Transactions of the
Bayesian Society

Simulated Feature Evolution using the TKF91 Model

D. M. Goldstein, J. P. Huelsenbeck and S. H. McCreight

Abstract

It all started in a little town called Madrid...

Introduction

In this paper, we attempt to do the impossible!

Methods

We used any and all means necessary.

Conclusion

Vene Vidi Vici

Authors

David M. Goldstein
UCLA, Los Angeles, CA 90095-1543, USA
E-mail: dgoldstein@humnet.ucla.edu

John P. Huelsenbeck
UC Berkeley, 3040 Valley Life Sciences Building #3140, Berkeley, CA 94720-3140, USA
E-mail: johnh@berkeley.edu

Shawn H. McCreight
3060 San Pasqual St., Pasadena, CA 91107, USA
E-mail: shawn.mccreight@gmail.com

Appendices

1) Language Tree
2) Words in each language by meaning
3) Feature Change
4) Character file
5) Segments in the target word list
6) Segment Groups
7) Diacritics
8) Euler Feature Diagram
9) International Phonetic Alphabet
10) Feature Tree
11) SAMPAConversion
12) Word Lists by Language
13) Language Tree File
14) Nytril Source Code

References

Höhna, Landis, Heath, Boussau, Lartillot, Moore, Huelsenbeck, Ronquist, RevBayes: Bayesian phylogenetic inference using graphical models and an interactive model-specification language, 2016, *Systematic Biology*. : http://www.revbayes.com

IPA Symbols Chart Complete, 2019, *InternationalPhoneticAlphabet.org*. : http://www.internationalphoneticalphabet.org/ipa-charts/ipa-symbols-chart-complete

IPA Symbols Chart Complete, 2019, *InternationalPhoneticAlphabet.org*. : http://www.internationalphoneticalphabet.org/ipa-charts/ipa-symbols-chart-complete

International Phonetic Alphabet, 2019, *Wikipedia*. : https://en.wikipedia.org/wiki/International_Phonetic_Alphabet

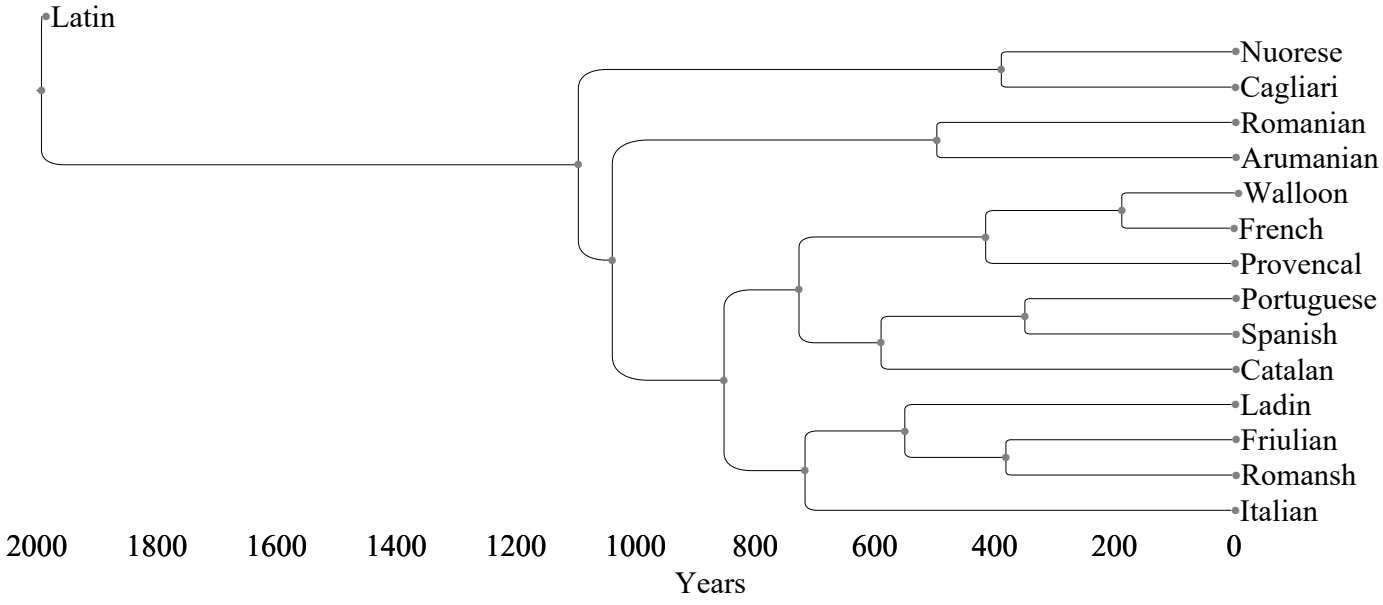
Wichmann, Søren, Eric W. Holman, And Cecil H. Brown (eds.), The ASJP Database (version 18), 2018, *ASJP*. : https://asjp.clld.org

Extended Speech Assessment Methods Phonetic Alphabet, 2016, *Wikipedia*. : https://en.wikipedia.org/wiki/X-SAMPA

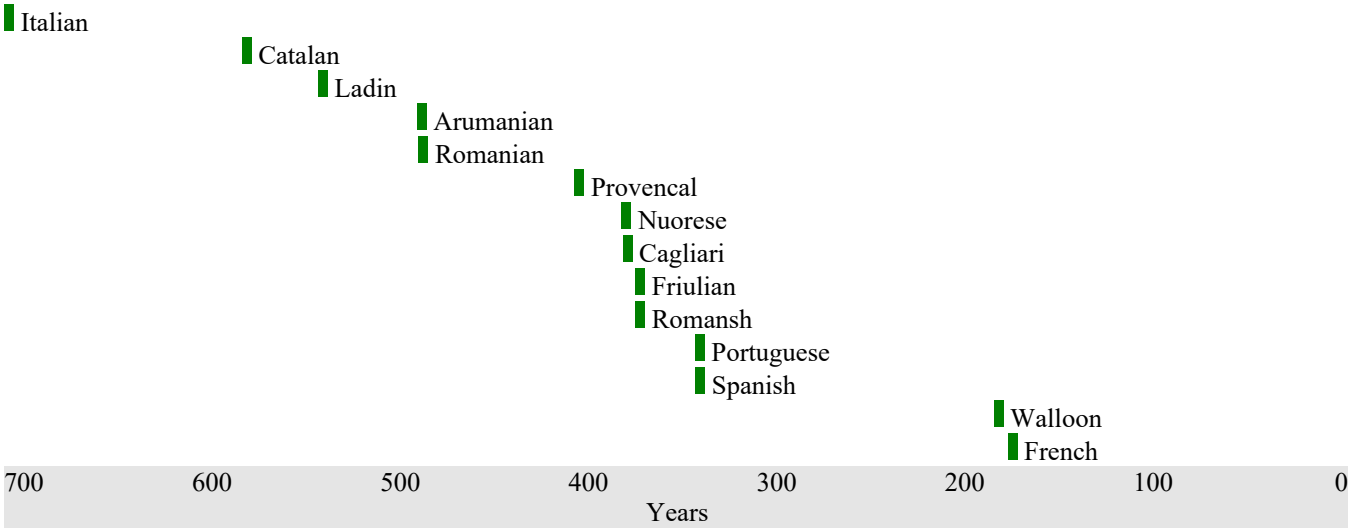
Distinctive Feature, 2020, *Wikipedia*. : https://en.wikipedia.org/wiki/Distinctive_feature

G. A. Lunter, I. Miklós, Y. S. Song, And J. Hein, An Efficient Algorithm for Statistical Multiple Alignment on Arbitrary Phylogenetic Trees, 2003, *Journal Of Computational Biology*

Appendix 1 - Language Tree



Last Branch



Appendix 2 - Words in each language by meaning

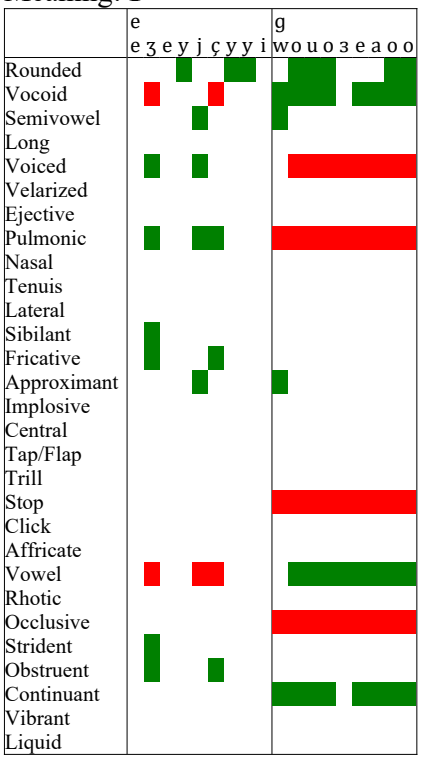
	I	You	We	One	Person	Dog	Skin	Ear
Latin	ego:	tu:	no:s	u:nus	perso:na	kanis	kutis	auris
Romanian	ew	tu	noy	unu	om	kaine	pyele	ureke
Catalan	ʒo	tu	nuzaltrɜs	un	pɜrsonɜ	kɜ	peʌ	urɛʌɜ
Portuguese	eu	tu	noʃ	ũ	pɛrzon	kɐ̃ũ	pɛlɜ	oraʌa
Spanish	yo	tu	nosotros	uno	persona	pero	piel	oreha
French	jɜ	ti	nu	œ̃	om	ʃiɐ̃	po	ore
Walloon	çe	te	nos	ɛ̃	ɔ̃m	çẽ̞	pow	oreye
Romansh	yaw	ti	nus	en	kɜrʃθawn	θawn	pel	ureʌɜ
Friulian	yo	tu	nou	uŋ	pɛrsoŋ	kỹaŋ	pỹel	oreli
Italian	io	tu	noi	uno	persona	kane	pɛlle	orekkyo

	Eye	Drink	Hear	Die	Come	Star	Water	Fire
Latin	okulus	bibere	audi:re	mori:	veni:re	ste:la	ak^{w}a	iŋnis
Romanian	oky	bea	auzy	mury	veny	stea	apɜ	fok
Catalan	uʌ	bɛurɜ	sɜnti	muri	bɜni	ɜstreʌɜ	aixw̃ɜ	fok
Portuguese	oʌu	bɜb	ov	mur	vir	ɜʃtrela	ɛgw̃a	fogu
Spanish	oho	bebe	oir	mori	veni	estreya	agw̃a	fuego
French	ɜy	bw̃a	ɔ̃tɐ̃dr	muri	vɜni	etw̃ol	o	fe
Walloon	ui	bwɛr	ʃute	murrir	vnir	twɛl	ɛwɜ	fɛ
Romansh	eʌ	bayvɜr	udir	murir	vɜɬir	ʃtaylɜ	awɜ	fyew
Friulian	voli	bevi	sintei	murei	viɬei	stelɛ	agɛ	fuk
Italian	okkyo	bere	ud	mor	vɛn	stella	akwa	fwoko

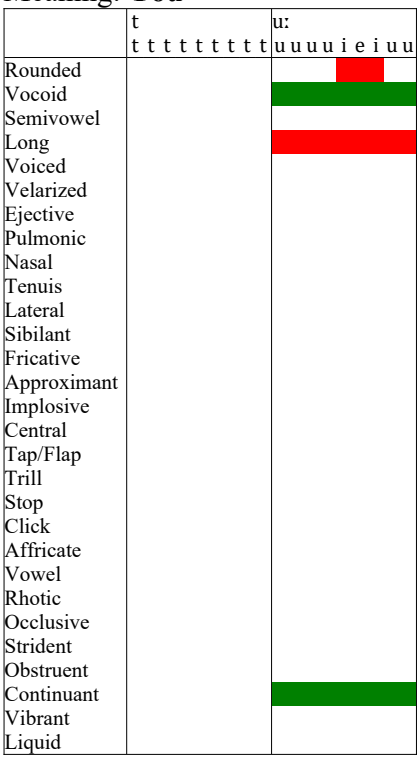
	Path	Full	New
Latin	wia	ple:nus	nowus
Romanian	cale	plin	now
Catalan	kɜmi	plɛ	nou
Portuguese	sẽ̞da	ʃeyu	novu
Spanish	senda	yeno	nuevo
French	rut	plɜ̃	nuvo
Walloon	vwɐ̃y	plĩ	novɛl
Romansh	viɜ	playn	nof
Friulian	stradɛ	plen	ɬuf
Italian	sentyaro	pyɛno	nwovo

