use long words/phrases rather than short ones. The longer the phrase, the more information is available, and the more accurate the recognition result will be. In this respect, it is also wise to use as distinct phrases as possible. The example below illustrates



how difficult it may be to make a correct distinction between two sentences, as they differ in only a very small location.

*"I am going to the store"*

*"I am going in the store"*

As a rule of thumb, try to use longer and more distinct sentences, even though logically it does not seem necessary.

Another problem may be opposite words. Many of them sound very similar. Typically, the only difference between the two resides in prefixes like “un-” and “in-”. (For instance “popular” versus “unpopular”). It is good practice to avoid using these kinds of opposites, although it might be very difficult to come up with good, logical alternatives. Using such opposites in longer sentences, where the ‘carrier’ sentences themselves are sufficiently different may help to differentiate between the two.

### Quotes

You have to use quotes with the necessary caution. When your terminal symbols contain phrases, instead of a single word, using quotes will increase the size of the grammar. The reason for this is that quoted phrases are regarded as single words. The following example illustrates this:

*"I am going to the store" | "I am going in the store"*

vs.

*I am going to the store | I am going in the store*

In the first case, the engine would consider that there are only two ‘words’, versus eleven in the second case. As a result, the size of the phonetic information stored will be larger for the first alternative. The reason for this is that in the first case, similar words will be stored phonetically multiple times. For instance, the phonetic string for “going” will be stored once as part of the “I am going to the store”-word and once for



the “I am going in the store”-word, whereas in the second case it is stored only once.

Moreover, the use of quotes also requires the user to speak the quoted string in a continuous way. The engine will not take into account possible gaps or short pauses between the words. On average, this results in poorer results if the users are not familiar with the possible answers.

As a general guideline, it is best not to use quotes.

However, there are some situations where the use of quotes is recommended. Some words have to be pronounced as a single word. An example of this would be the French *liaisons*, i.e. the linking of words. For instance “Ils ont” should not be pronounced as two separate words but rather as a single word “Ilsont”. Another less obvious example would be “New York” in English (and other languages).

### **Interaction**

A context free grammar is not a dictation grammar: users cannot just say what they want. A grammar should be designed to handle specific syntaxes. It is essential to be consistent in the design. That makes it easier for the users to get familiar with the context and ‘anticipate’ the expected responses. The use of well-chosen prompts can help as well. The less confused the user is about a possible response, the higher the chance he will use a correct one and the higher the accuracy of the recognition.

It is also important to give feedback to the user so that he may know that his command has been correctly understood. Feedback, although adequate, should not be overdone to avoid slowing down the dialog. Note that this feedback could also be blended in the prompts.

### **Limit the use of ‘special features’**

Although the special features may help you to create the perfect grammar, their use often restricts the optimization of



the grammars, resulting in longer response times, and decreased accuracy. Structures like !action, !ignore, etc. should be used moderately. The same applies to the uses of classes. Do not turn a symbol into a class unless you really intend to add words to or delete words from it. Classes too limit the number of optimization that can be performed.

The garbage model is somewhat special. It is actually a 'dangerous' thing to use as it models 'any' speech. If you know what is going to be said, even though you are not interested in it, it will give better performance if you list this speech rather than using the garbage model. Even if you know only a limited part of the text that should definitely be spoken within the space where you are using <...>, put that text in the grammar and surround it with <...>. The more speech you can identify the better the results.

Something similar to the overuse of the garbage-model is the overuse of repetitions. If you know the maximum number of repetitions, it is better to specify this maximum rather than just specifying an \*. It serves no purpose trying to recognize 8 digits in a 7-digit telephone-number.

Take a look at the following lines:

*What's your name / What's your address;*

vs.

*What's your (name / address);*

Both have the same effect, but the second option is better. If there are common parts, it is good practice to separate them from the specific parts. In this regard, the second option below is also better than the first:

*<digit><digit>/<digit><digit><digit>;*

vs.

*<digit><digit>!optional(<digit>;*



## Vocabulary Management

We already explained that you only want those words or phrases in your grammar that can actually be spoken and only those. This is not always enough. A context is a general representation of the speech that can be spoken during the life span of an application. This does not necessarily mean that all of these words have a valid meaning all of the time. Therefore it would be better to limit the recognizable (active) words to specific time frames in an application's life span. The fewer the number of words that are active at any point in the grammar, the lower the branching factor, and the easier to recognize.

