

Table 9: Mapping English-language letters

Letter	Phonemes	Example
<i>t</i>	t	<i>time</i>
	!	but tt er, san ity
	D	th em
	T	th in
	tS	ques ti on
	S	na ti on
<i>u</i>	^	u nder, pu tt
	U	pu t , wo u ld
	u	lu na r, ru d e
	j u	u sage, confu se
	*	fo cu s
<i>v</i>	v	ve ry
<i>w</i>	w	w isdom, w hich
<i>x</i>	z	x enophobia
	k s	ax is
<i>y</i>	j	y es
	i	ve ry
	aj	cr y
<i>z</i>	z	z ebra

As a general guideline for generating dictionary entries, ignore the spelling and focus on how the word sounds. Break it down into its component phonemes, sound by sound. Write the codes for those phonemes in the correct sequence. To verify, try to reproduce the sound of the word by reading the symbols in sequence. For example, in the word *backwards*, the sounds (represented by spelling, not phonemes) are as follows:

b, a, ck, w, ar, d, s

The letters are sufficient to tell you what the component sounds are, only because you can still see what the word is, and you know how to pronounce it.

To tell the recognizer that the sounds are spelled b-a-c-k-w-a-r-d-s, you must be more precise about the exact sounds:

- In the Nuance phoneme set, the [b] sound is straightforward, and the phoneme symbol for it is /b/.
- The next sound is a little harder. The letter *a* can represent many different sounds, even in this one word. The English letter table shows that the second sound in *backwards* is the same as the first sound in *apple*, and is represented by the symbol /a/.
- The third sound is represented in the spelling by two letters, *ck*, but the symbol for this phoneme is just /k/.
- The fourth phoneme is represented by the symbol /w/.
- The fifth phoneme is tricky. It is a syllabic *r*, meaning that it has blended with the vowel to make up the root of the syllable, and is therefore one sound. The symbol for this phoneme is /*r/. (In a Boston accent, the *r* part of the phoneme is dropped, and the phoneme is /*/.)
- The symbol for the sixth phone is /d/, and the final phone, which is spelled with an *s*, is actually represented by the symbol /z/, which is what it sounds like.

The dictionary entry for this word, therefore, might look like this:

```
backwards    b a k w *r d z
backwards    b a k w * d z
```

where the second pronunciation indicates a Boston accent.

Note: Be especially careful with words borrowed from other languages, words that would be hard for a child to read, and all vowels. Also, the letter *s* can correspond to many sounds, such as the phoneme /z/ as it does in *busy* and *dogs*. Similarly, the letter *d* can correspond to the phoneme /t/, as in *asked*, *missed*, and *pushed*.

Multiple pronunciations

You can provide multiple pronunciations for any word. The number of pronunciations you want to provide depends on how many dialects you want your system to recognize. List multiple pronunciations on separate lines. For example, the following table lists some examples of words that could have multiple pronunciations:

Table 10: Examples of words with multiple pronunciations

Word	Phonemes
either	aj D *r
	i D *r
compass	k ^ m p * s
	k A m p * s
defense	d * f E n s
	d i f E n s
sandwich	s a n d w I tS
	s a n w I tS

Sample dictionary file

The following lists a sample dictionary made up of the words in Table 9.

able	e b * l
adam	a ! * m
agent	e dZ * n t
alive	* l aj v
aphid	e f * d
apple	a p * l
asked	a s k t
ball	b O l
bird	b *r d
book	b U k
bought	b O t
butter	b ^ ! *r
cat	k a t
caught	k O t
child	tS aj l d
coat	k o t
compass	k ^ m p * s
confuse	k * n f j u z
cow	k aw
cry	k r aj
dogs	d A g z
dogs	d O g z
echo	E k o
even	i v * n
father	f A D *r
focus	f o k * s

golf	g A l f
grammar	g r a m *r
hotel	h o t E l
india	I n d i *
iraq	* r A k
iraq	I r A k
iraq	aj r A k
iraq	* r a k
iraq	I r a k
iraq	aj r a k
ivory	aj v *r i
ivory	aj v r i
jump	dZ ^ m p
key	k i
kissed	k I s t
knot	n A t
law	l O
lisa	l i s *
loop	l u p
lunar	l u n *r
mike	m aj k
nation	n e S * n
new	n u
new	n j u
olive	A l * v
open	o p * n
pan	p a n
pawn	p O n
proceed	p r * s i d
proceed	p r o s i d
pull	p U l
put	p U t
putt	p ^ t
question	k w E s tS * n
question	k w E S tS * n
quick	k w I k
rider	r aj ! *r
ring	r I n~
rude	r u d
sanity	s a n * ! i
shell	S E l
shield	S i l d
shoe	S u
show	S o
soup	s u p
them	D E m
thin	T I n
thumb	T ^ m

time	t a j m
under	^ n d *r
usage	j u s * dZ
usage	j u s I dZ
very	v E r i
vision	v I Z * n
which	w I tS
widen	w a j ! * n
wisdom	w I z d * m
would	w U d
xenophobia	z i n * f o b i *
yes	j E s
zebra	z i b r *