Appendix 3 - Feature Change

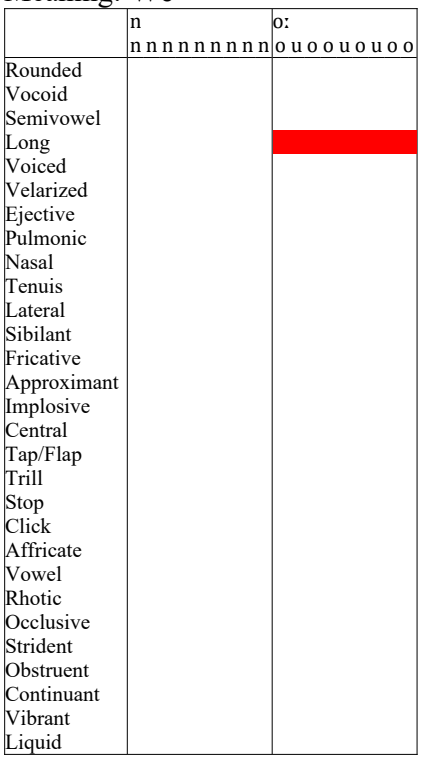
Meaning: **I**



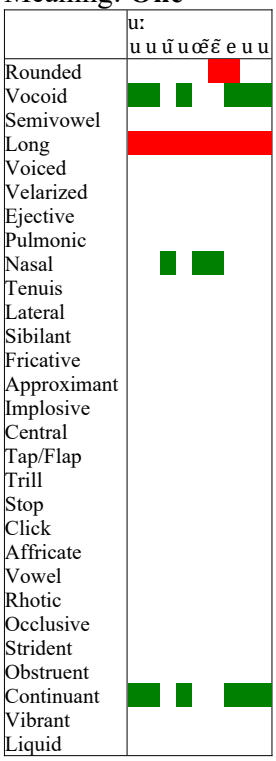
Meaning: **You**



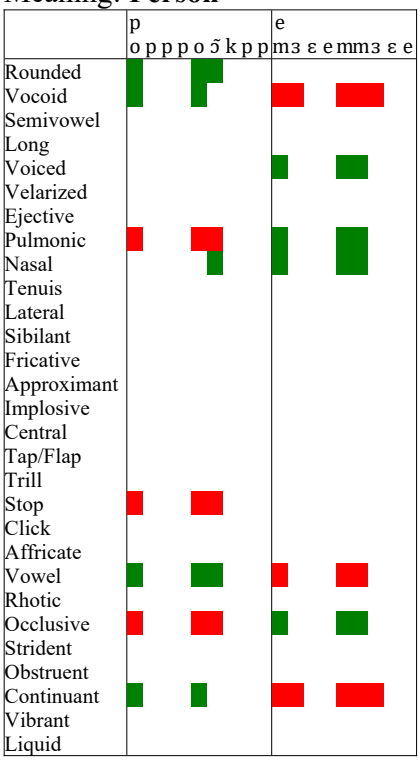
Meaning: **We**



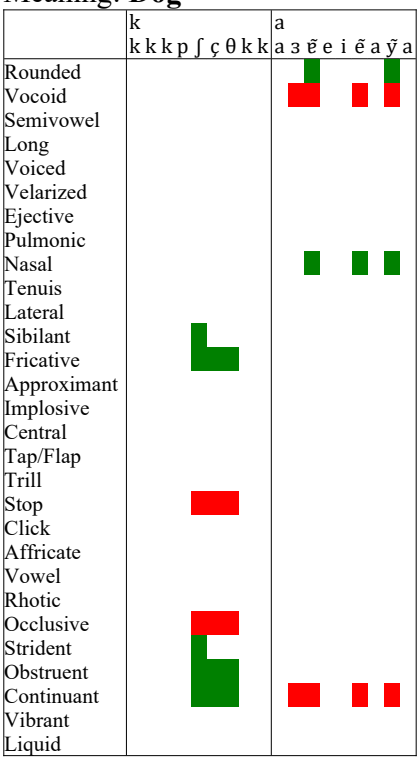
Meaning: **One**



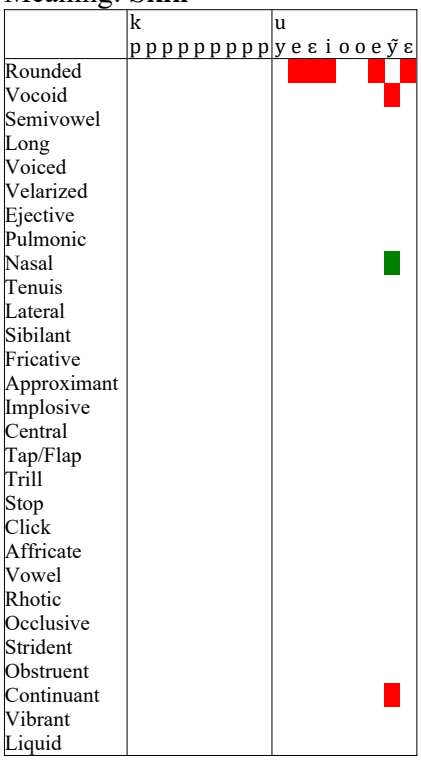
Meaning: **Person**



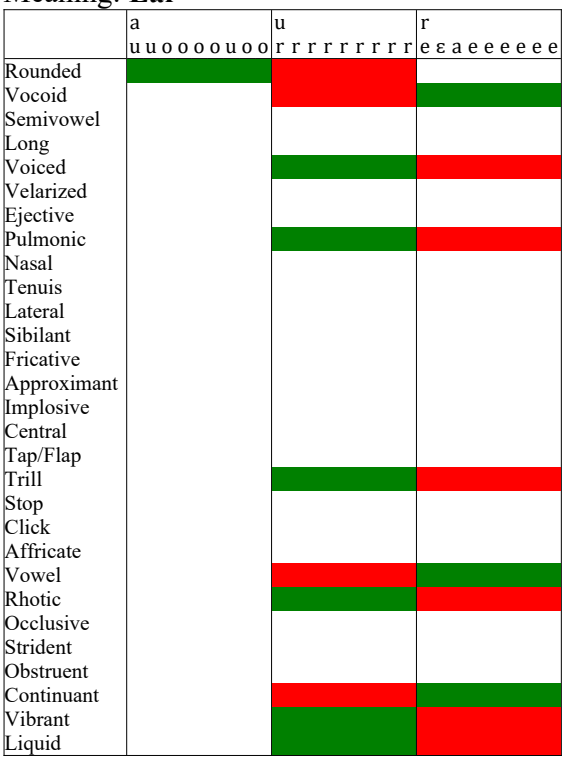
Meaning: **Dog**



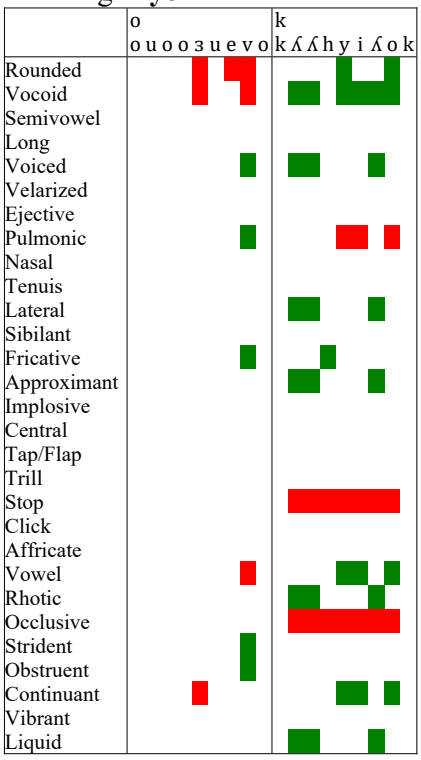
Meaning: **Skin**



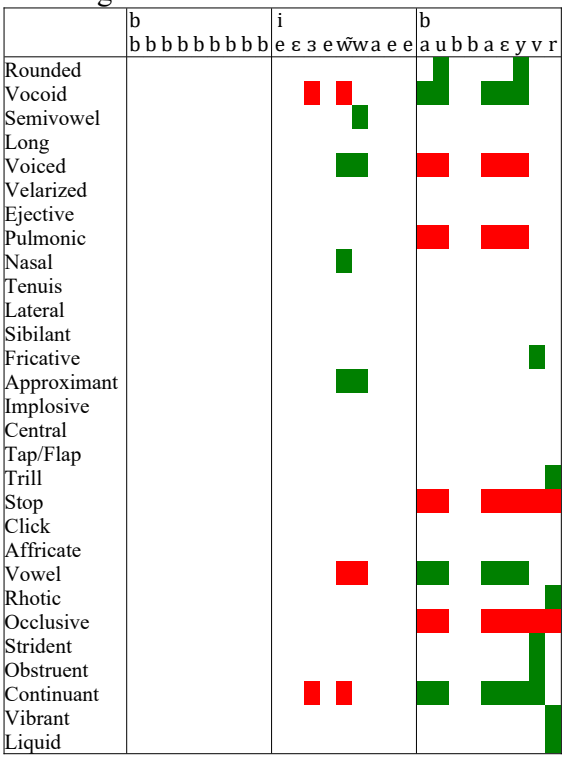
Meaning: **Ear**



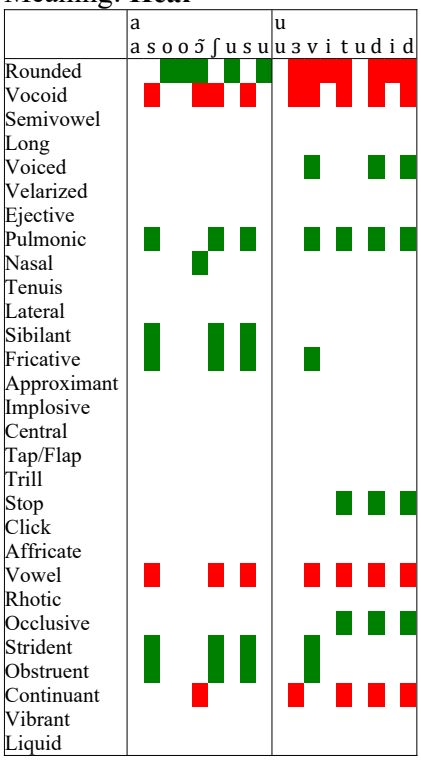
Meaning: **Eye**



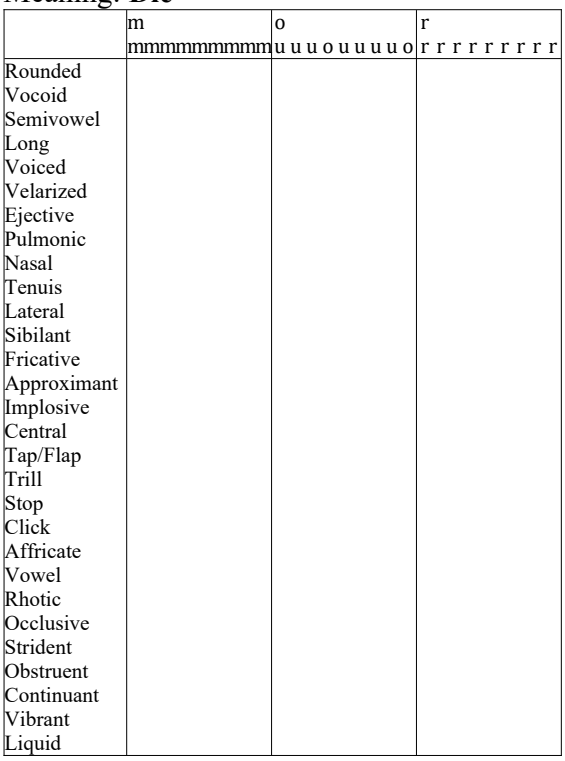
Meaning: **Drink**



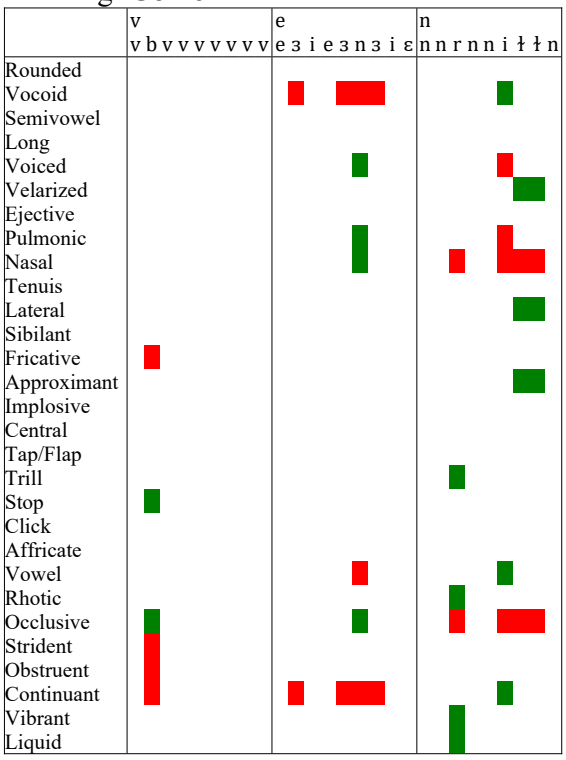
Meaning: **Hear**



Meaning: **Die**

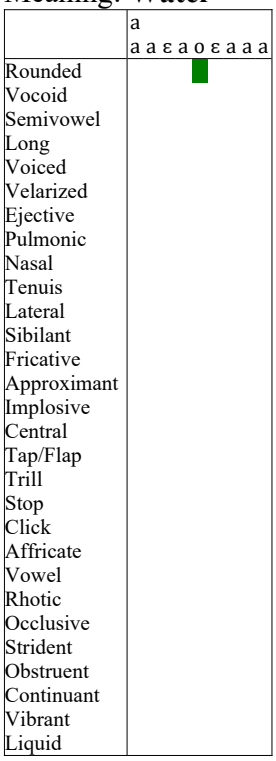


Meaning: **Come**

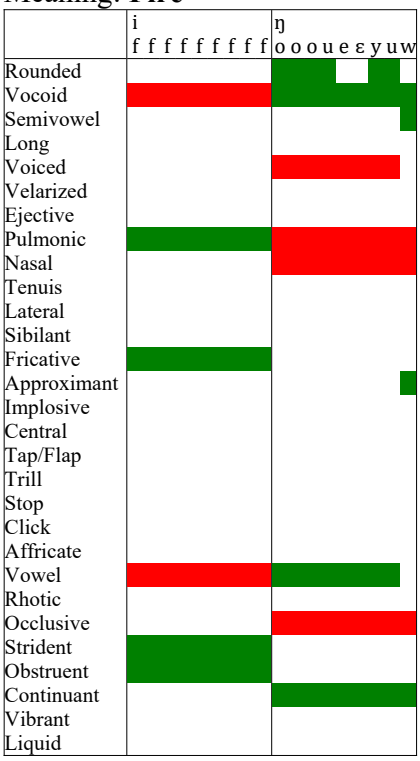


Meaning: **Star**

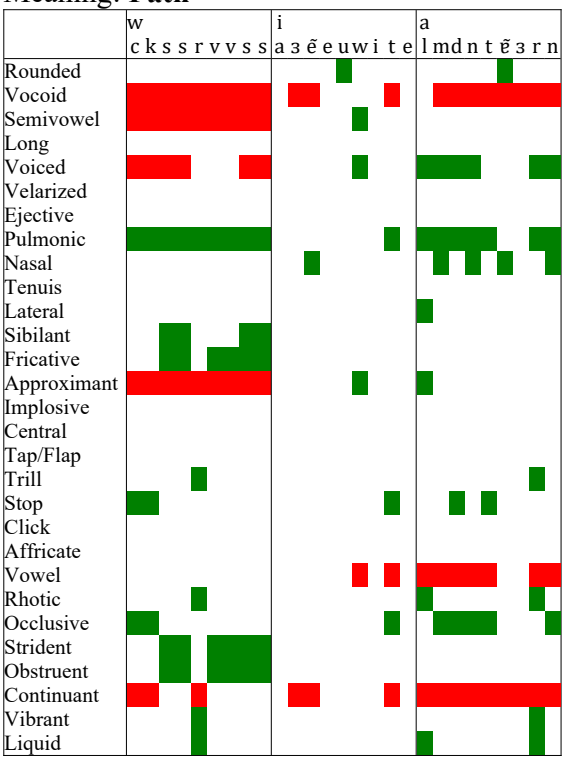
Meaning: **Water**



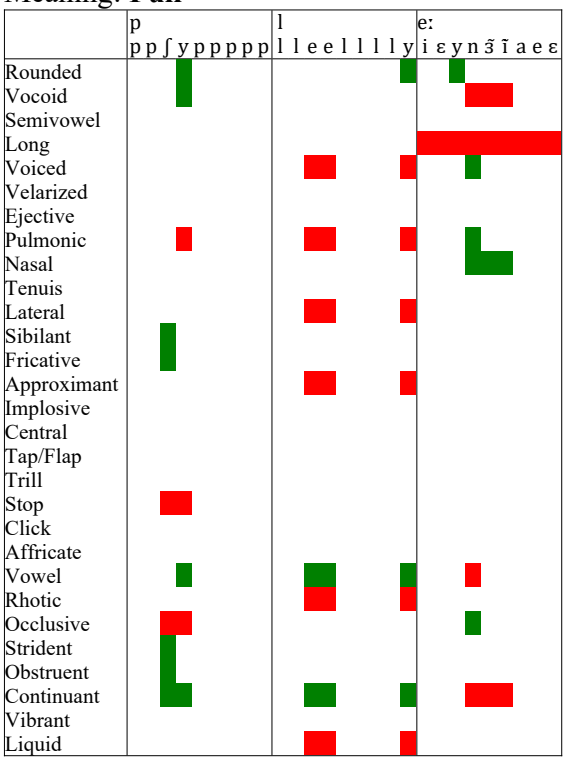
Meaning: **Fire**



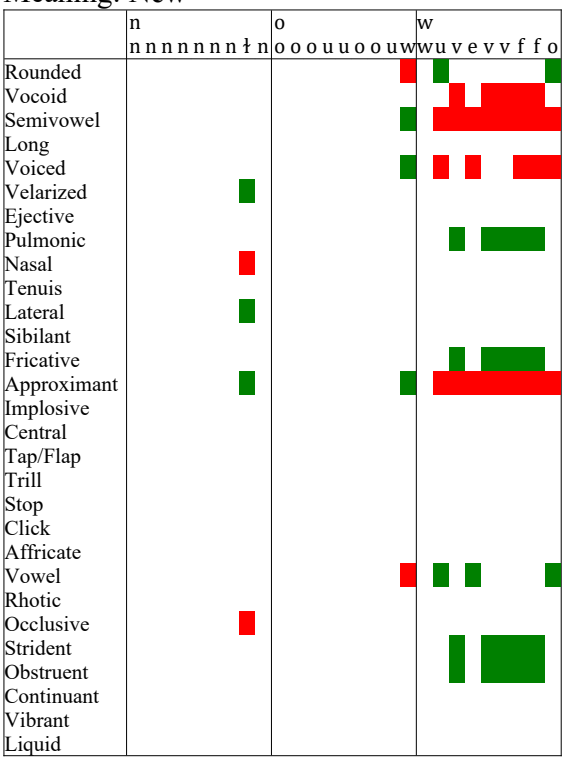
Meaning: **Path**



Meaning: **Full**



Meaning: **New**



```
#NEXUS
begin DATA;
  dimensions ntax=10 nchar=147;
  format datatype=STANDARD gap=- missing=? symbols="ABCDEFGHIJKLMNOPQRSTUVWXYZa
bcdefghijklmnopqrstuvw";

  matrix
  Latin      (ABC)(DE)(FCG------)(EFHG)(IAJGCFK-)(LKFMG)(LHDMG)(KHJMG--)
  Romanian   (AX-)(DH)(FNY------)(HFH-)(NS------)(LKMFA)(IYAOA)(HJALA--)
  Catalan    (dN-)(DH)(FHZKODJaG)(HF--)(IaJGNFa-)(La---)(IAe--)(HJfea--)
  Portuguese (AH-)(DH)(FNi------)(j---)(IfJZNF--)(Lkj--)(IfOa-)(NJKeK--)
  Spanish    (YN-)(DH)(FNGNDJNG-)(HFN-)(IAJGNFK-)(IAJN-)(IMAO-)(NJAmK--)
  French     (na-)(DM)(FH-------)(o---)(NS------)(iMk--)(IN---)(NJA----)
  Walloon    (rA-)(DA)(FNG------)(s---)(pS------)(rl---)(INX--)(NJAYA--)
  Romansh    (YKX)(DM)(FHG------)(AF--)(LaJiuKXF)(uKXF-)(IAO--)(HJAea--)
  Friulian   (YN-)(DH)(FNH------)(HW--)(IfJGNW--)(LwKW-)(IwAO-)(NJAOM--)
  Italian    (MN-)(DH)(FNM------)(HFN-)(IAJGNFK-)(LKFA-)(IfOOA)(NJALLYN)

  Latin      (NLHOHG)(PMPAJA)(KHQRJA)(SNJR--)(TAFRJA)(GDUOK--)(KVK--)(MW
  Romanian   (NLY---)(PAK---)(KHZY--)(SHJY--)(TAFY--)(GDAK---)(KIa--)(bN
  Catalan    (He----)(PfHJa-)(GaFDM-)(SHJM--)(PaFM--)(aGDJAea)(KMgha)(bN
  Portuguese (NeH---)(PaP---)(NT----)(SHJ---)(TMJ---)(aiDJAOK)(fBhK-)(bN
  Spanish    (NmN---)(PAPA--)(NMJ---)(SNJM--)(TAFM--)(AGDJAYK)(KBhK-)(bH
  French     (aY----)(PhK---)(pDkQJ-)(SHJM--)(TaFM--)(ADhNO--)(N----)(bA
  Walloon    (HM----)(PXfJ--)(iHDA--)(SHJJMJ)(TFMJ--)(DXfO---)(fXa--)(bf
  Romansh    (Ae----)(PKYTaJ)(HQMJ--)(SHJMJ-)(TavMJ-)(iDKYOa-)(KXa--)(bY
  Friulian   (TNOM--)(PATM--)(GMFDAM)(SHJAM-)(TMvAM-)(GDAOf--)(KBf--)(bH
  Italian    (NLLYN-)(PAJA--)(HQ----)(SNJ---)(TfF---)(GDAOOK-)(KLXK-)(bX

  Latin      FMG)(XMK-----)(IOUFHG)(FNXHG)
  Romanian   L--)(cKOA----)(IOMF--)(FNX--)
  Catalan    L--)(LaSM----)(IOf---)(FNH--)
  Portuguese BH-)(GlQK----)(iAYH--)(FNTH-)
  Spanish    ABN)(GAFQK---)(YAFN--)(FHATN)
  French     ---)(JHD-----)(IOq---)(FHTN-)
  Walloon    ---)(TXkY----)(IOt---)(FNTfO)
  Romansh    AX-)(TMa-----)(IOKYF-)(FNb--)
  Friulian   L--)(GDJKQf--)(IOAF--)(vHb--)
  Italian    NLN)(GAFDYKJN)(IYfFN-)(FXNTN)

  ;
end;
```

Appendix 5 - Segments in the target word list

Char.	Segment	Words containing this segment
A	e	egoː, persona, bibere, audiːre, veniːre, ew, kaine, pyele, ureke, bea, veny, stea, cale, peʎ, ɜstreʎɜ, eu, ɜʃtrela, ʃeyu, persona, pero, piel, oreha, bebe, veni, estreya, fuego, senda, yen o, nuevo, ore, etw̃ol, fe, çe, te, oreye, ʃute, en, pel, ureʎɜ, eʎ, fyew, pỹel, oreli, bevi, sintei, murei, viɬei, stelɛ, plen, kane, pɛlle, orekkyo, bere, stella, sentyaro
B	g	egoː, ɛgw̃a, fogu, agw̃a, fuego, agɛ
C	oː	egoː, noːs, persona
D	t	tuː, kutis, steːla, tu, stea, nuzaltrɜs, sɜnti, ɜstreʎɜ, ɜʃtrela, nosotros, estreya, ti, ɔ̃tẽdr, etw̃o l, rut, te, ʃute, twɛl, ʃtaylɜ, sintei, stelɛ, stradɛ, stella, sentyaro
E	uː	tuː, uːnus
F	n	noːs, uːnus, persona, kanis, veniːre, iŋnis, pleːnus, nowus, noy, unu, kaine, veny, plin, no w, nuzaltrɜs, un, pɜrsonɜ, sɜnti, bɜni, nou, noʃ, pɛrzon, novu, nosotros, uno, persona, veni , senda, yeno, nuevo, nu, vɜni, nuvo, nos, vnir, novɛl, nus, en, kɜrʃθawn, θawn, playn, nof, sintei, plen, noi, kane, vɛn, sentyaro, pyɛno, nwovo
G	s	noːs, uːnus, persona, kanis, kutis, auris, okulus, steːla, iŋnis, pleːnus, nowus, stea, nuzalt rɜs, pɜrsonɜ, sɜnti, ɜstreʎɜ, sẽda, nosotros, persona, estreya, senda, nos, nus, pɛrsoŋ, sint ei, stelɛ, stradɛ, stella, sentyaro
H	u	uːnus, kutis, auris, okulus, audiːre, pleːnus, nowus, tu, unu, ureke, auzy, mury, nuzaltrɜs, un, urɛʎɜ, uʎ, bɛurɜ, muri, nou, eu, oʎu, mur, fogu, ʃeyu, novu, uno, fuego, nuevo, nu, rut, nuvo, ui, ʃute, murrir, nus, ureʎɜ, udir, murir, uŋ, murei, fuk, ɬuf, ud