In this respect, we would like to point out the importance of *exported* symbols. Symbols that are exported can be activated, deactivated, enabled or disabled, thus allowing easy vocabulary management. For more information on how to accomplish this, refer to the section on Vocabulary Management in the *User's Guide*.



---

---



# THE L&H+ PHONETIC TRANSCRIPTION SYSTEM

## Introduction

Phonetic transcriptions are an essential tool in the description of human speech. The aim of a phonetic transcription is to provide a method of representing pronunciation in a written form in an unambiguous manner.

The most widely used phonetic alphabet is the one established by the *International Phonetic Association (IPA)*. This alphabet consists mainly of the letters of the Roman alphabet together with a few letters from the Greek alphabet. A few other symbols are especially designed for IPA. The general principle is that *a distinctive sound is represented by one and only one symbol*. The IPA system also provides a set of *diacritics* to be added to the basic symbols. The purpose of these diacritics is to allow a finer description of the sounds if required. The last version of the IPA alphabet dates from 1993.

The set of IPA symbols is not available on standard keyboards and it is not supported in common computer character sets such as ASCII. To overcome this problem L&H has designed an encoding system that allows for the representation of the complete IPA character set using a limited range of ASCII symbols. This encoding system is called L&H+.

L&H+ covers all IPA symbols in the 1993 version but it also adds a few symbols to represent phonetic elements not



included in the IPA system. L&H+ uses members of the ASCII set in the range 33-126 to construct a phonetic symbol. In L&H+ the percentage symbol “%” separates the basic symbol from the diacritics. For example, IPA /ɛ / is /E%~/ in L&H+.

The tables on the following pages provide the list of phonemes for several languages with their corresponding transcriptions in the L&H+ and IPA systems. Written and transcribed examples per phoneme are included. In these examples, bold letters indicate the graphemes that correspond to each phoneme in normal pronunciation.

## Multiple Pronunciations

In all languages, the orthographic representation of a word can have (sometimes completely) different pronunciations.

These alternative pronunciations sometimes can be represented orthographically. For example, in English the digit ‘0’ can be read as *null* or as *zero*.

In other cases only a phonetic transcription can reflect the pronunciation variability. For instance, the word ‘*the*’ can be pronounced as /’DS/ or /’Di/ (depending on the context).

Another example is the word ‘*different*’, which can be read as /’dI.f\$.R+\$nt/ but also as /’dI.fR+\$nt/. The difference between the two transcriptions is the deletion of a /’\$’/.

In order to cope with these cases, a notational formalism is introduced here. This formalism provides an economic way of representing variability of pronunciations and increases the flexibility of traditional phonetic transcriptions.



The individual alternative pronunciations are separated by the pipe symbol ('|') and enclosed by parentheses. The action of deletion is represented by the deletion symbol (=0).

Applying this formalism to the above examples gives:

'0' ( null | zero)  
'the' /D ( \$ | i )/  
'different' /'dI.f ( \$ | =0 ).R+\$nt/

This formalism should be used with care. The combination of alternative pronunciations can lead to the generation of invalid strings of phonemes or sounds. For example, the word '*record*' can be transcribed either as /R+I.'kOR+d/ (the verb *to record*) or as /'R+E.kOR+d/ (the noun *record*). Combining these two representations into /(' | =0) R+ ( I | E).( ' | =0)kOR+d / can generate 8 possible combinations.

One correct way to reflect the alternatives in a phonetic transcription based on our notation system is /(R+I. ' | 'R+E.) kOR+d/.



---

---



# Phoneme Table American English

L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Vowels</b>				
i	i	beat	'bit	'bit
I	ɪ	bit	'bIt	'bɪt
E	ɛ	bed	'bEd	'bed
@	æ	map	'm@p	'mæp
A	ɑ	car	'kAR+	'kar
^	ʌ	but	'b^t	'bʌt
O	ɔ	bought	'bOt	'bɔt
U	ʊ	book	'bUk	'buk
u	u	boot	'but	'but
\$	ə	about	\$. 'ba&Ut	ə. 'baʊt
E0	ɜ	turn	'tE0R+n	'tɜɹn
<b>Diphthongs</b>				
e&I	eɪ	bait	'be&It	'beɪt
O&I	ɔɪ	boy	'bO&I	'boɪ
a&I	aɪ	buy	'ba&I	'baɪ
a&U	aʊ	down	'da&Un	'daʊn
o&U	oʊ	show	'So&U	'ʃoʊ
<b>Approximants</b>				
j	j	you	'ju	'ju
w	w	wit	'wIt	'wɪt
R+	ɹ	ride	'R+a&Id	'ɹaɪd
l	l	let	'lEt	'let