To create pronunciations, you can also use the command-line program *pronounce* to get pronunciations for words that have similar pronunciations. For example, to generate a pronunciation for the word “glyph,” you could base it on the pronunciation for the word “cliff.” To use *pronounce*, specify the master package you are using and a list of one or more words for which you would like to see pronunciations. See the online documentation for *pronounce* for more details.

Converting non-CPA dictionaries

If you have Nuance dictionary files you created with a version of the Nuance System older than 6.2, you need to convert these to CPA. The Nuance System includes the tool *arpabet-to-cpa* that can do this for you. See the online documentation for usage information.

Phoneme sets for other languages

In addition to American English, Nuance supports a variety of other languages and English dialects, including Australian and U.K. English, French, German, Japanese, Brazilian Portuguese, and Latin American Spanish.

To work with any language, you need to use a specific Nuance model set to process that language. See the section “Choosing a model set” on page 33. The *nuance-master-packages* utility will also output a list of the models available on your system.

Tables for mapping the phonemes for all supported languages to their CPA representation are located at the documentation site on the Nuance Developer Network website at extranet.nuance.com.

GSL reference



This appendix provides a definition of the syntax of the Grammar Specification Language (GSL), and a summary of all the package files used to compile a package.

The following conventions are used in the syntax description listed below:

Expression	Meaning
X	A nonterminal
"xyz"	The literal string xyz should actually be typed into the grammar file
X ::= Y	X consists of Y
X Y Z	X or Y or Z
X Y	X followed by Y, with white space allowed
X~Y	X followed by Y with no white space intervening

GSL syntax

The following lists format definitions of the GSL syntax rules:

```
GrammarFile ::= GrammarDefinitions
GrammarDefinitions ::= GrammarDefinition |
                        GrammarDefinition GrammarDefinitions
GrammarDefinition ::= GrammarName GrammarConstruct |
                        GrammarName~":dynamic"GrammarConstruct |
                        GrammarName~":dynaref"GrammarConstruct |
                        GrammarName~":filler"GrammarConstruct
GrammarConstruct ::= BareConstruct | BareConstruct Commands
BareConstruct ::= BareConstructWithoutProb |
                        BareConstructWithoutProb~"~"~Prob
```

```

BareConstructWithoutProb ::= Word
                           GrammarReference
                           "?" GrammarConstruct
                           "*" GrammarConstruct
                           "+" GrammarConstruct
                           "[" GrammarConstructs "]"
                           "(" GrammarConstructs ")"
GrammarConstructs ::= GrammarConstruct
                    GrammarConstruct GrammarConstructs
Word ::= WordChar | WordChar~Word
GrammarReference ::= GrammarName | GrammarName~":"~Variable
GrammarName ::= LimitedString
Commands ::= "{" CommandList "}"
CommandList ::= Command | Command CommandList
Command ::= "<" Slot Value ">"
           "return (" Value ")"
           "insert-begin (" Variable Value ")"
           "insert-end (" Variable Value ")"
           "concat (" Variable Value ")"
Slot ::= SlotString
SlotString ::= VOrFChar | VOrFChar~SlotString
Values ::= Value | Value Values
Value ::= Integer
        LimitedString
        "\""~AnyString~ "\""
        Structure
        "$"~VariableExpression
        List
        FunctionApplication
Structure ::= "[" FeatureValuePairs "]"
FeatureValuePairs ::= FeatureValuePair |
                    FeatureValuePair FeatureValuePairs
FeatureValuePair ::= "<" Feature Value ">"
Feature ::= VariableOrFeatureString
Variable ::= VariableOrFeatureString
VariableExpression ::= Variable | Variable~"."~FeaturePath
FeaturePath ::= Feature | Feature~"."~FeaturePath
List ::= "()" | "(" Values ")"
FunctionApplication ::= Function~ "(" Values ")"
Function ::= LimitedString
WordChar ::= LowerCaseLetter | "-" | "_" | "." | "@" | "'" | Digit
LimitedString ::= LimitedChar | LimitedChar~LimitedString
LimitedChar ::= UpperCaseLetter | LowerCaseLetter | Digit
              | "-" | "_" | "@" | "'" | "."
Digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
VariableOrFeatureString ::= VOrFChar |
                          VOrFChar~VariableOrFeatureString