I	p	persona, pleːnus, pyele, apɜ, plin, pɜrsonɜ, peʎ, plɛ, pɛrzon, pɛlɜ, persona, pero, piel, po, plɜ̃, pow, plĩ, pel, playn, pɛrsoŋ, pỹel, plen, pɛlle, pyɛno
J	r	persona, auris, bibere, audiːre, moriː, veniːre, ureke, mury, nuzaltrɜs, pɜrsonɜ, urɛʎɜ, bɛ urɜ, muri, ɜstreʎɜ, pɛrzon, oraʎa, mur, vir, ɜʃtrela, nosotros, persona, pero, oreha, oir, mo ri, estreya, ore, ɔ̃tẽdr, rut, oreye, bwɛr, murrir, vnir, kɜrʃθawn, ureʎɜ, bayvɜr, udir, murir, vɜɬir, pɛrsoŋ, oreli, murei, stradɛ, orekkyo, bere, mor, sentyaro
K	a	persona, kanis, auris, audiːre, steːla, ak^{w}a, wia, kaine, bea, auzy, stea, apɜ, cale, nuzaltrɜs , aixw̃ɜ, oraʎa, ɜʃtrela, ɛgw̃a, sẽda, persona, oreha, estreya, agw̃a, senda, bw̃a, yaw, kɜrʃθ awn, θawn, bayvɜr, ʃtaylɜ, awɜ, playn, kỹaŋ, agɛ, stradɛ, kane, stella, akwa, sentyaro
L	k	kanis, kutis, okulus, kaine, ureke, oky, fok, kɜ, kɜmi, kẽũ, kɜrʃθawn, kỹaŋ, fuk, kane, orek kyo, okkyo, akwa, fwoko
M	i	kanis, kutis, auris, bibere, iŋnis, wia, kaine, plin, sɜnti, muri, bɜni, aixw̃ɜ, kɜmi, vir, piel, oi r, mori, veni, ti, ʃiẽ, vɜni, ui, murrir, vnir, udir, murir, vɜɬir, viɜ, oreli, voli, bevi, sintei, mur ei, viɬei, io, noi
N	o	okulus, moriː, nowus, noy, om, oky, fok, now, ʒo, pɜrsonɜ, nou, noʃ, pɛrzon, oraʎa, oʎu, o v, fogu, novu, yo, nosotros, uno, persona, pero, oreha, oho, oir, mori, fuego, yeno, nuevo, po, ore, etw̃ol, o, nuvo, nos, pow, oreye, novɛl, nof, pɛrsoŋ, oreli, voli, io, noi, orekkyo, ok kyo, mor, fwoko, sentyaro, pyɛno, nwovo
O	l	okulus, steːla, pleːnus, pyele, cale, plin, nuzaltrɜs, plɛ, pɛlɜ, ɜʃtrela, piel, etw̃ol, plɜ̃, twɛl, p lĩ, novɛl, pel, ʃtaylɜ, playn, pỹel, oreli, voli, stelɛ, plen, pɛlle, stella
P	b	bibere, bea, bɛurɜ, bɜni, bɜb, bebe, bw̃a, bwɛr, bayvɜr, bevi, bere
Q	d	audiːre, sẽda, senda, ɔ̃tẽdr, udir, stradɛ, ud
R	iː	audiːre, moriː, veniːre
S	m	moriː, om, mury, muri, kɜmi, mur, mori, ɔ̃m, murrir, murir, murei, mor
T	v	veniːre, veny, ov, vir, novu, veni, nuevo, vɜni, nuvo, vnir, vwẽy, novɛl, bayvɜr, vɜɬir, viɜ, v oli, bevi, viɬei, vɛn, nwovo
U	eː	steːla, pleːnus
V	k^{w}	ak^{w}a
W	ŋ	iŋnis, uŋ, pɛrsoŋ, kỹaŋ

X	w	wia, nowus, ew, now, pow, bwɛr, twɛl, ɛwɜ, vwɛ̃y, yaw, kɜrʃθawn, θawn, awɜ, fyew, akwa, fwoko, nwovo
Y	y	noy, pyele, oky, auzy, mury, veny, ʃeyu, yo, estreya, yeno, ɜy, oreye, vwɛ̃y, yaw, bayvɜr, ʃtaylɜ, fyew, playn, orekkyo, okkyo, sentyaro, pyɛno
Z	z	auzy, nuzaltrɜs, pɛrzon
a	ɜ	apɜ, nuzaltrɜs, pɜrsonɜ, kɜ, urɛʌɜ, bɛurɜ, sɜnti, bɜni, ɜstreʌɜ, aixw̃ɜ, kɜmi, pɛlɜ, bɜb, ɜʃtrela, jɜ, ɜy, vɜni, ɛwɜ, kɜrʃθawn, ureʌɜ, bayvɜr, vɜłir, ʃtaylɜ, awɜ, viɜ
b	f	fok, fogu, fuego, fe, fɛ, fyew, nof, fuk, łuf, fwoko
c	c	cale
d	ʒ	ʒo
e	ʌ	peʌ, urɛʌɜ, uʌ, ɜstreʌɜ, oraʌa, oʌu, ureʌɜ, eʌ
f	ɛ	urɛʌɜ, bɛurɜ, plɛ, pɛrzon, pɛlɜ, ɛgw̃a, bwɛr, twɛl, ɛwɜ, fɛ, novɛl, pɛrsoŋ, stelɛ, agɛ, stradɛ, pɛlle, vɛn, pyɛno
g	x	aixw̃ɜ
h	w̃	aixw̃ɜ, ɛgw̃a, agw̃a, bw̃a, etw̃ol
i	ʃ	noʃ, ɜʃtrela, ʃeyu, ʃiẽ, ʃute, kɜrʃθawn, ʃtaylɜ
j	ũ	ũ, kẽũ
k	ẽ	kẽũ, ʃiẽ, ɔ̃tẽdr, vwɛ̃y
l	ɛ̃	sɛ̃da, çɛ̃
m	h	oreha, oho
n	j	jɜ
o	œ̃	œ̃
p	ɔ̃	ɔ̃tẽdr, ɔ̃m
q	ɜ̃	plɜ̃
r	ç	çe, çɛ̃
s	ɛ̃	ɛ̃
t	ĩ	plĩ
u	θ	kɜrʃθawn, θawn
v	ł	vɜłir, viłei, łuf
w	ỹ	kỹaŋ, pỹel

Pulmonic Consonants

Manner	Bilabial	Labial	Labio-Dental	Linguo-Labial	Dental	Alveolar	Post-Alveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal-Epiglottal	Glottal
Nasal	m̥ m		ɱ	n̼		n̥ n		ɳ̊ ɳ	ɲ̊ ɲ	ŋ̊ ŋ	ɴ		
Stop	p b	kʷ gʷ	p̪ b̪	t̼ d̼		t d		ʈ ɖ	c ɟ	k g	q ɢ	ʡ	ʔ
Sibilant, Fricative						s z	ʃ ʒ	ʂ ʐ	ɕ ʑ				
Fricative	ɸ β		f v	θ̼ ð̼	θ ð	θ̱ ð̱	ʃ̝ ʒ̝	ʅ̞	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Approximant			ʋ̥ ʋ			ɻ̥ ɻ		ɻ̊ ɻ	j̊ j	ɯ̥ ɯ			ʔ̞
Tap/Flap	ⱱ̟		ⱱ	ɾ̼		ɾ̥ ɾ		ɽ̊ ɽ			ɢ̆	ʡ̆	
Trill	ʙ̥ ʙ					r̥ r		ɽ̊r̥ ɽr			ʀ	ʜ ʢ	
Lateral, Fricative						ɬ ɮ		ɭ̊˔ ɭ˔	ʎ̝̊ ʎ̝	ʟ̝̊ ʟ̝			
Lateral, Approximant						l̥ l		ɭ̊ ɭ	ʎ̥ ʎ	ʟ̥ ʟ	ʟ̠		
Lateral, Tap/Flap						ɺ		ɭ̆	ʎ̆	ʟ̆			

Shaded areas denote articulations judged to be impossible. Where symbols appear in pairs, the one to the right represents a modally voiced consonant.

Non-Pulmonic Consonants

Manner	Bilabial	Labio-Dental	Dental	Alveolar	Post-Alveolar	Retroflex	Palatal	Velar	Uvular	Pharyngeal-Epiglottal
Ejective, Stop				tʼ		ʈʼ	cʼ	kʼ	qʼ	ʡʼ
Ejective, Fricative	ɸʼ	fʼ	θʼ	sʼ	ʃʼ	ʂʼ	ɕʼ	xʼ	χʼ	
Ejective, Lateral, Fricative				ɬʼ						
Tenuis, Click	ʘ ǀ		ǀ	! ǃ			ǂ ǂ			
Nasal, Click	ʘ̃		ǀ̃	!̃			ǂ̃			
Tenuis, Lateral, Click				ǁ						
Implosive	ɓ̥ ɓ			ɗ̥ ɗ		ᶑ̥ ᶑ	ʄ ʄ	ɠ ɠ	ʛ̥ ʛ	

Shaded areas denote articulations judged to be impossible. Where symbols appear in pairs, the one to the right represents a modally voiced consonant.