L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Plosives</b>				
p	p	pan	'p@n	'pæn
t	t	tan	't@n	'tæn
k	k	can	'k@n	'kæn
b	b	boy	'bO&I	'bɔɪ
d	d	day	'de&I	'deɪ
g	g	got	'gAt	'gɒt
ʔ	ʔ	glottal stop	'ʔit	'ʔɪt
<b>Fricatives</b>				
f	f	fine	'fa&In	'faɪn
θ	θ	thin	'θIn	'θɪn
s	s	sin	'sIn	'sɪn
ʃ	ʃ	shine	'Sa&In	'ʃaɪn
v	v	vine	'va&In	'vaɪn
ð	ð	that	'D@t	'ðæt
z	z	zone	'zo&Un	'zəʊn
ʒ	ʒ	vision	'vɪ.Zʒn	'vɪ.ʒən
h	h	head	'hEd	'hed
<b>Affricates</b>				
t&S	tʃ	church	't&SE0R+t&S	'tʃɜ:ʃ
d&Z	dʒ	jungle	'd&Z^ nK.g\$!l	'dʒʌŋ.gəl
<b>Nasals</b>				
m	m	my	'ma&I	'maɪ
n	n	no	'no&U	'noʊ
nK	ŋ	sing	'sInK	'sɪŋ
<b>Special</b>				
'	'	primary stress		
'2	'2	secondary stress		
.	.	syllable boundary		
-	-	word boundary		
#	#	silence (pause)		



## Phoneme Table Spanish

L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Vowels</b>				
i	i	vivo	'bi.Bo	'bi.βo
e	e	peso	'pe.so	'pe.so
a	a	casa	'ka.sa	'ka.sa
o	o	bobo	'bo.Bo	'bo.βo
u	u	cuñado	ku.'n~a.Do	ku.'ja.ðo
<b>Approximants</b>				
j	j	ciego	'Tje.Go	'θje.yo
w	w	cuenta	'kwen.ta	'kwen.ta
l	l	lata	'la.ta	'la.ta
<b>Trill</b>				
r	r	carro	'ka.ro	'ka.ro
<b>Tap</b>				
r6	ɾ	caro	'ka.r6o	'ka.ro
<b>Plosives</b>				
p	p	pepa	'pe.pa	'pe.pa
t	t	tía	'ti.a	'ti.a
k	k	kilo	'ki.lo	'ki.lo
b	b	cambio	'kam.bjo	'kam.bjo
d	d	dedo	'de.Do	'de.ðo
g	g	hangar	an.'gar6	an.'gar



L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
Fricatives				
f	f	fin	'fin	'fin
T	θ	cero	'Te.r6o	'θe.ro
D	ð	dedo	'de.Do	'de.ðo
s	s	seso	'se.so	'se.so
J	ʃ	mayo	'ma.Jo	'ma.ʃo
x	ɣ	ajo	'a.xo	'a.xo
G	ɣ	agua	'a.Gwa	'a.ɣwa
B	z	bobo	'bo.Bo	'bo.βo
Affricates				
t&S	tʃ	chico	't&Si.ko	'tʃi.ko
d&Z	dʒ	yo	'd&Zo	'dʒo
Nasals				
m	m	amo	'a.mo	'a.mo
n	n	nene	'ne.ne	'ne.ne
n~	ɲ	niño	'ni.n~o	'ni.ɲo
Special				
'	'	primary stress		
.	.	syllable boundary		
-	-	word boundary		
#	#	silence (pause)		



## Phoneme Table Italian

L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Vowels</b>				
a	a	case	'ka.se	'ka.se
e	e	nero	'ne.ro	'ne.ro
E	ɛ	epoca	'E.po.ka	'ɛ.po.ka
i	i	vino	'vi.no	'vi.no
o	o	volo	'vo.lo	'vo.lo
O	ɔ	cosa	'kO.sa	'kɔ.sa
u	u	lumaca	lu.'ma.ka	lu.'ma.ka
<b>Diphthongs</b>				
e&u	eu	europa	e&u.'rO.pa	eu.'rɔ.pa
E&u	ɛu	feudo	'fE&u.do	'fɛu.do
a&u	au	causa	'ka&u.za	'kau.za
<b>Trills</b>				
r	r	raro	'ra.ro	'ra.ro
r:	r:	carro	'ka.r:o	'ka.rɔ
<b>Approximants</b>				
j	j	piú	'pju	'θje.ɣo
w	w	può	'pwO	'pwɔ
l	l	lama	'la.ma	'la.ma
l:	l:	bello	'bE.l:o	'bɛ.lɔ
l~	ɭ	gli	'l~i	'ɭi
l~:	ɭ:	figlio	'fi.l~:o	'fi.ɭɔ