```

```

VOrFChar ::= UpperCaseLetter | LowerCaseLetter |
           | "-" | "_" | "@" | "'"
AnyString ::= Char | Char~AnyString

```

Notes

- A semicolon indicates a comment. The compiler ignores everything on the line, after a semicolon.
- Grammar names must contain at least one uppercase letter.
- White space is a sequence of at least one of the following characters: space, tab, new line, return, and page feed.
- Char signifies any character other than the double quote or white space.
- Prob denotes a *non negative* floating point number not greater than 1.0.
- The following strings are reserved for internal use and cannot be used to denote a word or grammar name (*n* designates any integer):
AND-*n*, OR-*n*, OP-*n*, KC-*n*, PC-*n*

Summary of package files

The table below lists all the files that *nuance-compile* takes into account when compiling a recognition package, with a brief description of the expected contents.

The common name of these files is the name of the package being compiled. For example, you compile the file *myapp.grammar* to generate a recognition package called *myapp*. All files should be stored in the same directory.

Table 11: Recognition package files

File extension	Required?	Contents
.grammar	Yes	GSL code
.dictionary	No	One word (or compound) followed by a sequence of phonemes per line
.slot_definitions	No	One slot name per line
.functions	No	User-defined functions

Index

SYMBOLS

#include 21, 22

A

acoustic models
 naming conventions 33
 usage 34
acoustic-phonetic models 10
add operator 54
adding NL commands 9
adding probabilities 25
AddPhraseList 77
AddPhraseToDynamicGrammar 77
ambiguity test 62, 66
anticipating responses 6
-app option to *batchrec* 80
 arpabet-to-cpa 94
audio file 75
-auto_pron option to *nuance-compile* 38

B

batchrec 73–84
 commands 76
 Nuance parameters 81
 optional arguments 78
 output 82
 testset example 75
 timeout 80
 usage 74
batchrec commands
 *Echo 75
 *Exit 75
 *GetParam 76
 *NewAudioChannel 76
 *SetParam 76

C

cellular phones 35
CITY grammar 8

CloseDynamicGrammarDatabase 76
commands, natural language 46
 list 58
 return 48
 slot-filling 47
-compare option to *nl-tool* 70
CompileAndInsertDynamicGrammar 78
compiling recognition packages
 see nuance-compile
compound-word modeling 38, 39
Computer Phonetic Alphabet 86
 see CPA
CopyDynamicGrammar 77
coverage test 61, 65
CPA 86–91
 converting to 94
crossword modeling 38
crosswords 41

D

DATE grammar 8
-debug_level option to *batchrec* 81
defining slots 4
DeleteDynamicGrammar 77
dialog
 directed 5
 mixed-initiative 5
dialog design 3
dictionaries
 creating 85–94
 sample 92
dictionary
 example 38
 merge 37
 override 37
 standard 35
dictionary example 92
directed dialog 5
div operator 54
-do_crossword option to *nuance-compile* 41
-dont_endpoint option to *batchrec* 81
-dont_flatten option to *nuance-compile* 29

- dynamic grammar 27
 - dynaref keyword 29
- enrollment 29
- GSL support 28
- referencing 29

E

- Echo, *see batchrec* commands
- endpoint option to *batchrec* 81
- EnrollAbortPhrase 78
- EnrollCommitPhrase 78
- enrollment 29
- EnrollNewPhrase 78
- Exit, *see batchrec* commands

F

- features 56
- filling multiple slots 48
- flight info example 4, 5
- functions 53–55
 - standard 53
 - user-defined 54

G

- gen_ref option to *nl-tool* 70
- generate* 69–70
- generate-nlref* 79
- GetParam, *see batchrec* commands
- grammar
 - core 7
 - definition 16
 - description 16, 17
 - design principles 1
 - dynamic 27
 - example 13
 - example of naturally phrased numbers 21
 - filler 7, 8
 - flattening 40
 - name 16, 17
 - operators 18
 - probabilities 25