Vowels					
	Front	Near-Front	Central	Near-Back	Back
Close	i • y		ɨ • ʉ		ɯ • u
Near-close		ɪ • ʏ	ɪ̈ • ʊ̈	ʊ	
Close-mid	e • ø		ə • ɵ		ɤ • o
Mid	ø̞		ɘ		o̞
Open-mid		ɛ • œ	ɞ • ɜ		ʌ • ɔ
Near-open		æ • œ̞	ɐ		
Open		a • ɶ	ä		ɑ • ɒ

Where symbols appear in pairs, the one to the right represents a rounded vowel.

Long Vowels					
	Front	Near-Front	Central	Near-Back	Back
Close	iː				uː
Mid	eː				oː
Open			aː		

Pulmonic Affricates												
Manner	Bilabial	Labio-Dental	Dental	Alveolar	Alveolo-Palatal	Retroflex	Palatal	Palato-Alveolar	Velar	Uvular	Pharyngeal-Epiglottal	Glottal
Sibilant				ts dz	tɕ dʑ	ʈʂ ɖʐ		tʃ dʒ				
Non-Sibilant	pɸbβ	p̪f b̪v	t̪θ d̪ð	t̪ɹ̝̊ d̪ɹ̝			cç ɟʝ	t̪ɹ̝̊˔d̪ɹ̝˔	kx gɣ	qχ	ʡʢ	ʔh
Lateral				tɬ dɮ		ʈɭ˔	cʎ̝̊		kʟ̝̊ gʟ̝			

Shaded areas denote articulations judged to be impossible.

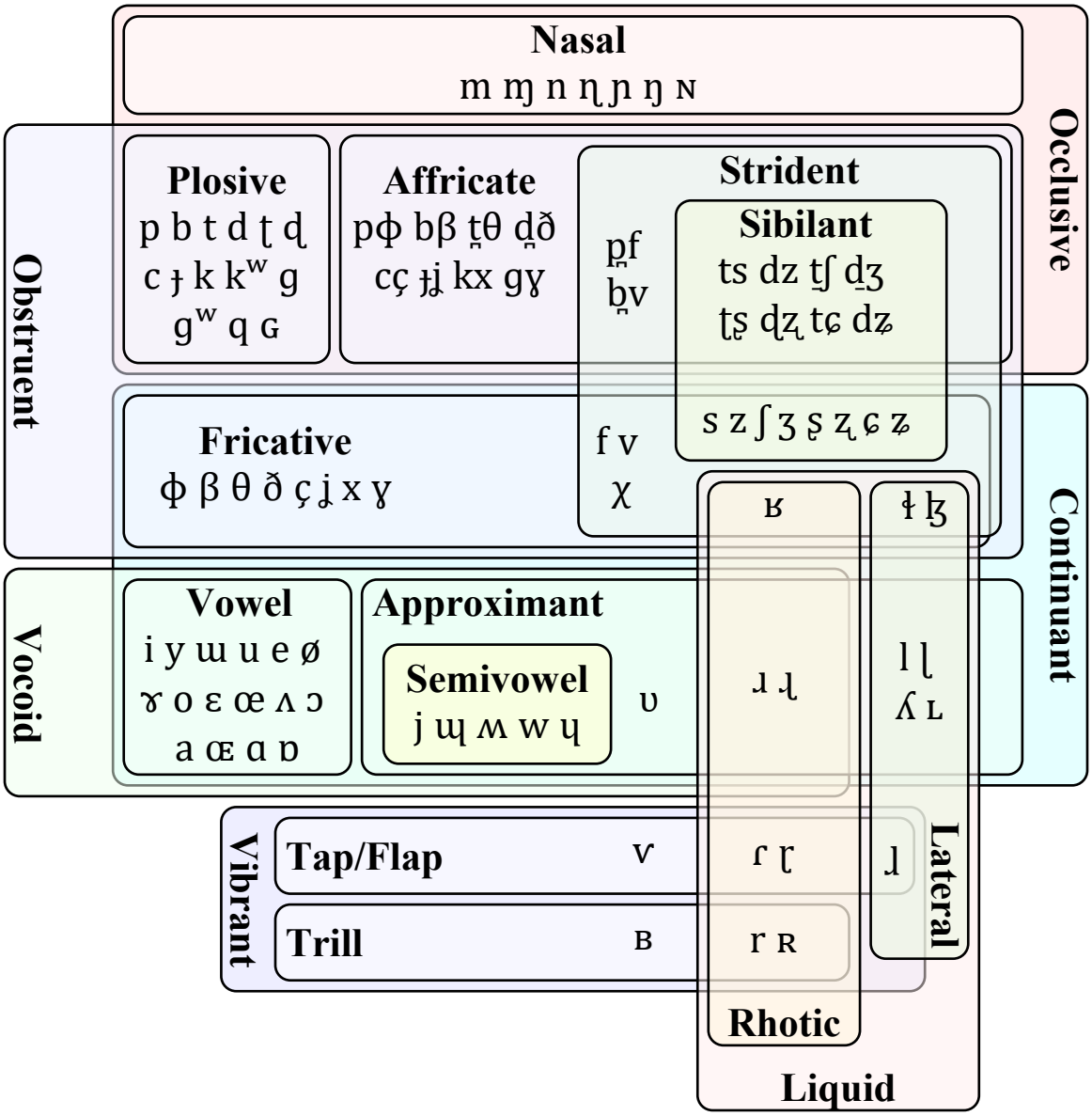
Ejective Affricates										
Manner	Bilabial	Labio-Dental	Alveolar	Retroflex	Palatal	Palato-Alveolar	Velar	Uvular	Pharyngeal-Epiglottal	Glottal
Central			tsʼ	ʈʂʼ		tʃʼ	kxʼ	qχʼ		
Lateral			tɬʼ		cʎ̝̊ʼ		kʟ̝̊ʼ			

Shaded areas denote articulations judged to be impossible.

Other Segments

ɾ̃ Uvular Voiced Pulmonic Nasal Fricative	ɬ Alveolar Voiced Velarized Pulmonic Lateral Approximant
ʍ Labial-Velar Vocoid Semivowel Approximant Continuant	w̃ Labial-Velar Voiced Nasal Approximant
w Labial-Velar Vocoid Semivowel Voiced Approximant Continuant	ɥ Labial-Palatal Vocoid Semivowel Voiced Approximant Continuant
ʃ Post-Alveolar Sibilant Fricative	ʕ Pharyngeal-Epiglottal Voiced Fricative
ʢ Pharyngeal-Epiglottal Ejective	pʼ Bilabial Ejective Pulmonic Stop
œ̃ Open-mid Near-Front Rounded Nasal Vowel	ỹ Close Front Rounded Nasal Vowel
ã Open Near-Front Nasal Vowel	ɐ̃ Near-open Central Rounded Nasal Vowel
ɔ̃ Open-mid Back Rounded Nasal Vowel	ɜ̃ Open-mid Near-Front Nasal Vowel
ɛ̃ Open-mid Near-Front Nasal Vowel	ũ Close Back Rounded Nasal Vowel
ẽ Close-mid Front Nasal Vowel	ĩ Close Front Nasal Vowel

Diacritics					
	Undefined escape character	◌̃	Nasalized	◌̈	Centralized
◌ʰ	Advanced	◌̝	Retracted	◌ˇ	RisingTone
◌̥	Voiceless	◌̩	Implosive	◌̍	Syllabic
◌ʼ	Ejective	◌ˤ	Pharyngealized	◌̂	Falling tone
◌̯	Non-syllabic	◌̚	No audible release	◌̴	Rhotic hook
◌̘	Advanced tongue root	◌̺	Apical	◌̋	Extra low tone
◌˩˥	Low rising tone	◌̜	Less rounded	◌̪	Dental
◌̴	Velarized or Pharyngealized	◌↘	Global fall	◌ˠ	Velarized
◌́	High tone	◌˩	High rising tone	◌ʰ	Aspirated
◌ʲ	Palatalized	◌̰	Creaky voiced	◌̀	Low tone
◌ˡ	Lateral release	◌̄	Mid tone	◌̼	Laminal
◌ⁿ	Linguo-Labial	◌ⁿ	Nasal release	◌̹	More rounded
◌˕	Lowered	◌̙	Retracted tongue root	◌↗	Global rise
◌˦˧˥	Rising falling tone	◌̟	Raised	◌̋	Extra high tone
◌̤	Breathy voiced	◌̬	Voiced	◌ʷ	Labialized
◌̆	Extra short	◌̽	Mid-centralized	◌↓	Down-step
◌↑	Up-step	◌.	Sylable break	◌ˈ	Primary stress
◌ˌ	Secondary stress	◌ː	Long	◌ˑ	Half-long
	Indeterminacy in french vowels		Begin Non-segmental notation		End non-segmental notation
ʢ	Voiced epiglottal fricative	!	Post-alveolar click	|	Minor group
|	Dental click	‖	Major group	‖	Alveolar lateral click
ǂ	Palatal click	◌̥	Voiceless descender	◌̄	Combining macron
◌͡◌	Tie-bar below	◌͡◌	Tie-bar above	ʧ	Ready made combination
→	Becomes		Separator		


Nasal
m ɱ n ɳ ɲ ŋ ɴ
Occlusive
Obstruent
Plosive
p b t d ʈ ɖ
c ɟ k kʷ g
gʷ q ɢ
Affricate
pɸ bβ t̪θ d̪ð
cç ɟʝ kx gɣ
Strident
p̪f
b̪v
Sibilant
ts dz tʃ dʒ
ʈʂ ɖʐ tɕ dʑ
Fricative
ɸ β θ ð ç ʝ x ɣ
f v
χ
s z ʃ ʒ ʂ ʐ ɕ ʑ
Continuant
Vocoid
Vowel
i y ɯ u e ø
ɤ o ɛ œ ʌ ɔ
a ɶ ɑ ɒ
Approximant
Semivowel
j ɰ ʍ w ɥ
ʋ
ʁ
ɻ ɹ
ɬ ɮ
l ɭ
ʏ ʟ
Vibrant
Tap/Flap
ⱱ
ɾ ɽ
ɺ
Lateral
Trill
ʙ
r ʀ
Rhotic
Liquid

IPA Segments

a Open Near-Front Vocoid Vowel Continuant 0061 a	b Bilabial Voiced Pulmonic Stop Occlusive 0062 b	c Palatal Pulmonic Stop Occlusive 0063 c	d Alveolar Voiced Pulmonic Stop Occlusive 0064 d
e Close-mid Front Vocoid Vowel Continuant 0065 e	f Labio-Dental Pulmonic Fricative Strident Obstruent Continuant 0066 f	h Glottal Pulmonic Fricative 0068 h	i Close Front Vocoid Vowel Continuant 0069 i
j Palatal Vocoid Semivowel Voiced Pulmonic Approximant Continuant 006A j	k Velar Pulmonic Stop Occlusive 006B k	l Alveolar Vocoid Voiced Pulmonic Lateral Approximant Rhotic Liquid 006C l	m Bilabial Voiced Pulmonic Nasal Occlusive 006D m
n Alveolar Voiced Pulmonic Nasal Occlusive 006E n	o Close-mid Back Rounded Vocoid Vowel Continuant 006F o	p Bilabial Pulmonic Stop Occlusive 0070 p	q Uvular Pulmonic Stop Occlusive 0071 q
r Alveolar Voiced Pulmonic Trill Rhotic Vibrant Liquid 0072 r	s Alveolar Pulmonic Sibilant Fricative Strident Obstruent Continuant 0073 s	t Alveolar Pulmonic Stop Occlusive 0074 t	u Close Back Rounded Vocoid Vowel Continuant 0075 u

v Labio-Dental Voiced Pulmonic Fricative Strident Obstruent Continuant 0076 v	w Labial-Velar Vocoid Semivowel Voiced Approximant Continuant 0077 w	x Velar Pulmonic Fricative Obstruent Continuant 0078 x	y Close Front Rounded Vocoid Vowel Continuant 0079 y
z Alveolar Voiced Pulmonic Sibilant Fricative Strident Obstruent Continuant 007A z	ä Open Central Vowel 00E4 a_"	æ Near-open Near-Front Vowel 00E6 {	ç Palatal Pulmonic Fricative Obstruent Continuant 00E7 C
ð Dental Voiced Pulmonic Fricative Obstruent Continuant 00F0 D	ø Close-mid Front Rounded Vocoid Vowel Continuant 00F8 2	ħ Pharyngeal-Epiglottal Pulmonic Fricative 0127 X\	ŋ Velar Voiced Pulmonic Nasal Occlusive 014B N
œ Open-mid Near-Front Rounded Vocoid Vowel Continuant 0153 9	ǀ Dental Ejective Tenuis Click Affricate 01C0 |	ǁ Alveolar Ejective Tenuis Lateral Click Affricate 01C1 | \ | \	ǂ Palatal Ejective Tenuis Click Affricate 01C2 =\
! Alveolar Ejective Tenuis Click Affricate 01C3 !\	ɐ Near-open Central Rounded Vowel 0250 6	ɑ Open Back Vocoid Vowel Continuant 0251 A	ɒ Open Back Rounded Vocoid Vowel Continuant 0252 Q

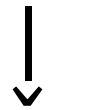
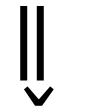
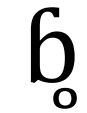
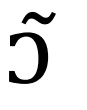
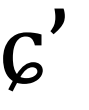
ɓ Bilabial Voiced Ejective Implosive Click Affricate 0253 b_<	ɔ Open-mid Back Rounded Vocoid Vowel Continuant 0254 O	ɕ Palatal Pulmonic Sibilant Fricative Strident Obstruent Continuant 0255 s\	ɖ Retroflex Voiced Pulmonic Stop Occlusive 0256 d`
ɗ Alveolar Voiced Ejective Implosive Click Affricate 0257 d_<	ɘ Mid Central Vowel 0258 @\	ə Close-mid Central Vowel 0259 @	ɛ Open-mid Near-Front Vocoid Vowel Continuant 025B E
ɜ Open-mid Central Vowel 025C 3	ɞ Open-mid Central Rounded Vowel 025E 3\	ɟ Palatal Voiced Pulmonic Stop Occlusive 025F J\	ɠ Velar Voiced Ejective Implosive Click Affricate 0260 g_<
ɡ Velar Voiced Pulmonic Stop Occlusive 0261 g	ɢ Uvular Voiced Pulmonic Stop Occlusive 0262 G\	ɣ Velar Voiced Pulmonic Fricative Obstruent Continuant 0263 G	ɤ Close-mid Back Vocoid Vowel Continuant 0264 7
ɥ Labial-Palatal Vocoid Semivowel Voiced Approximant Continuant 0265 H	ɦ Glottal Voiced Pulmonic Fricative 0266 h\	ɧ Post-Alveolar Sibilant Fricative 0267 x\	ɨ Close Central Vowel 0268 1
ɪ Near-close Near-Front Vowel 026A I	ɬ Alveolar Voiced Velarized Pulmonic Lateral Approximant 026B 5	ɮ Alveolar Pulmonic Lateral Fricative Strident Obstruent Continuant Liquid 026C K	ɭ Retroflex Vocoid Voiced Pulmonic Lateral Approximant Rhotic Liquid 026D n`