L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Plosives</b>				
p	p	copia	'kO.pja	'kɔ.pja
p:	p:	pappa	'pa.p:a	'pa.p:a
t	t	tana	'ta.na	'ta.na
t:	t:	fatto	'fa.t:o	'fa.tɔ
k	k	casa	'ka.sa	'ka.sa
k:	k:	macchia	'ma.k:ja	'ma.kja
b	b	bacio	'ba.t&So	'ba.tʃo
b:	b:	babbo	'ba.b:o	'ba.bɔ
d	d	dito	'di.to	'di.to
d:	d:	freddo	'fre.d:o	'fre.dɔ
g	g	ago	'a.go	'a.go
g:	g:	leggo	'lE.g:o	'lɛ.go
<b>Fricatives</b>				
f	f	festa	'fEs.ta	'fes.ta
f:	f:	baffi	'ba.f:i	'ba.fi
s	s	sole	'so.le	'so.le
s:	s:	fisso	'fi.s:o	'fi.sɔ
S	ʃ	scivolo	'Si.vo.lo	'ʃi.vo.lo
S:	ʃ:	liscio	'li.S:o	'li.ʃo
v	v	vino	'vi.no	'vi.no
v:	v:	ovvio	'O.vjo	'ɔ.vjo
z	z	viso	'vi.zo	'vi.zo
<b>Affricates</b>				
t&s	ts̩	zio	't&si.o	'ts̩i.o
t&s:	ts̩:	ragazza	ra.'ga.t&s:a	ra.'ga.ts̩a
t&S	tʃ	ci	't&Si	'tʃi
t&S:	tʃ:	faccia	'fa.t&S:a	'fa.tʃ:a
d&Z	dʒ	giusto	'd&Zus.to	'dʒus.to
d&Z:	dʒ:	oggi	'O.d&Z:i	'ɔ.dʒ:i
d&z	d͡z	zona	'd&zO.na	'd͡zɔ.na
d&z:	d͡z:	mezzo	'mE.d&z:o	'mɛ.d͡zɔ



L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Nasals</b>				
m	m	mano	'ma.no	'ma.no
m:	m:	mamma	'ma.m:a	'ma.m:a
n	n	nano	'na.no	'na.no
n:	n:	danno	'da.n:o	'da.n:o
n~	ɲ	gnocco	'n~O.mo	'ɲo.mo
n~:	ɲ:	ogni	'O.n~:i	'ɔ.ɲi
<b>Special</b>				
'	ˈ	primary stress		
'2	ˈ	secondary stress		
.	.	syllable boundary		
—		word boundary		
#		silence (pause)		



---

---



# Phoneme Table German

L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Vowels</b>				
i:	i:	Riese	'Ri:.z\$	'ri:.zə
y:	y:	grün	'gRy:n	'gry:n
ɪ	ɪ	Milch	'mɪɕ	'milç
ʏ	ʏ	Küste	'kʏs.t\$	'kʏs.tə
e:	e:	Kehle	'ke:.l\$	'ke:.lə
e+	ø	Öl	'e+l	'øl
ɛ	ɛ	letzte	'lɛt.st\$	'lets.tə
ɛ+	œ	löschen	'lɛ+.S\$n	'lœ.ʃən
a	a	Stadt	'Stat	'ʃtat
a:	a:	Wagen	'va:.g\$n	'va:.gən
ɔ	ɔ	voll	'fɔl	'fəl
o:	o:	groß	'gRo:s	'gro:s
ʊ	ʊ	Kunst	'kʊnst	'kunst
u:	u:	Fuß	'fu:s	'fʊs
\$	ə	Taste	'tas.t\$	'tas.tə
<b>Diphthongs</b>				
O&y	oy	heute	'hO&y.t\$	'hoy.tə
a&ɪ	ai	Teil	'ta&ɪl	'tail
a&u	au	Baum	'ba&um	'baum
<b>Approximants</b>				
j	j	jemand	'je:.mant	'je:.mant
ɫ	ɫ	Licht	'ɫɪɕt	'liçt
<b>Trill</b>				
R	ʀ	Reise	'Ra&i.z\$	'rai.zə