- prompt coordination 3
- recursion 24
- subgrammars 20
- testing 63
- top-level 20
- grammar compiler
 - see nuance-compile*
- grammar development 1
 - designing responses 2
 - guidelines 1
 - predicting responses 2
 - tasks 3
- grammar file 16
 - comments 18
 - compound words 38
 - dynamic grammar support 28
 - dynamic keyword 28
 - dynaref keyword 29
 - enrollment support 29
 - grammar naming conventions 17
 - including other files 21
 - lists 57
 - reserved strings 19
 - sample 21
- Grammar keyword 75
- grammar option to *batchrec* 75, 80
- Grammar Specification Language
 - See* GSL
- grammar writing recommendations 30
- GSL 9
 - code example 9
 - enrollment support 29
 - include directive 22
 - probabilities 25
 - reserved strings 97
 - syntax 95
- GSL operators
 - concatenation 18
 - disjunction 18
 - kleene closure 18
 - optional 18
 - positive closure 18

I

- include directive 21

InsertDynamicGrammar 77
integer functions, in natural language
 slots 54
International Phonetic Alphabet 86
interpretation test 61, 66

L

language-specific conventions 30
lists, as interpretation values 57
 commands 58
 functions 57

M

master packages 33
-merge_dictionary option to
 nuance-compile 37
misspellings 37
mixed-initiative dialog 5
ModifyPhrase 78
mul operator 54
multiple pronunciations 91
multiword 38
multiword modeling 38
multiwords and -*dont_flatten option* 38

N

natural language commands 9
natural language interpretation 45–59
 commands 46
 compiling packages 51
 complex slot values 55
 features 56
 lists 57
 return commands 48
 slot commands 47
 variables 48
N-best processing 79
neg operator 54
nested structures 57
NewAudioChannel, *see batchrec*
 commands
NewDynamicGrammarEmpty 76

NewDynamicGrammarFromGSL 76
NewDynamicGrammarFromPhraseList
 77

NGB 13

 assigning probabilities 25
 editing grammar attributes 15
 grammar diagram view 14, 19
 Grammar Library 8, 23
 grammar text view 19
 grammar writing support 19
 include directive 23

NL commands and unary operators 50

NL transcriptions file 79

-nl_transcriptions option to *batchrec* 79

nl-compile 51

nl-tool 70

Nuance Developer Network x

Nuance phoneme set 86

nuance-compile 10, 31

nuance-master-packages 34

-num option to *generate* 69

O

OpenDynamicGrammarDatabase 76

operators

 add 54

 div 54

 mul 54

 neg 54

 strcat 54

 sub 54

over-generation test 62, 67

-override_dictionary option to
 nuance-compile 37

P

package 31–32

package directory 32

package files summary 97

package name convention 33

package optimization 39–41

 graph algorithms 40

 size 40

- unflattened grammars 40
- parse-tool* 69
- phonemes 85–91
 - American English 86
 - mappings to letters 87
- phrase pronunciation 38
- principles of grammar development 1
- print_confidence_scores option to
 - batchrec* 81
- print_trees option to *parse-tool* 69
- print_word_confidence_scores option
 - to *batchrec* 81
- probabilities and
 - expressions 25
 - kleene closure 27
 - optional operator 27
 - positive closure 27
- prompt design 5
- prompt wording 62
- pronounce* 36
- pronunciation test 62
- pronunciations 35–39
 - compound-word 38
 - cross-word 38
 - missing words 37
 - multiple 91
 - multiword 38

Q

- QueryDynamicGrammarContents 77
- QueryDynamicGrammarContentsWithI
 - D 77
- QueryDynamicGrammarExists 77

R

- rc_timeout option to *batchrec* 80
- rcapi option to *batchrec* 80
- rcengine option to *batchrec* 80
- recognition
 - tuning, with *batchrec* 73
- recognition package 10
 - slot definitions files 50
- recognition server 71

- recording
 - with *batchrec* 73
- recursive grammars 24
- referencing variables 50
- regression test 68
- RemovePhrase 77
- return command 48

S

- server 71
- SetParam *see batchrec* commands
- slot definition 4
- slot filling and values 50
- slot name 79
- slot-filling commands 47–50
- slots
 - complex values 55
 - definitions file 50
 - features 56
 - nested structures 57
 - structures 55
- standard dictionary 35
- strcat operator 54
- string functions, in natural language
 - slots 54
- string variable 49
- structures 56
- sub operator 54
- subgrammar 19, 20
- summary of package files 97

T

- technical support x
- test set 62
- testing
 - ambiguity 62
 - command-line programs 68
 - coverage 61
 - interpretation 61
 - over-generation 62
 - pronunciation 62
 - step-by-step 63–68
- testset option to *batchrec* 75

transcriptions file 78
 example 79
-transcriptions option to *batchrec* 78

U

unary operators and NL commands 50
user-defined functions 54

V

variables 50
 assigning 48

 valid characters 49
-vrs option to *batchrec* 80

W

wavconvert 73
word names 17
-write_auto_pron_output option to
 nuance-compile 38

X

Xwavedit 73