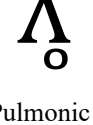
ɬʒ Alveolar Voiced Pulmonic Lateral Fricative Strident Obstruent Continuant Liquid 026E K\	ɯ Close Back Vocoid Vowel Continuant 026F M	ɰɥ Velar Vocoid Semivowel Voiced Pulmonic Approximant Continuant 0270 M\	ɱ Labio-Dental Voiced Pulmonic Nasal Occlusive 0271 F
ɲ Palatal Voiced Pulmonic Nasal Occlusive 0272 J	ɳ Retroflex Voiced Pulmonic Nasal Occlusive 0273 n`	ɴ Uvular Voiced Pulmonic Nasal Occlusive 0274 N\	ɵ Close-mid Central Rounded Vowel 0275 8
œ Near-open Near-Front Rounded Vowel 0276 &	œ Open Near-Front Rounded Vocoid Vowel Continuant 0276 &	ɸ Bilabial Pulmonic Fricative Obstruent Continuant 0278 p\	ɹ Alveolar Vocoid Voiced Pulmonic Approximant Rhotic Liquid 0279 r\
ɺ Alveolar Voiced Pulmonic Lateral Tap/Flap Rhotic Vibrant Liquid 027A l\	ɻ Retroflex Vocoid Voiced Pulmonic Approximant Rhotic Liquid 027B r\`	ɽ Retroflex Voiced Pulmonic Tap/Flap Rhotic Vibrant Liquid 027D r`	ɾ Alveolar Voiced Pulmonic Tap/Flap Rhotic Vibrant Liquid 027E 4
ʀ Uvular Pulmonic Trill Rhotic Vibrant Liquid 0280 R\	ʁ Uvular Voiced Pulmonic Fricative Rhotic Strident Obstruent Continuant Liquid 0281 R	ʂ Retroflex Pulmonic Sibilant Fricative Strident Obstruent Continuant 0282 s`	ʃ Post-Alveolar Pulmonic Sibilant Fricative Strident Obstruent Continuant 0283 S
ʄ Palatal Voiced Ejective Implosive Click Affricate 0284 J_<	ʈ Retroflex Pulmonic Stop Occlusive 0288 t`	ʉ Close Central Rounded Vowel 0289 }	ʊ Near-close Near-Back Rounded Vowel 028A U

 Labio-Dental Vocoid Voiced Pulmonic Approximant 028B v\	 Open-mid Back Vocoid Vowel Continuant 028C V	 Labial-Velar Vocoid Semivowel Approximant Continuant 028D W	 Palatal Vocoid Voiced Pulmonic Lateral Approximant Rhotic Liquid 028E L
 Near-close Near-Front Rounded Vowel 028F Y	 Retroflex Voiced Pulmonic Sibilant Fricative Strident Obstruent Continuant 0290 z`	 Palatal Voiced Pulmonic Sibilant Fricative Strident Obstruent Continuant 0291 z\	 Post-Alveolar Voiced Pulmonic Sibilant Fricative Strident Obstruent Continuant 0292 Z
 Glottal Pulmonic Stop 0294 ?	 Pharyngeal-Epiglottal Voiced Pulmonic Fricative 0295 ?\	 Bilabial Ejective Tenuis Click Affricate 0298 O\	 Bilabial Voiced Pulmonic Trill Vibrant 0299 B\
 Uvular Voiced Ejective Implosive Click Affricate 029B G_<	 Pharyngeal-Epiglottal Pulmonic Trill 029C H\	 Palatal Voiced Pulmonic Fricative Obstruent Continuant 029D j\	 Velar Vocoid Voiced Pulmonic Lateral Approximant Rhotic Liquid 029F L\
 Pharyngeal-Epiglottal Ejective 02A1 <\	 Pharyngeal-Epiglottal Pulmonic Stop 02A1 >\	 Pharyngeal-Epiglottal Voiced Pulmonic Trill 02A2 <\	 Pharyngeal-Epiglottal Voiced Fricative 02A2 ?\
 Bilabial Voiced Pulmonic Fricative Obstruent Continuant 03B2 B	 Dental Pulmonic Fricative Obstruent Continuant 03B8 T	 Uvular Pulmonic Fricative Strident Obstruent Continuant 03C7 X	 Near-close Central Vowel 1D7B I\

ʉ Near-close Central Rounded Vowel 1D7F U\	ɻ Retroflex Voiced Ejective Implosive Click Affricate 1D91 No Sampa	ⱱ Labio-Dental Voiced Pulmonic Tap/Flap Vibrant 2C71 No Sampa	ʟ̆ Velar Voiced Pulmonic Lateral Tap/Flap 004C, 0306 No Sampa
aː Open Central Vocoid Long Vowel Continuant 0061, 02D0 a:	ã Open Near-Front Nasal Vowel 0061, 0303 ~a	b̪ Labio-Dental Voiced Pulmonic Stop 0062, 032A b_d	bβ Bilabial Voiced Pulmonic Affricate Occlusive 0062, 03B2 No Sampa
cç Palatal Pulmonic Affricate Occlusive 0063, 00E7 No Sampa	c’ Palatal Ejective Stop 0063, 02BC c_>	dz Alveolar Voiced Pulmonic Sibilant Affricate Occlusive Strident 0064, 007A No Sampa	dɮ Alveolar Voiced Pulmonic Lateral Affricate 0064, 026E No Sampa
dʑ Alveolo-Palatal Voiced Pulmonic Sibilant Affricate Occlusive Strident 0064, 0291 No Sampa	d̪ Linguo-Labial Voiced Pulmonic Stop 0064, 033C No Sampa	eː Mid Front Vocoid Long Vowel Continuant 0065, 02D0 e:	ẽ Close-mid Front Nasal Vowel 0065, 0303 e*
f’ Labio-Dental Ejective Fricative 0066, 02BC f_>	iː Close Front Vocoid Long Vowel Continuant 0069, 02D0 i:	ĩ Close Front Nasal Vowel 0069, 0303 i*	j̊ Palatal Pulmonic Approximant 006A, 030A No Sampa
kx Velar Pulmonic Affricate Occlusive 006B, 0078 No Sampa	kʷ Labial Pulmonic Stop Occlusive 006B, 02B7 k_W	k’ Velar Ejective Stop 006B, 02BC k_>	l̥ Alveolar Pulmonic Lateral Approximant 006C, 0325 No Sampa

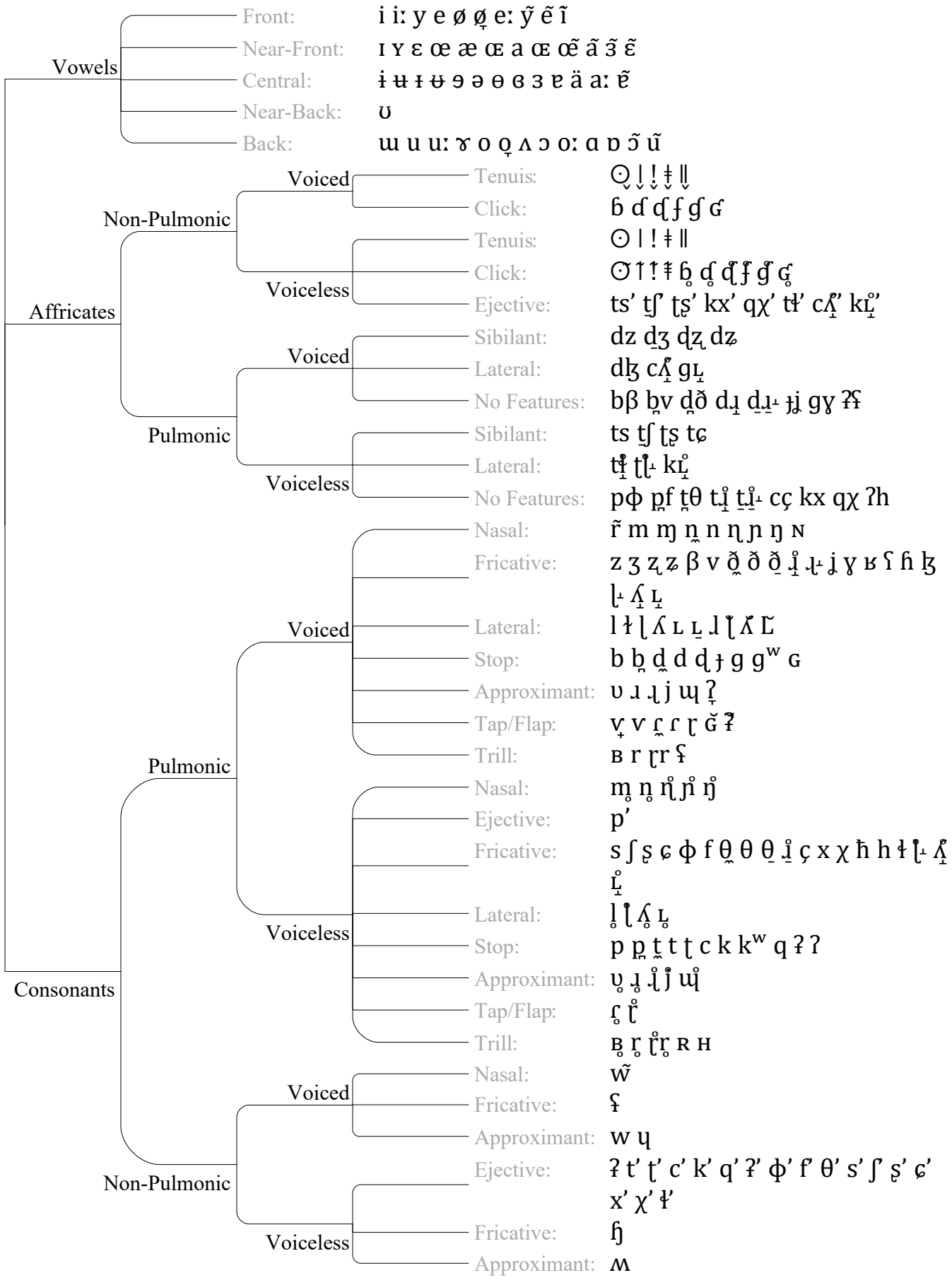
m̥ Bilabial Pulmonic Nasal 006D, 0325 m_0	n̥ Alveolar Pulmonic Nasal 006E, 0325 n_0	n̼ Linguo-Labial Voiced Pulmonic Nasal 006E, 033C m_d	oː Mid Back Rounded Vocoid Long Vowel Continuant 006F, 02D0 o:
o̞ Mid Back Vowel 006F, 031E No Sampa	pɸ Bilabial Pulmonic Affricate Occlusive 0070, 0278 No Sampa	pʼ Bilabial Ejective Pulmonic Stop 0070, 02BC p_>	p̪ Labio-Dental Pulmonic Stop 0070, 032A p_d
qʼ Uvular Ejective Stop 0071, 02BC q_>	qχ Uvular Pulmonic Affricate 0071, 03C7 No Sampa	r̃ Uvular Voiced Pulmonic Nasal Fricative 0072, 0303 r~	r̥ Alveolar Pulmonic Trill 0072, 0325 No Sampa
sʼ Alveolar Ejective Fricative 0073, 02BC s_>	ts Alveolar Pulmonic Sibilant Affricate Occlusive Strident 0074, 0073 No Sampa	tɕ Alveolo-Palatal Pulmonic Sibilant Affricate Occlusive Strident 0074, 0255 No Sampa	tʼ Alveolar Ejective Stop 0074, 02BC t_>
t̼ Linguo-Labial Pulmonic Stop 0074, 033C No Sampa	uː Close Back Rounded Long Vowel 0075, 02D0 u:	ũ Close Back Rounded Nasal Vowel 0075, 0303 u*	w̃ Labial-Velar Voiced Nasal Approximant 0077, 0303 w~
xʼ Velar Ejective Fricative 0078, 02BC x_>	ỹ Close Front Rounded Nasal Vowel 0079, 0303 y~	ð̠ Alveolar Voiced Pulmonic Fricative 00F0, 0320 No Sampa	ð̼ Linguo-Labial Voiced Pulmonic Fricative 00F0, 033C No Sampa

 Mid Front Vowel 00F8, 031E No Sampa	 Velar Pulmonic Nasal 014B, 030A N_0	 Open-mid Near-Front Rounded Nasal Vowel 0153, 0303 oe*	 Dental Ejective Nasal Click Affricate 01C0, 0303 No Sampa
 Dental Voiced Ejective Tenuis Click Affricate 01C0, 032C No Sampa	 Alveolar Voiced Ejective Tenuis Lateral Click Affricate 01C1, 032C No Sampa	 Palatal Ejective Nasal Click Affricate 01C2, 0303 No Sampa	 Palatal Voiced Ejective Tenuis Click Affricate 01C2, 032C No Sampa
 Alveolar Ejective Nasal Click Affricate 01C3, 0303 No Sampa	 Alveolar Voiced Ejective Tenuis Click Affricate 01C3, 032C No Sampa	 Near-open Central Rounded Nasal Vowel 0250, 0303 a*	 Bilabial Ejective Implosive Click Affricate 0253, 0325 No Sampa
 Open-mid Back Rounded Nasal Vowel 0254, 0303 o*	 Palatal Ejective Fricative 0255, 02BC s_>	 Retroflex Voiced Pulmonic Sibilant Affricate Occlusive Strident 0256, 0290 No Sampa	 Alveolar Ejective Implosive Click Affricate 0257, 0325 No Sampa
 Open-mid Near-Front Nasal Vowel 025B, 0303 E*	 Open-mid Near-Front Nasal Vowel 025C, 0303 3*	 Palatal Voiced Pulmonic Affricate Occlusive 025F, 029D No Sampa	 Velar Ejective Implosive Click Affricate 0260, 030A No Sampa
 Velar Voiced Pulmonic Affricate Occlusive 0261, 0263 No Sampa	 Labial Voiced Pulmonic Stop Occlusive 0261, 02B7 g_W	 Uvular Voiced Pulmonic Tap/Flap 0262, 0306 No Sampa	 Alveolar Ejective Lateral Fricative 026C, 02BC K_>

 Retroflex Voiced Pulmonic Lateral Fricative 026D, 02D4 No Sampa	 Retroflex Voiced Pulmonic Lateral Tap/Flap 026D, 0306 No Sampa	 Retroflex Pulmonic Lateral Approximant 026D, 030A No Sampa	 Velar Pulmonic Approximant 0270, 030A No Sampa
 Palatal Pulmonic Nasal 0272, 030A J_0	 Retroflex Pulmonic Nasal 0273, 030A n`_0	 Bilabial Ejective Fricative 0278, 02BC p_>	 Alveolar Pulmonic Approximant 0279, 0325 No Sampa
 Retroflex Voiced Pulmonic Fricative 027B, 02D4 No Sampa	 Retroflex Pulmonic Approximant 027B, 030A No Sampa	 Retroflex Voiced Pulmonic Trill 027D, 0072 No Sampa	 Retroflex Pulmonic Tap/Flap 027D, 030A No Sampa
 Alveolar Pulmonic Tap/Flap 027E, 0325 No Sampa	 Linguo-Labial Voiced Pulmonic Tap/Flap 027E, 033C No Sampa	 Retroflex Ejective Fricative 0282, 02BC s`_>	 Post-Alveolar Ejective Fricative 0283, 02BC S_>
 Palatal Ejective Implosive Click Affricate 0284, 030A No Sampa	 Retroflex Pulmonic Sibilant Affricate Occlusive Strident 0288, 0282 No Sampa	 Retroflex Ejective Stop 0288, 02BC t`_>	 Labio-Dental Pulmonic Approximant 028B, 0325 No Sampa
 Palatal Voiced Pulmonic Lateral Tap/Flap 028E, 0306 No Sampa	 Palatal Voiced Pulmonic Lateral Fricative 028E, 031D No Sampa	 Palatal Pulmonic Lateral Approximant 028E, 0325 No Sampa	 Glottal Pulmonic Affricate 0294, 0068 No Sampa