L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Plosives</b>				
p	p	Post	'pOst	'pɔst
t	t	Tinte	'tIn.t\$	'tɪn.tə
k	k	klein	'kla&in	'klaɪn
b	b	Bein	'ba&in	'baɪn
d	d	dich	'dIC	'dɪç
g	g	liegen	'li:.g\$ŋ	'li:.gən
ʔ	ʔ	(glottal stop)	b\$.ʔax.t\$ŋ	bə.ʔax.tən
<b>Fricatives</b>				
f	f	Feld	'fElT	'felt
s	s	Fels	'fElS	'fels
S	ʃ	Schnee	'Sne:	'ʃne:
v	v	wach	'vax	'vax
z	z	Saal	'za:l	'zaɪl
Z	ʒ	Journal	ZUR.'na:l	ʒʊR.'naɪl
C	ç	Milch	'mIlC	'mɪlç
x	x	Bach	'bax	'bax
h	h	Hand	'hant	'hant
<b>Affricates</b>				
p&f	p̥f	Pferd	'p&fe:Rt	'p̥fe:ɪt
t&s	t̪s	Zug	't&su:k	't̪su:k
t&S	t̪ʃ	klatschen	'kla.t&S\$ŋ	'kla.t̪ʃən
d&Z	d̪ʒ	Gin	'd&ZIn	'd̪ʒɪn
<b>Nasals</b>				
m	m	Mann	'man	'man
n	n	Norden	'nOR-d\$ŋ	'nɔR.dən
nK	ŋ	Ring	'RInK	'Rɪŋ
<b>Special</b>				
ˈ	ˈ	primary stress		
ˈ2	ˈ	secondary stress		
ˌ	ˌ	syllable boundary		
–		word boundary		
#		silence (pause)		



# Phoneme Table British English

L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Vowels</b>				
i	i	beat	'bit	'bit
I	ɪ	bit	'bIt	'bɪt
E	ɛ	bed	'bEd	'bɛd
@	æ	map	'm@p	'mæp
A	ɑ	car	'kA	'kɑ
Λ+	ɒ	pot	'pA+t	'pɒt
^	ʌ	but	'b^t	'bʌt
O	ɔ	bought	'bOt	'bɔt
U	ʊ	book	'bUk	'bʊk
u	u	boot	'but	'but
\$	ə	about	\$. 'ba&Ut	ə. 'baʊt
E0	ɜ	turn	'tE0n	'tɜn
<b>Diphthongs</b>				
I&\$	ɪə	here	'hI&\$	'hɪə
E&\$	ɛə	there	'DE&\$	'dɛə
U&\$	ʊə	poor	'pU&\$	'pʊə
e&I	eɪ	bait	'be&It	'beɪt
O&I	ɔɪ	boy	'bO&I	'bɔɪ
a&I	aɪ	buy	'ba&I	'baɪ
a&U	aʊ	down	'da&Un	'daʊn
o&U	oʊ	show	'So&U	'ʃoʊ
<b>Approximants</b>				
j	j	you	'ju	'ju
w	w	wit	'wIt	'wɪt
R+	ɹ	ride	'R+a&Id	'ɹaɪd
l	l	let	'lEt	'lɛt



L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Plosives</b>				
p	p	pan	'p@n	'pæn
t	t	tan	't@n	'tæn
k	k	can	'k@n	'kæn
b	b	boy	'bO&I	'bɔɪ
d	d	day	'de&I	'deɪ
g	g	got	'gA+t	'gɒt
ʔ	ʔ	(glottal stop)	'ʔit	'ʔɪt
<b>Fricatives</b>				
f	f	fine	'fa&In	'faɪn
θ	θ	thin	'θIn	'θɪn
s	s	sin	'sIn	'sɪn
ʃ	ʃ	shine	'Sa&In	'ʃaɪn
v	v	vine	'va&In	'vaɪn
ð	ð	that	'D@t	'ðæt
z	z	zone	'zo&Un	'zəʊn
ʒ	ʒ	vision	'vɪ.Z\$ɪn	'vɪ.ʒən
h	h	head	'hEd	'hed
<b>Affricates</b>				
t&S	tʃ	church	't&SE0t&S	'tʃɜ:ʃ
d&Z	dʒ	jungle	'd&Z^ nK.g\$l	'dʒʌŋ.gəl
<b>Nasals</b>				
m	m	my	'ma&I	'maɪ
n	n	no	'no&U	'noʊ
nK	ŋ	sing	'sInK	'sɪŋ
<b>Special</b>				
'	'	primary stress		
'2	'2	secondary stress		
.	.	syllable boundary		
–	–	word boundary		
#	#	silence (pause)		



# Phoneme Table French

L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Vowels</b>				
i	i	minute	mi.'nyt	mi.'nyt
y	y	mûr	'myR	'myr
e	e	élément	e.le.'mA%~	e.le.'mã
e+	ø	deux	'de+	'dø
E	ɛ	très	'tRE	'tre
E+	œ	oeuvre	'E+.vR\$	'œ.vrə
a	a	la	'la	'la
A	ɑ	pâte	'pAt	'pat
O	ɔ	morte	'mORt	'mort
o	o	eau	'o	'o
u	u	tour	'tuR	'tur
\$	ə	le	'l\$	'lə
E%~	ẽ	vin	'vE%~	'vẽ
A%~	ā	blanc	'bLA%~	'blā
O%~	ō	bon	'bO%~	'bō
E+%~	œ̃	brun	'bRE+%~	'brœ̃
<b>Approximants</b>				
j	j	briller	bRi.'je	bri.'je
h\	ɥ	lui	'lh\i	'lɥi
w	w	oui	'wi	'wi
l	l	la	'la	'la
<b>Tap</b>				
R	r	rond	'RO%~	'rɔ̃



L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Plosives</b>				
p	p	pas	'pa	'pa
t	t	tas	'ta	'ta
k	k	canne	'kan	'kan
b	b	bas	'ba	'ba
d	d	de	'd\$	də
g	g	gomme	'gOm	'gəm
<b>Fricatives</b>				
f	f	faim	'fE%~	'fɛ
s	s	sauce	'sos	'sɔs
S	ʃ	charme	'SaR.m\$	'ʃar.mə
v	v	vol	'vOl	'vɔl
z	z	zéro	ze.'Ro	ze.'ro
Z	ʒ	jardin	ZaR.'dE%~	ʒar.'dɛ
<b>Nasals</b>				
m	m	mot	'mo	'mo
n	n	nous	'nu	'nu
n~	ɲ	agneau	a.'n~o	a.'ɲo
nK	ŋ	smoking	smO.'kinK	smɔ.'kiŋ
<b>Special</b>				
'	'	primary stress		
.	.	syllable boundary		
-		word boundary		
#		silence (pause)		



# Phoneme Table Dutch

L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Vowels</b>				
i	i	vies	'vis	'vis
y	y	vuur	'vyr	'vyr
ɪ	ɪ	vis	'vɪs	'vis
e	e	veel	'vel	'vel
e+	ø	beuk	'be+k	'bøk
ɛ	ɛ	pet	'pɛt	'pet
E:	ɛ:	primaire	'pri.mɛ:r	pri.'mɛr
a	a	paars	'pars	'pars
ʌ	ʌ	pak	'pʌk	'pak
^	ʌ	put	'p^t	'pat
O	ɔ	pot	'pOt	'pɔt
o	o	poot	'pot	'pɔt
u	u	voet	'vut	'vut
\$	ɔ	de	'd\$	'də
<b>Diphthongs</b>				
E&I	ɛi	hij	'hE&i	'hei
A&u	au	vrouw	'vrA&u	'vrɔu
^&y	ʌy	tuin	't^&yn	'tɔyn
<b>Approximants</b>				
j	j	jood	'jot	'jɔt
w	w	nieuw	'niw	'niw
V	v	water	'Va.t\$ɾ	'vɔ.təɾ
l	l	lach	'lAx	'lax
<b>Trill</b>				
r	r	rood	'rot	'rɔt



L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Plosives</b>				
p	p	poot	'pot	'pɒt
t	t	toen	'tun	'tʊn
k	k	zak	'zAk	'zak
b	b	boot	'bot	'bɒt
d	d	doen	'dun	'dʊn
g	g	zakdoek	'zAg.duk	'zak.dʊk
ʔ	ʔ	(glottal stop)	G\$.?ar.z\$lt	ɣə.'ar.zəlt
<b>Fricatives</b>				
f	f	fiets	'fits	'fɪts
s	s	sap	'sAp	'sap
ʃ	ʃ	show	'Sow	'ʃow
v	v	leven	'le.v\$n	'le.vən
z	z	zap	'zAp	'zap
ʒ	ʒ	journaal	Zur.'nal	ʒur.'nal
x	x	lach	'lAx	'lax
ɣ	ɣ	lagen	'la.G\$n	'la.ɣən
h	h	hoed	'hut	'hʊt
<b>Nasals</b>				
m	m	moed	'mut	'mʊt
n	n	niet	'nit	'nɪt
nK	ŋ	bang	'bAnK	'baŋ
<b>Special</b>				
'	'	primary stress		
.	.	syllable boundary		
_		word boundary		
#		silence (pause)		