 Glottal Voiced Pulmonic Approximant 0294, 031E No Sampa	 Bilabial Ejective Nasal Click Affricate 0298, 0303 No Sampa	 Bilabial Voiced Ejective Tenuis Click Affricate 0298, 032C No Sampa	 Bilabial Pulmonic Trill 0299, 0325 No Sampa
 Uvular Ejective Implosive Click Affricate 029B, 0325 No Sampa	 Velar Voiced Pulmonic Lateral Fricative 029F, 031D No Sampa	 Uvular Voiced Pulmonic Lateral Approximant 029F, 0320 No Sampa	 Velar Pulmonic Lateral Approximant 029F, 0325 No Sampa
 Pharyngeal-Epiglottal Voiced Pulmonic Affricate 02A1, 02A2 No Sampa	 Pharyngeal-Epiglottal Ejective Stop 02A1, 02BC >_>	 Pharyngeal-Epiglottal Voiced Pulmonic Tap/Flap 02A1, 0306 No Sampa	 Dental Ejective Fricative 03B8, 02BC T_>
 Alveolar Pulmonic Fricative 03B8, 0320 No Sampa	 Linguo-Labial Pulmonic Fricative 03B8, 033C No Sampa	 Uvular Ejective Fricative 03C7, 02BC X_>	 Retroflex Ejective Implosive Click Affricate 1D91, 030A No Sampa
 Bilabial Voiced Pulmonic Tap/Flap 2C71, 031F No Sampa	 Labio-Dental Voiced Pulmonic Affricate Occlusive Strident 0062, 032A, 0076 No Sampa	 Alveolar Voiced Pulmonic Affricate 0064, 0279, 031D No Sampa	 Palato-Alveolar Voiced Pulmonic Sibilant Affricate Occlusive Strident 0064, 0320, 0292 No Sampa
 Dental Voiced Pulmonic Affricate Occlusive 0064, 032A, 00F0 No Sampa	 Velar Ejective Central Affricate 006B, 0078, 02BC No Sampa	 Labio-Dental Pulmonic Affricate Occlusive Strident 0070, 032A, 0066 No Sampa	 Uvular Ejective Central Affricate 0071, 03C7, 02BC No Sampa

ts’ Alveolar Ejective Central Affricate 0074, 0073, 02BC No Sampa	tɬ’ Alveolar Ejective Lateral Affricate 0074, 026C, 02BC No Sampa	t͡ʃ Palato-Alveolar Pulmonic Sibilant Affricate Occlusive Strident 0074, 0320, 0283 No Sampa	t̪θ Dental Pulmonic Affricate Occlusive 0074, 032A, 03B8 No Sampa
gʟ̝ Velar Voiced Pulmonic Lateral Affricate 0261, 029F, 031D No Sampa	ɭ˔ Retroflex Pulmonic Lateral Fricative 026D, 030A, 02D4 No Sampa	ʝ̟ Post-Alveolar Voiced Pulmonic Fricative 0279, 031D, 030A No Sampa	ʝ̟ Post-Alveolar Pulmonic Fricative 0279, 0320, 030A No Sampa
ʈʂ’ Retroflex Ejective Central Affricate 0288, 0282, 02BC No Sampa	ʎ̝ Palatal Pulmonic Lateral Fricative 028E, 031D, 030A No Sampa	ʟ̝ Velar Pulmonic Lateral Fricative 029F, 031D, 030A No Sampa	cʎ̝ Palatal Voiced Pulmonic Lateral Affricate 0063, 028E, 031D, 030A No Sampa
kʟ̝ Velar Pulmonic Lateral Affricate 006B, 029F, 031D, 030A No Sampa	tɬ̪ Alveolar Pulmonic Lateral Affricate 0074, 026C, 031D, 030A No Sampa	tɻ̟ Alveolar Pulmonic Affricate 0074, 0279, 031D, 030A No Sampa	t͡ʃˀ Palato-Alveolar Ejective Central Affricate 0074, 0320, 0283, 02BC No Sampa
ʈr̥ Retroflex Pulmonic Trill 027D, 030A, 0072, 0325 No Sampa	tɭ˔ Retroflex Pulmonic Lateral Affricate 0288, 026D, 030A, 02D4 No Sampa	cʎ̝’ Palatal Ejective Lateral Affricate 0063, 028E, 031D, 030A, 02BC No Sampa	d̠ɹ̠˔ Palato-Alveolar Voiced Pulmonic Affricate 0064, 0320, 0279, 0320, 02D4 No Sampa
kʟ̝’ Velar Ejective Lateral Affricate 006B, 029F, 031D, 030A, 02BC No Sampa	t̠ɹ̠˔ Palato-Alveolar Pulmonic Affricate 0074, 0320, 0279, 0320, 030A, 02D4 No Sampa		


Vowels
Front: i iː y e ø ø̟ eː ỹ ẽ ĩ
Near-Front: ɪ ʏ ɛ œ æ ɶ a ɶ̟ œ̃ ã ɜ̃ ɛ̃
Central: ɨ ʉ ɪ̈ ʊ̈ ɘ ə ɵ ɞ ɜ ɐ ä aː ɐ̃
Near-Back: ʊ
Back: ɯ u uː ɤ o o̟ ʌ ɔ oː ɑ ɒ ɔ̃ ũ
Consonants
Affricates
Non-Pulmonic
Voiced
Tenuis: ʘ̬ ǀ̬ ǃ̬ ǂ̬ ǁ̬
Click: ɓ ɗ ᶑ ʄ ɠ ɢ
Voiceless
Tenuis: ʘ ǀ ǃ ǂ ǁ
Click: ʘ̹ ǀ̹ ǃ̹ ǂ̹ ɓ̥ ɗ̥ ᶑ̊ ʄ̊ ɠ̊ ɢ̊
Ejective: tsʼ tʃʼ ʈʂʼ kxʼ qχʼ tɬʼ cʎ̝ʼ kʟ̝ʼ
Pulmonic
Voiced
Sibilant: dz d̻ʒ ɖʐ dʑ
Lateral: dɮ cʎ̝ gʟ̝
No Features: bβ b̪v d̪ð dɹ̝ d̻ɹ̝ˠ ɟʝ gɣ ʔɬ
Voiceless
Sibilant: ts t̻ʃ ʈʂ tɕ
Lateral: tɬ̪ ʈɭˠ kʟ̝̊
No Features: pɸ p̪f t̪θ t̻ɹ̝̊ t̻ɹ̝̊ˠ cç kx qχ ʔh
Pulmonic
Voiced
Nasal: ɾ̃ m ɱ n̪ n ɳ ɲ ŋ ɴ
Fricative: z ʒ ʐ ʑ β v ð̪ ð θ̬̪ ɹ̝̊ ɹ̝̊ˠ ɻ ɣ ʁ ʕ ɦ ɮ ɭˠ ʎ̝ ʟ̝
Lateral: l ɬ ɭ ʎ̝ ʟ ʟ̪ ɺ ɭ̆ ʎ̝̆ ʟ̆
Stop: b b̪ d̪ d ɖ ɟ g gʷ ɢ
Approximant: ʋ ɹ ɹ̠ j ɰ ʔ̞
Tap/Flap: ⱱ̟ ⱱ ɾ̪ ɾ ɽ ɢ̆ ʔ̆
Trill: ʙ r ɽr ʢ
Voiceless
Nasal: m̥ n̥ ɳ̊ ɲ̊ ŋ̊
Ejective: pʼ
Fricative: s ʃ ʂ ɕ ɸ f θ̪ θ θ̱ ɹ̝̊ ç x χ ħ h ɬ ɭˠ ʎ̝̊ ʟ̝̊
Lateral: l̥ ɭ̊ ʎ̝̥ ʟ̥
Stop: p p̪ t̪ t ʈ c k kʷ q ʡ ʔ
Approximant: ʋ̥ ɹ̥ ɹ̠̊ j̊ ɰ̊
Tap/Flap: ɾ̥ ɽ̊
Trill: ʙ̥ r̥ ɽ̊r̥ ʀ ʜ
Non-Pulmonic
Voiced
Nasal: w̃
Fricative: ʢ
Approximant: w ɥ
Voiceless
Ejective: ʡ tʼ ʈʼ cʼ kʼ qʼ ʡʼ ɸʼ fʼ θʼ sʼ ʃ ʂʼ ɕʼ xʼ χʼ ɬʼ
Fricative: ɧ
Approximant: ʍ

SAMPA	IPA	Segment
		SpaceSegment
&	œ	FrontOpenRounded
1	ɨ	CloseCentralUnrounded
2	ø	CloseMidFrontRounded
3	ɜ	aeh
4	ɾ	VdAlveolarTap
5	ɬ	ssha
6	ɐ	OpenMidSchwa
7	ɤ	CloseMidBackUnrounded
8	ɵ	ooh
9	œ	OpenMidNearFrontRounded
?	ʔ	GlottalStop
@	ə	Schwa
A	ɑ	OpenBackUnrounded
B	β	VdBilabialFricative
C	ç	sh
D	ð	VdDentalFricative
E	ɛ	eh
F	ɱ	VdLabioDentalNasal
G	ɣ	VdVelarFricative
H	ɥ	VdLabialPalatalApproximant
I	ɪ	NearCloseFrontUnrounded
J	ɲ	VdPalatalNasal
K	ɬ	VlAlveolarLateralFricative
L	ʎ	yuh
M	ɯ	CloseBackUnrounded
N	ŋ	nya
O	ɔ	OpenMidBackRounded
Q	ɒ	OpenBackRounded
R	ʁ	VdUvularFricative
S	ʃ	shh
T	θ	th
U	ʊ	NearCloseBackRounded
V	ʌ	OpenMidBackUnrounded
W	ʍ	VlLabialVelarApproximant
X	χ	VlUvularFricative
Y	ʏ	NearCloseFrontRounded
Z	ʒ	gzah
a	a	ah
b	b	b
c	c	tya
d	d	d
e	e	ay
f	f	f
g	g	g
h	h	h
i	i	e
j	j	jg
k	k	k

l	l	l
m	m	m
n	n	n
o	o	oh
p	p	p
q	q	VlUvularStop
r	r	r
s	s	s
t	t	t
u	u	u
v	v	v
w	w	w
x	x	xha
y	y	eeh
z	z	zz
{	æ	NearFrontUnrounded
|	ǀ	VlDentalTenuisClick
}	ʉ	CloseCentralRounded
!\	!	VlAlveolarTenuisClick
3*	ɜ̃	aehn
3\	ɞ	OpenMidCentralRounded
<\	ʢ	VdPharyngealTrill
=\	ǂ	VlPalatalTenuisClick
>\	ʡ	VdEpiglottalStop
?\	ʕ	VdPharyngealFricative
@\	ə	MidCentralUnrounded
B\	ʙ	VdBilabialTrill
E*	ɛ̃	ehnn
G\	ɢ	VdUvularStop
H\	ʜ	VlPharyngealTrill
I\	ɪ	NearCloseCentralUnrounded
J\	ɟ	VdPalatalStop
K\	ɮ	VdAlveolarLateralFricative
L\	ʟ	VdVelarLateral
M\	ɯ̟	VdVelarApproximant
N\	ɴ	VdUvularNasal
O\	ʘ	VlBilabialTenuisClick
R\	ʀ	VlUvularTrill
U\	ʊ	NearCloseCentralRounded
X\	ħ	VlPharyngealFricative
a*	ɐ̃	ahn
a:	aː	A
d`	ɖ	VdRetroflexStop
e*	ẽ	en
e:	eː	ai
h\	ɦ	VdGlottalFricative
i*	ĩ	een
i:	iː	E
j\	ʝ	VdPalatalFricative
l\	ɺ	VdAlveolarLateralFlap
n`	ɳ	VdRetroFlexNasal
o*	ɔ̃	oon

o:	oː	O
p\	ɸ	VlBilabialFricative
r\	ɹ	VdPostalveolarApproximant
r`	ɽ	VdRetroflexFlap
r~	ɾ̃	rn
s\	ɕ	VlPalatalSibFricative
s`	ʂ	VlRetroflexSibFricative
t`	ʈ	VlRetroflexStop
u*	ʉ̃	uh
u:	uː	U
v\	ʋ	VdLabioDentalApproximant
w~	w̃	wh
x\	ɧ	SimultaneousSx
y~	ỹ	ey
z\	ʑ	VdPalatalSibFricative
z`	ʐ	VdRetroflexSibFricative
~a	ã	aa
J_0	ɲ̊	VlPalatalNasal
K_>	ɬ’	VlPostalveolarLatFricEjective
N_0	ŋ̊	VlVelarNasal
S_>	ʃ’	VlPostalveolarFricativeEjective
T_>	θ’	VlDentalFricativeEjective
X_>	χ’	VlUvularFricativeEjective
a_"	ä	OpenCentralUnrounded
b_<	ɓ	VdBilabialImplosiveClick
b_d	b̪	VdLabioDentalStop
c_>	c’	VlPalatalStopEjective
d_<	ɗ	VdAlveolarImplosiveClick
f_>	f’	VlLabiodentalFricativeEjective
g_<	ɠ	VdVelarImplosiveClick
g_W	g^w	gw
k_>	k’	VlVelarStopEjective
k_W	k^w	kw
m_0	m̥	VlBilabialNasal
m_d	n̪	VdLinguoLabioNasal
n_0	n̥	VlAlveolarNasal
oe*	œ̃	uuh
p_>	p’	VlBilabialStopEjective
p_d	p̪	VlLabioDentalStop
q_>	q’	VlUvularStopEjective
r\`	ɻ	VdRetroflexApproximant
s_>	s’	VlAlveolarFricativeEjective
t_>	t’	VlAveolarStopEjective
x_>	x’	VlVelarFricativeEjective
>_>	ʔ’	VlEpiglottalStopEjective
G_<	ɢ	VdUvularImplosiveClick
J_<	ʄ	VdPalatalImplosiveClick
n`_0	ɳ̊	VlRetroFlexNasal
p_>	ɸ’	VlBilabialFricativeEjective
s_>	ɕ’	VlPalatalFricativeEjective
s`_>	ʂ’	VlRetroflexFricativeEjective
t`_>	ʈ’	VlRetroflexStopEjective

ǀ \ ǀ \ ǁ	VlAlveolarTenuisLateralClick

Latin

Meaning	SAMPA	IPA	Sounds
I	ego:	egoː	ay-g-O
You	tu:	tuː	t-U
We	no:s	noːs	n-O-s
One	u:nus	uːnus	U-n-u-s
Two	duo	duo	d-u-oh
Person	perso:na	persoːna	p-ay-r-s-O-n-ah
Fish	piskis	piskis	p-e-s-k-e-s
Dog	kanis	kanis	k-ah-n-e-s
Louse	pedikulus	pedikulus	p-ay-d-e-k-u-l-u-s
Tree	arbor	arbor	ah-r-b-oh-r
Leaf	foly~u*	folỹũ	f-oh-l-ey-uh
Skin	kutis	kutis	k-u-t-e-s
Blood	sang_Wis	sang^{w}is	s-ah-n-gw-e-s
Bone	o:s	oːs	O-s
Horn	kornu:	kornuː	k-oh-r-n-U
Ear	auris	auris	ah-u-r-e-s
Eye	okulus	okulus	oh-k-u-l-u-s
Nose	na:sus	naːsus	n-A-s-u-s
Tooth	de:ns	deːns	d-ai-n-s
Tongue	liNgw~E	liŋgw̃ɛ	l-e-nya-g-wh-eh
Knee	genu:	genuː	g-ay-n-U
Hand	manus	manus	m-ah-n-u-s
Breast	pektus	pektus	p-ay-k-t-u-s
Breast	mama	mama	m-ah-m-ah
Liver	jekur	jekur	jg-ay-k-u-r
Drink	bibere	bibere	b-e-b-ay-r-ay
See	wide:re	wideːre	w-e-d-ai-r-ay
Hear	audi:re	audiːre	ah-u-d-E-r-ay
Die	mori:	moriː	m-oh-r-E
Come	veni:re	veniːre	v-ay-n-E-r-ay
Sun	so:5	soːɬ	s-O-ssha
Star	ste:la	steːla	s-t-ai-l-ah
Water	ak_Wa	ak^{w}a	ah-kw-ah
Stone	lapis	lapis	l-ah-p-e-s
Fire	iNnis	iŋnis	e-nya-n-e-s
Path	wia	wia	w-e-ah
Mountain	mo:ns	moːns	m-O-n-s
Night	noks	noks	n-oh-k-s
Full	ple:nus	pleːnus	p-l-ai-n-u-s
New	nowus	nowus	n-oh-w-u-s
Name	no:men	noːmen	n-O-m-ay-n