# Phoneme Table Japanese

L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Vowels</b>				
A	α	角	kA'do	kado
A:	α:	カード	'FkA:do	ka:do
i	i	居合い	iAi	iai
i:	i:	言い合い	i: Ai	i: ai
u-	u	売る	u-r6u-	uɾu
u-:	u:	ウール	'Fu:-r6u-	u:ɾu
e	e	風	kAze	kaze
e:	e:	賛成	sANse:	sanse:
o	o	てこ	te'ko	teko
o:	o:	立法	r6ip:o:	ɾip:o:
<b>Plosives</b>				
p	p	スパイ	supA'i	supai
p:	p:	すっぱい	sup:A'i	sup:ai
b	b	馬場	bA'bA	baba
b:	b:	アップード	A'Fb:A:do	ab:a:do
t	t	肩	kA'tA	kata
t:	t:	買った	kA':t:A	kat:a
d	d	駄々	dA'dA	dada
d:	d:	ヘッド	he'd:o	hed:o
k	k	坂	sAkA'	saka
k:	k:	サッカー	sA'k:A:	sak:a
g	g	ガラス	gAr6Asu-	garasu
g:	g:	ドッグ	do'g:u-	dog:u
ʔ	ʔ	あっあ	AʔA	αʔα



Fricatives				
z	z	飾る	kAzAr6u-	kazaru
Z	ʒ	喜寿	ki'Zu-	kizɯ
h	h	花	hAnA'	hana
h:	h:	バッハ	ba'h:A	bah:a
P	ɸ	ファン	PA'N	ɸaɴ
P:	ɸ:	バッファー	bAP:A:	baɸ:a:
s	s	火災	kAsAi	kasai
s:	s:	喝采	kAs:Ai	kas:ai
S	ʃ	歌手	kA'Su-	kaʃɯ
S:	ʃ:	脱臭	dAS:u:-	daʃ:ɯ:
v	v	ヴァイオリン	vAior6iN	vaiorin
v:	v:	サッヴィ	sA'v:i	sav:i
Affricates				
t&s	ts	説	se't&su-	setsɯ:
t&s:	ts:	撰津	set&s:u-	sets:ɯ
t&S	tʃ	調査	'Ft&So:sA	tʃo:sa
t&S:	tʃ:	発注	hAt&S:u:-	hatʃ:ɯ:
z:	z	オッズ	o'z:u-	oz:ɯ
Z:	ʒ:	バッジョ	bA'Z:o	baʒ:o
Nasals				
m	m	巻き	mAki	maki
n	n	何	nA'ni	nani
N	ɴ	三	sAN	san
Flap				
r6	r	楽	r6Aku-'	raɸɯ
r6:	r:	アッラー	Ar6:A:	ar:a:
Approximant				
j	j	野菜	jAsAi	jasai
w	w	私	wAtASi	wataɸi



Special				
'		Falling accent		
'F		Late falling accent		



---

---



## Phoneme Table Korean

L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Vowels</b>				
a	a	아기	a.gi	a.gi
O	o	엄마	Om.ma	om.ma
o	o	오이	o.i	o.i
u	u	우유	u.ju	u.ju
u-	u	음성	u-m.sOnK	um.sŋ
i	i	이름	i.ru-m	i.rum
e	e	에는	e.nu-n	e.nun
e+	ø	회사	he+.sa	hø.sa
E	ɛ	개발	kE.bal	kɛ.bal
<b>Diphthongs</b>				
u-&i	ui	의사	u-&i.sa	ui.sa
ja	ja	야기	ja.gi	ja.gi
jO	jɔ	여름	jO.ru-m	jɔ.rum
jo	jo	요가	jo.ga	jo.ga
ju	ju	유람선	ju.ram.sOn	ju.ram.sŋ
je	je	예절	je.j-Ōl	je.jɔl
jE	jɛ	애기	jE.gi	jɛ.gi
wE	wɛ	왜	wE	wɛ
wi	wi	위기	wi.gi	wi.gi
wa	wa	완수	wan.su	wan.su
we	wɛ	웬말	wen.mal	wen.mal
wO	wɔ	원고지	wOn.go.j-i	wŋ.go.ji