Romanian

Meaning	SAMPA	IPA	Sounds
I	ew	ew	ay-w
You	tu	tu	t-u
We	noy	noy	n-oh-eeh
One	unu	unu	u-n-u
Two	doy	doy	d-oh-eeh
Person	om	om	oh-m
Fish	peSte	peʃte	p-ay-shh-t-ay
Dog	kaine	kaine	k-ah-e-n-ay
Louse	paduke	paduke	p-ah-d-u-k-ay
Tree	arbore	arbore	ah-r-b-oh-r-ay
Tree	pom	pom	p-oh-m
Leaf	frunz3	frunzɜ	f-r-u-n-zz-aeh
Skin	pyele	pyele	p-eeh-ay-l-ay
Blood	s3nje	sɜnje	s-aeh-n-jg-ay
Bone	os	os	oh-s
Horn	korn	korn	k-oh-r-n
Ear	ureke	ureke	u-r-ay-k-ay
Eye	oky	oky	oh-k-eeh
Nose	nas	nas	n-ah-s
Tooth	dinte	dinte	d-e-n-t-ay
Tongue	limb3	limbɜ	l-e-m-b-aeh
Knee	jenuNky	jenuŋky	jg-ay-n-u-nya-k-eeh
Hand	m3n3	mɜnɜ	m-aeh-n-aeh
Breast	s3n	sɜn	s-aeh-n
Liver	fikat	fikat	f-e-k-ah-t
Drink	bea	bea	b-ay-ah
See	vedea	vedea	v-ay-d-ay-ah
Hear	auzy	auzy	ah-u-zz-eeh
Die	mury	mury	m-u-r-eeh
Come	veny	veny	v-ay-n-eeh
Sun	soare	soare	s-oh-ah-r-ay
Star	stea	stea	s-t-ay-ah
Water	ap3	apɜ	ah-p-aeh
Stone	pyatr3	pyatrɜ	p-eeh-ah-t-r-aeh
Fire	fok	fok	f-oh-k
Path	cale	cale	tya-ah-l-ay
Mountain	munte	munte	m-u-n-t-ay
Night	noapte	noapte	n-oh-ah-p-t-ay
Full	plin	plin	p-l-e-n
New	now	now	n-oh-w
Name	nume	nume	n-u-m-ay

Catalan

Meaning	SAMPA	IPA	Sounds
I	Zo	ʒo	gzah-oh
You	tu	tu	t-u

We	nuzaltr3s	nuzaltrɜs	n-u-zz-ah-l-t-r-aeh-s
One	un	un	u-n
Two	dos	dos	d-oh-s
Person	p3rson3	pɜrsonɜ	p-aeh-r-s-oh-n-aeh
Fish	peS	peʃ	p-ay-shh
Dog	k3	kɜ	k-aeh
Louse	poL	poʎ	p-oh-yuh
Tree	abr3	abrɜ	ah-b-r-aeh
Leaf	fuL3	fuʎɜ	f-u-yuh-aeh
Skin	peL	peʎ	p-ay-yuh
Blood	saN	saŋ	s-ah-nya
Bone	os	os	oh-s
Horn	korn	korn	k-oh-r-n
Horn	ba53	bałɜ	b-ah-ssha-aeh
Ear	urEL3	urɛʎɜ	u-r-eh-yuh-aeh
Eye	uL	uʎ	u-yuh
Nose	nas	nas	n-ah-s
Tooth	den	den	d-ay-n
Tongue	LeNgw~3	ʎeŋgw̃ɜ	yuh-ay-nya-g-wh-aeh
Knee	j3noL	jɜnoʎ	jg-aeh-n-oh-yuh
Hand	ma	ma	m-ah
Breast	pit	pit	p-e-t
Liver	fej3	fejɜ	f-ay-jg-aeh
Drink	bEur3	bɛurɜ	b-eh-u-r-aeh
See	bEur3	bɛurɜ	b-eh-u-r-aeh
Hear	s3nti	sɜnti	s-aeh-n-t-e
Die	muri	muri	m-u-r-e
Come	b3ni	bɜni	b-aeh-n-e
Sun	sol	sol	s-oh-l
Star	3streL3	ɜstreʎɜ	aeh-s-t-r-ay-yuh-aeh
Water	aixw~3	aixw̃ɜ	ah-e-xha-wh-aeh
Stone	pe8r3	peθrɜ	p-ay-ooh-r-aeh
Fire	fok	fok	f-oh-k
Path	k3mi	kɜmi	k-aeh-m-e
Mountain	mon	mon	m-oh-n
Night	nit	nit	n-e-t
Full	plE	plɛ	p-l-eh
New	nou	nou	n-oh-u
Name	nom	nom	n-oh-m

Portuguese

Meaning	SAMPA	IPA	Sounds
I	eu	eu	ay-u
You	tu	tu	t-u
We	noS	noʃ	n-oh-shh
One	u*	ũ	uh
Two	doiS	doiʃ	d-oh-e-shh

Person	pErzon	pɛrzon	p-eh-r-zz-oh-n
Fish	paiS3	paiʃɜ	p-ah-e-shh-aeh
Dog	ka*u*	kɐ̃ũ	k-ahn-uh
Louse	pioLu	pioʎu	p-e-oh-yuh-u
Tree	Ervur3	ɛrvurɜ	eh-r-v-u-r-aeh
Leaf	foLa	foʎa	f-oh-yuh-ah
Skin	pEl3	pɛlɜ	p-eh-l-aeh
Blood	sa*x3	sɐ̃xɜ	s-ahn-xha-aeh
Bone	osu	osu	oh-s-u
Horn	Sifr3	ʃifrɜ	shh-e-f-r-aeh
Ear	oraLa	oraʎa	oh-r-ah-yuh-ah
Eye	oLu	oʎu	oh-yuh-u
Nose	nariS	nariʃ	n-ah-r-e-shh
Tooth	de*t3	dɐ̃tɜ	d-en-t-aeh
Tongue	li*gua	lĩgua	l-een-g-u-ah
Knee	ZuaLu	ʒuaʎu	gzah-u-ah-yuh-u
Hand	ma*u	mɐ̃u	m-ahn-u
Breast	saiuS	saiuʃ	s-ah-e-u-shh
Liver	fixa8u	fixaɵu	f-e-xha-ah-ooh-u
Drink	b3b	bɜb	b-aeh-b
See	ver	ver	v-ay-r
Hear	ov	ov	oh-v
Die	mur	mur	m-u-r
Come	vir	vir	v-e-r
Sun	sol	sol	s-oh-l
Star	3Strela	ɜʃtrela	aeh-shh-t-r-ay-l-ah
Water	Egw~a	ɛgwɐ̃a	eh-g-wh-ah
Stone	pEdra	pɛdra	p-eh-d-r-ah
Fire	fogu	fogu	f-oh-g-u
Path	se*da	sɐ̃da	s-en-d-ah
Mountain	mo*ta5a	mɔ̃taɬa	m-oon-t-ah-ssha-ah
Night	noyt3	noytɜ	n-oh-eeh-t-aeh
Full	Seyu	ʃeyu	shh-ay-eeh-u
New	novu	novu	n-oh-v-u
Name	nom3	nomɜ	n-oh-m-aeh

Spanish

Meaning	SAMPA	IPA	Sounds
I	yo	yo	eeh-oh
You	tu	tu	t-u
We	nosotros	nosotros	n-oh-s-oh-t-r-oh-s
This	este	este	ay-s-t-ay
That	ese	ese	ay-s-ay
That	akely~a	akelỹa	ah-k-ay-l-ey-ah
Who	kien	kien	k-e-ay-n
What	ke	ke	k-ay
Not	no	no	n-oh

All	todos	todos	t-oh-d-oh-s
Many	muCos	muços	m-u-sh-oh-s
One	uno	uno	u-n-oh
Two	dos	dos	d-oh-s
Big	grande	grande	g-r-ah-n-d-ay
Long	largo	largo	l-ah-r-g-oh
Small	peke5o	pekeło	p-ay-k-ay-ssha-oh
Small	Ciko	çiko	sh-e-k-oh
Woman	muher	muher	m-u-h-ay-r
Man	ombre	ombre	oh-m-b-r-ay
Person	persona	persona	p-ay-r-s-oh-n-ah
Fish	peskado	peskado	p-ay-s-k-ah-d-oh
Fish	pes	pes	p-ay-s
Bird	ave	ave	ah-v-ay
Bird	paharo	paharo	p-ah-h-ah-r-oh
Dog	pero	pero	p-ay-r-oh
Louse	pioho	pioho	p-e-oh-h-oh
Tree	arbol	arbol	ah-r-b-oh-l
Tree	palo	palo	p-ah-l-oh
Seed	semiya	semiya	s-ay-m-e-eeh-ah
Leaf	oha	oha	oh-h-ah
Root	rais	rais	r-ah-e-s
Bark	kortesa	kortesa	k-oh-r-t-ay-s-ah
Bark	kaskara	kaskara	k-ah-s-k-ah-r-ah
Skin	piel	piel	p-e-ay-l
Flesh	karne	karne	k-ah-r-n-ay
Blood	sangre	sangre	s-ah-n-g-r-ay
Bone	weso	weso	w-ay-s-oh
Grease	grasa	grasa	g-r-ah-s-ah
Egg	wevo	wevo	w-ay-v-oh
Horn	kw~erno	kw̃erno	k-wh-ay-r-n-oh
Tail	kola	kola	k-oh-l-ah
Tail	rabo	rabo	r-ah-b-oh
Feather	pluma	pluma	p-l-u-m-ah
Hair	pelo	pelo	p-ay-l-oh
Hair	cabeyo	cabeyo	tya-ah-b-ay-eeh-oh
Head	kabesa	kabesa	k-ah-b-ay-s-ah
Ear	oreha	oreha	oh-r-ay-h-ah
Eye	oho	oho	oh-h-oh
Nose	naris	naris	n-ah-r-e-s
Mouth	boka	boka	b-oh-k-ah
Tooth	diente	diente	d-e-ay-n-t-ay
Tongue	lengw~a	lengw̃a	l-ay-n-g-wh-ah
Claw	gara	gara	g-ah-r-ah
Foot	pie	pie	p-e-ay
Foot	pata	pata	p-ah-t-ah
Knee	rodiya	rodiya	r-oh-d-e-eeh-ah
Hand	mano	mano	m-ah-n-oh
Belly	bariga	bariga	b-ah-r-e-g-ah
Neck	kw~eyo	kw̃eyo	k-wh-ay-eeh-oh

Neck	peskw~eso	peskwẽso	p-ay-s-k-wh-ay-s-oh
Breast	peCo	peço	p-ay-sh-oh
Breast	seno	seno	s-ay-n-oh
Heart	korason	korason	k-oh-r-ah-s-oh-n
Liver	igado	igado	e-g-ah-d-oh
Drink	bebe	bebe	b-ay-b-ay
Eat	kome	kome	k-oh-m-ay
Bite	morde	morde	m-oh-r-d-ay
See	ve	ve	v-ay
Hear	oir	oir	oh-e-r
Know	sabe	sabe	s-ah-b-ay
Know	konose	konose	k-oh-n-oh-s-ay
Sleep	dormi	dormi	d-oh-r-m-e
Die	mori	mori	m-oh-r-e
Kill	mata	mata	m-ah-t-ah
Swim	nada	nada	n-ah-d-ah
Fly	vola	vola	v-oh-l-ah
Walk	anda	anda	ah-n-d-ah
Walk	kamina	kamina	k-ah-m-e-n-ah
Come	veni	veni	v-ay-n-e
Lie	akosta	akosta	ah-k-oh-s-t-ah
Lie	eCa	eça	ay-sh-ah
Sit	senta	senta	s-ay-n-t-ah
Stand	esta de pie	esta de pie	ay-s-t-ah- -d-ay- -p-e-ay
Give	da	da	d-ah
Say	desi	desi	d-ay-s-e
Sun	sol	sol	s-oh-l
Moon	luna	luna	l-u-n-ah
Star	estreya	estreya	ay-s-t-r-ay-eeh-ah
Water	agw~a	agwã	ah-g-wh-ah
Rain	yuvia	yuvia	eeh-u-v-e-ah
Stone	piedra	piedra	p-e-ay-d-r-ah
Sand	arena	arena	ah-r-ay-n-ah
Earth	tiera	tiera	t-e-ay-r-ah
Cloud	nube	nube	n-u-b-ay
Smoke	humo	humo	h-u-m-oh
Fire	fuego	fuego	f-u-ay-g-oh
Ash	senisa	senisa	s-ay-n-e-s-ah
Burn	kema	kema	k-ay-m-ah
Burn	arde	arde	ah-r-d-ay
Path	senda	senda	s-ay-n-d-ah
Mountain	sero	sero	s-ay-r-oh
Mountain	monta5a	montała	m-oh-n-t-ah-ssha-ah
Red	roho	roho	r-oh-h-oh
Red	kolorado	kolorado	k-oh-l-oh-r-ah-d-oh
Green	verde	verde	v-ay-r-d-ay
Yellow	amariyo	amariyo	ah-m-ah-r-e-eeh-oh
White	blanko	blanko	b-l-ah-n-k-oh
Black	negro	negro	n-ay-g-r-oh
Night	noCe	noçe	n-oh-sh-ay

Hot	kaliente	kaliente	k-ah-l-e-ay-n-t-ay
Cold	frio	frio	f-r-e-oh
Full	yeno	yeno	eeh-ay-n-oh
New	nuevo	nuevo	n-u-ay-v-oh
Good	bw~eno	bw̃eno	b-wh-ay-n-oh
Round	redondo	redondo	r-ay-d-oh-n-d-oh
Dry	seko	seko	s-ay-k-oh
Name	nombre	nombre	n-oh-m-b-r-ay

French

Meaning	SAMPA	IPA	Sounds
I	j3	jɜ	jg-aeh
You	ti	ti	t-e
We	nu	nu	n-u
This	s3si	sɜsi	s-aeh-s-e
That	s3la	sɜla	s-aeh-l-ah
Who	ki	ki	k-e
What	kwa	kwa	k-w-ah
Not	n3 pa	nɜ pa	n-aeh- -p-ah
All	tu	tu	t-u
Many	boku	boku	b-oh-k-u
One	oe*	œ̃	uuh
Two	de	de	d-ay
Big	gra*	grɐ̃	g-r-ahn
Long	lo*	lɔ̃	l-oon
Small	p3ti	pɜti	p-aeh-t-e
Woman	fam	fam	f-ah-m
Man	om	om	oh-m
Person	om	om	oh-m
Fish	pw~aso*	pw̃asɔ̃	p-wh-ah-s-oon
Bird	wazo	wazo	w-ah-zz-oh
Dog	Sia*	ʃiɐ̃	shh-e-ahn
Louse	pu	pu	p-u
Tree	arbr3	arbrɜ	ah-r-b-r-aeh
Seed	gran	gran	g-r-ah-n
Leaf	f3y	fɜy	f-aeh-eeh
Root	rasin	rasin	r-ah-s-e-n
Bark	ekors	ekors	ay-k-oh-r-s
Skin	po	po	p-oh
Flesh	vy~a*d	vỹɐ̃d	v-ey-ahn-d
Blood	sa*	sɐ̃	s-ahn
Bone	os	os	oh-s
Grease	grais	grais	g-r-ah-e-s
Egg	3f	ɜf	aeh-f
Horn	korn	korn	k-oh-r-n
Tail	ke	ke	k-ay
Feather	ply~m	plỹm	p-l-ey-m