L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Plosives</b>				
p	p	바다	pa.da	p a. da
b	b	지방	ci.banK	ci. baŋ
p`	p'	빨간	p`al.gan	p`al. gan
p{	p <sup>h</sup>	파괴	p{a.gwe	p <sup>h</sup> a. gwe
t	t	다리	ta.ri	ta. ri
d	d	외다리	e+.da.ri	we. da. ri
t`	t'	딸기	t`al.gi	t`al. gi
t{	t <sup>h</sup>	탄소	t{an.so	t <sup>h</sup> an. so
k	k	가랑비	ka.ranK.bi	ka. raŋ. bi
g	g	도구	to.gu	to. gu
k`	k'	까치	k`a.c{I	k`a. c <sup>h</sup> i
k{	k <sup>h</sup>	콩나물	k{onK.na.mul	k <sup>h</sup> oŋ. na. mul
c	c	자식	ca.sik	ca. sik
j-	j	자장가	ca.j-anK.ga	ca. ʒaŋ. ga
c`	c'	짜임새	c`a.im.sE	ca. im. sɛ
c{	c <sup>h</sup>	차남	c{a.nam	c <sup>h</sup> a. nam
?	?	음성	u-m.sOnK	u m. sɔŋ
<b>Fricatives</b>				
s	s	사람	sa.ram	sa. ram
s`	s'	싸움	s`a.um	s`a. um
h	h	하늘	ha.nu-l	ha. nu l
h6	ɦ	무허가	mu.h6O.ga	mu. ɦɔ. ga



L&H+	IP A	Example	L&H+ Transcription	IPA Transcription
<b>Approximants</b>				
j	j	거울	kjO.ul	kjɔ.ul
w	w	웬말	wen.mal	wen.mal
<b>Nasals</b>				
m	m	마술	ma.sul	ma.sul
n	n	나라	na.ra	na.ra
nK	ŋ	강산	kanK.san	kaŋ.san
<b>Laterals</b>				
l	l	갈등	kal.tʰu-N	kal.tʰuŋ
r	r	실연	si.rjOn	si.rjɔn
<b>Special</b>				
.	.	syllable boundary		
#		silence (pause)		
<space>		Word/phrase delimiter		
*		End of declarative		
*,		comma		
*!		End of exclamation		
*?		End of question		
*,		semicolon		
*,		colon		



---

---



# Phoneme Table Mexican Spanish

L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
<b>Vowels</b>				
i	i	vivo	'bi.Bo	'bi.βo
e	e	peso	'pe.so	'pe.so
a	a	casa	'ka.sa	'ka.sa
o	o	bobo	'bo.Bo	'bo.βo
u	u	cuñado	ku.'n~a.Do	ku.'ja.ðo
<b>Approximants</b>				
j	j	ciego	'Tje.Go	'θje.yo
w	w	cuenta	'kwen.ta	'kwen.ta
l	l	lata	'la.ta	'la.ta
<b>Trill</b>				
r	r	carro	'ka.ro	'ka.ro
<b>Tap</b>				
r6	ɾ	caro	'ka.r6o	'ka.ro
<b>Plosives</b>				
p	p	pepa	'pe.pa	'pe.pa
t	t	tía	'ti.a	'ti.a
k	k	kilo	'ki.lo	'ki.lo
b	b	cambio	'kam.bjo	'kam.bjo
d	d	dedo	'de.Do	'de.ðo
g	g	hangar	an.'gar6	an.'gar



L&H+	IPA	Example	L&H+ Transcription	IPA Transcription
Fricatives				
f	f	fin	'fin	'fin
D	ð	dedo	'de.Do	'de.ðo
s	s	seso	'se.so	'se.so
z	z	desde	'dez.De	'dez.ðe
J	j	mayo	'ma.Jo	'ma.jo
x	x	ajo	'a.xo	'a.xo
G	ɣ	agua	'a.Gwa	'a.ɣwa
B	β	bobo	'bo.Bo	'bo.βo
Affricates				
t&S	tʃ	chico	't&Si.ko	'tʃi.ko
d&Z	dʒ	yo	'd&Zo	'dʒo
Nasals				
m	m	amo	'a.mo	'a.mo
n	n	nene	'ne.ne	'ne.ne
n~	ɲ	niño	'ni.n~o	'ni.ɲo
nK	ŋ	cinco	'sinK.ko	'siŋ.ko
Special				
'	'	primary stress		
.	.	syllable boundary		
_		word boundary		
#		silence (pause)		