Hair	S3ve	ʃɜve	shh-aeh-v-ay
Head	t3t	tɜt	t-aeh-t
Ear	ore	ore	oh-r-ay
Eye	3y	ɜy	aeh-eeh
Nose	ne	ne	n-ay
Mouth	buS	buʃ	b-u-shh
Tooth	da*	dɐ̃	d-ahn
Tongue	la*g	lɐ̃g	l-ahn-g
Claw	o*gl	ɔ̃gl	oon-g-l
Foot	py~e	pỹe	p-ey-ay
Knee	j3nu	jɜnu	jg-aeh-n-u
Hand	ma*	mɐ̃	m-ahn
Belly	va*tr	vɐ̃tr	v-ahn-t-r
Neck	ku	ku	k-u
Breast	pw~atrin	pwãatrin	p-wh-ah-t-r-e-n
Heart	k3r	kɜr	k-aeh-r
Liver	fw~a	fwã	f-wh-ah
Drink	bw~a	bwã	b-wh-ah
Eat	ma*g	mɐ̃g	m-ahn-g
Bite	mord	mord	m-oh-r-d
See	vw~a	vwã	v-wh-ah
Hear	o*ta*dr	ɔ̃tɐ̃dr	oon-t-ahn-d-r
Know	savw~a	savwã	s-ah-v-wh-ah
Sleep	dormi	dormi	d-oh-r-m-e
Die	muri	muri	m-u-r-e
Kill	tue	tue	t-u-ay
Swim	naje	naje	n-ah-jg-ay
Fly	vw~ale	vwãale	v-wh-ah-l-ay
Walk	marSe	marʃe	m-ah-r-shh-ay
Come	v3ni	vɜni	v-aeh-n-e
Lie	seta*dr	setɐ̃dr	s-ay-t-ahn-d-r
Lie	etra*da*dE	etrɐ̃dɐ̃dɛ	ay-t-r-ahn-d-ahn-d-eh
Sit	sasw~a	saswã	s-ah-s-wh-ah
Sit	etrasi	etrasi	ay-t-r-ah-s-e
Stand	s3l3ve	sɜlɜve	s-aeh-l-aeh-v-ay
Stand	s3t3nird3vu	sɜtɜnirdɜvu	s-aeh-t-aeh-n-e-r-d-aeh-v-u
Give	done	done	d-oh-n-ay
Say	di	di	d-e
Sun	sole	sole	s-oh-l-ay
Moon	len	len	l-ay-n
Star	etw~ol	etwɔ̃l	ay-t-wh-oh-l
Water	o	o	oh
Rain	plui	plui	p-l-u-e
Stone	py~er	pỹer	p-ey-ay-r
Sand	sabl	sabl	s-ah-b-l
Earth	ter	ter	t-ay-r
Cloud	nuaj	nuaj	n-u-ah-jg
Smoke	fEme	fɛme	f-eh-m-ay
Fire	fe	fe	f-ay
Ash	sa*dr	sɐ̃dr	s-ahn-d-r

Burn	brule	brule	b-r-u-l-ay
Path	rut	rut	r-u-t
Mountain	mo*taj	mɔ̃taj	m-oon-t-ah-jg
Red	ruj	ruj	r-u-jg
Green	ver	ver	v-ay-r
Yellow	jon	jon	jg-oh-n
White	bla*	blɛ̃	b-l-ahn
Black	nw~ar	nw̃ar	n-wh-ah-r
Night	nui	nui	n-u-e
Hot	So	ʃo	shh-oh
Cold	fr~wa	fr̃wa	f-rn-w-ah
Full	pl3*	plɜ̃	p-l-aehn
New	nuvo	nuvo	n-u-v-oh
Good	bo*	bɔ̃	b-oon
Round	ro*	rɔ̃	r-oon
Dry	s3k	sɜk	s-aeh-k
Name	no*	nɔ̃	n-oon

Walloon

Meaning	SAMPA	IPA	Sounds
I	Ce	çe	sh-ay
You	te	te	t-ay
We	nos	nos	n-oh-s
One	E*	ɛ̃	ehnn
Person	o*m	ɔ̃m	oon-m
Dog	Ce*	çɛ̃	sh-en
Skin	pow	pow	p-oh-w
Ear	oreye	oreye	oh-r-ay-eeh-ay
Eye	ui	ui	u-e
Drink	bwEr	bwɛr	b-w-eh-r
Hear	Sute	ʃute	shh-u-t-ay
Die	murrir	murrir	m-u-r-r-e-r
Come	vnir	vnir	v-n-e-r
Star	twEl	twɛl	t-w-eh-l
Water	Ew3	ɛwɜ	eh-w-aeh
Fire	fE	fɛ	f-eh
Path	vwa*y	vwɛ̃y	v-w-ahn-eeh
Full	pli*	plĩ	p-l-een
New	novEl	novɛl	n-oh-v-eh-l

Romansh

Meaning	SAMPA	IPA	Sounds
I	yaw	yaw	eeh-ah-w
You	ti	ti	t-e
We	nus	nus	n-u-s

One	en	en	ay-n
Two	dus	dus	d-u-s
Person	k3rSTawn	kɜrʃθawn	k-aeh-r-shh-th-ah-w-n
Fish	peS	peʃ	p-ay-shh
Dog	Tawn	θawn	th-ah-w-n
Louse	pluL	pluʎ	p-l-u-yuh
Tree	plant3	plantɜ	p-l-ah-n-t-aeh
Leaf	feL	feʎ	f-ay-yuh
Skin	pel	pel	p-ay-l
Blood	saNk	saŋk	s-ah-nya-k
Bone	os	os	oh-s
Horn	korn3	kornɜ	k-oh-r-n-aeh
Ear	ureL3	ureʎɜ	u-r-ay-yuh-aeh
Eye	eL	eʎ	ay-yuh
Nose	nas	nas	n-ah-s
Tooth	dEnt	dɛnt	d-eh-n-t
Tongue	lyewNg3	lyewŋgɜ	l-eeh-ay-w-nya-g-aeh
Knee	Z3neye	ʒɜneye	gzah-aeh-n-ay-eeh-ay
Hand	mawn	mawn	m-ah-w-n
Breast	pET	pɛθ	p-eh-th
Liver	5irom	ɬirom	ssha-e-r-oh-m
Drink	bayv3r	bayvɜr	b-ah-eeh-v-aeh-r
See	v3zayr	vɜzayr	v-aeh-zz-ah-eeh-r
Hear	udir	udir	u-d-e-r
Die	murir	murir	m-u-r-e-r
Come	v35ir	vɜɬir	v-aeh-ssha-e-r
Sun	suleL	suleʎ	s-u-l-ay-yuh
Star	Stayl3	ʃtaylɜ	shh-t-ah-eeh-l-aeh
Water	aw3	awɜ	ah-w-aeh
Stone	krap	krap	k-r-ah-p
Fire	fyew	fyew	f-eeh-ay-w
Path	vi3	viɜ	v-e-aeh
Mountain	munto53	muntoɬɜ	m-u-n-t-oh-ssha-aeh
Night	noT	noθ	n-oh-th
Full	playn	playn	p-l-ah-eeh-n
New	nof	nof	n-oh-f
Name	num	num	n-u-m

Friulian

Meaning	SAMPA	IPA	Sounds
I	yo	yo	eeh-oh
You	tu	tu	t-u
We	nou	nou	n-oh-u
One	uN	uŋ	u-nya
Two	doi	doi	d-oh-e
Person	pErsoN	pɛrsoŋ	p-eh-r-s-oh-nya
Fish	pes	pes	p-ay-s

Dog	ky~aN	kỹaŋ	k-ey-ah-nya
Louse	pEdoli	pɛdoli	p-eh-d-oh-l-e
Tree	arbul	arbul	ah-r-b-u-l
Leaf	fw~eE	fw̃eɛ	f-wh-ay-eh
Skin	py~el	pỹel	p-ey-ay-l
Blood	saNk	saŋk	s-ah-nya-k
Bone	vw~es	vw̃es	v-wh-ay-s
Horn	kw~ar	kw̃ar	k-wh-ah-r
Ear	oreli	oreli	oh-r-ay-l-e
Eye	voli	voli	v-oh-l-e
Nose	nas	nas	n-ah-s
Tooth	dint	dint	d-e-n-t
Tongue	leNgE	leŋgɛ	l-ay-nya-g-eh
Knee	zEnoli	zɛnoli	zz-eh-n-oh-l-e
Knee	jEnoli	jɛnoli	jg-eh-n-oh-l-e
Hand	man	man	m-ah-n
Breast	pet	pet	p-ay-t
Liver	fiat	fiat	f-e-ah-t
Liver	fy~at	fỹat	f-ey-ah-t
Drink	bevi	bevi	b-ay-v-e
See	viodi	viodi	v-e-oh-d-e
See	vy~odi	vỹodi	v-ey-oh-d-e
Hear	sintei	sintei	s-e-n-t-ay-e
Die	murei	murei	m-u-r-ay-e
Come	vi5ei	viɬei	v-e-ssha-ay-e
Sun	soreli	soreli	s-oh-r-ay-l-e
Star	stelE	stelɛ	s-t-ay-l-eh
Water	agE	agɛ	ah-g-eh
Stone	py~erE	pỹerɛ	p-ey-ay-r-eh
Fire	fuk	fuk	f-u-k
Path	stradE	stradɛ	s-t-r-ah-d-eh
Mountain	mont	mont	m-oh-n-t
Mountain	monta5E	montaɬɛ	m-oh-n-t-ah-ssha-eh
Night	5ot	ɬot	ssha-oh-t
Full	plen	plen	p-l-ay-n
New	5uf	ɬuf	ssha-u-f
Name	non	non	n-oh-n

Italian

Meaning	SAMPA	IPA	Sounds
I	io	io	e-oh
You	tu	tu	t-u
We	noi	noi	n-oh-e
One	uno	uno	u-n-oh
Two	due	due	d-u-ay
Person	persona	persona	p-ay-r-s-oh-n-ah
Fish	peSe	peʃe	p-ay-shh-ay

Dog	kane	kane	k-ah-n-ay
Louse	pidokky~o	pidokkỹo	p-e-d-oh-k-k-ey-oh
Tree	albero	albero	ah-l-b-ay-r-oh
Leaf	foLa	foʎa	f-oh-yuh-ah
Skin	pElle	pɛlle	p-eh-l-l-ay
Blood	saNgwe	saŋgwe	s-ah-nya-g-w-ay
Bone	osso	osso	oh-s-s-oh
Horn	korno	korno	k-oh-r-n-oh
Ear	orekkyo	orekkyo	oh-r-ay-k-k-eeh-oh
Eye	okkyo	okkyo	oh-k-k-eeh-oh
Nose	naso	naso	n-ah-s-oh
Tooth	dante	dante	d-ah-n-t-ay
Tongue	liNgwa	liŋgwa	l-e-nya-g-w-ah
Knee	jinokkyo	jinokkyo	jg-e-n-oh-k-k-eeh-oh
Hand	mano	mano	m-ah-n-oh
Breast	pEtto	pɛtto	p-eh-t-t-oh
Liver	fegato	fegato	f-ay-g-ah-t-oh
Drink	bere	bere	b-ay-r-ay
See	ved	ved	v-ay-d
Hear	ud	ud	u-d
Die	mor	mor	m-oh-r
Come	vEn	vɛn	v-eh-n
Sun	sole	sole	s-oh-l-ay
Star	stella	stella	s-t-ay-l-l-ah
Water	akwa	akwa	ah-k-w-ah
Stone	pyEtra	pyɛtra	p-eeh-eh-t-r-ah
Fire	fwoko	fwoko	f-w-oh-k-oh
Path	sentyaro	sentyaro	s-ay-n-t-eeh-ah-r-oh
Mountain	monta5a	montała	m-oh-n-t-ah-ssha-ah
Night	notte	notte	n-oh-t-t-ay
Full	pyEno	pyɛno	p-eeh-eh-n-oh
New	nwovo	nwovo	n-w-oh-v-oh
Name	nome	nome	n-oh-m-ay

```
#NEXUS
begin taxa;
  dimensions ntax=15;
  taxlabels Latin Nuorese Cagliari Romanian Arumanian Walloon French Provencal
Portuguese Spanish Catalan Ladin Friulian Romansh Italian;
end;

begin trees;
  tree LanguageTree = (Latin:0.81,((Nuorese:411.2,Cagliari:410.52):755.27,((
Romanian:526.1,Arumanian:526.97):579.53,((((Walloon:200.55,French:192.63):
243.09,Provencal:437.74):333.32,((Portuguese:369.61,Spanish:369.4):256.8,
Catalan:626.07):146.55):133.85,((Ladin:582.91,(Friulian:403.47,Romansh:403.37):
180.58):178.13,Italian:760.59):144.66):199.54):60.6):958.67):1324.32;
end;
```

English.nytril

```
with Lang
  let Separator          = ", "
  let Abstract           = "Abstract"
  let Affricate          = "Affricate"
  let Affricates         = "Affricates"
  let Alveolar           = "Alveolar"
  let AlveoloPalatal     = "Alveolo-Palatal"
  let And                = "and"
  let Appendices         = "Appendices"
  let Appendix           = "Appendix"
  let Approximant        = "Approximant"
  let Approximants       = "Approximants"
  let Authors            = "Authors"
  let AvailableAt        = "Available at"
  let Back               = "Back"
  let Bilabial           = "Bilabial"
  let Category           = "Category"
  let Categories         = "Categories"
  let Central            = "Central"
  let Click              = "Click"
  let Close              = "Close"
  let CloseMid           = "Close-mid"
  let Conclusion         = "Conclusion"
  let Consonant          = "Consonant"
  let Consonants         = "Consonants"
  let Continuant         = "Continuant"
  let Continuants        = "Continuants"
  let Coronal            = "Coronal"
  let Dental             = "Dental"
  let Dorsal             = "Dorsal"
  let Diacritic          = "Diacritic"
  let Diacritics         = "Diacritics"
  let Ejective           = "Ejective"
  let EMail              = "E-mail"
  let EjectiveAffricates = "Ejective Affricates"
  let Feature            = "Feature"
  let Features           = "Features"
  let Fricative          = "Fricative"
  let Fricatives         = "Fricatives"
  let Front              = "Front"
  let Glottal            = "Glottal"
  let Implosive          = "Implosive"
  let Impossible         = "Impossible"
  let ImpossibleShaded   = "Shaded areas denote articulations judged to be impossible."
  let Introduction       = "Introduction"
  let IPA                = "IPA"
  let IPAListing         = "IPA Segments"
  let IPAFullName        = "International Phonetic Alphabet"
  let Labial             = "Labial"
  let LabioDental        = "Labio-Dental"
  let LabialPalatal      = "Labial-Palatal"
  let LabialVelar        = "Labial-Velar"
  let LinguoLabial       = "Linguo-Labial"
  let Laryngeal          = "Laryngeal"
  let Language           = "Language"
  let LanguagePhylogeny  = "Language Tree"
  let LanguageList       = "List of Languages"
  let Lateral            = "Lateral"
  let Laterals           = "Laterals"
  let Liquid             = "Liquid"
  let Liquids            = "Liquids"
  let LongVowel          = "Long"
  let LongVowels         = "Long Vowels"
  let Manner             = "Manner"
  let Manners            = "Manners"
  let Meaning            = "Meaning"
  let Meanings           = "Meanings"
  let Methods            = "Methods"
  let Mid                = "Mid"
```

```
  let Name               = "Name"
  let Nasal              = "Nasal"
  let Nasals             = "Nasals"
  let NearBack           = "Near-Back"
  let NearClose          = "Near-close"
  let NearFront          = "Near-Front"
  let NearOpen           = "Near-open"
  let NoFeatures         = "No Features"
  let NonSibilant        = "Non-Sibilant"
  let NonPulmonic        = "Non-Pulmonic"
  let NPConsonants       = "Non-Pulmonic Consonants"
  let NytrilSourceCode   = "Nytril Source Code"
  let Obstruent          = "Obstruent"
  let Obstruents         = "Obstruents"
  let Occlusive          = "Occlusive"
  let Occlusives         = "Occlusives"
  let Open               = "Open"
  let OpenMid            = "Open-mid"
  let Or                 = "or"
  let OtherSegments      = "Other Segments"
  let Pharyngeal         = "Pharyngeal-Epiglottal"
  let Pulmonic           = "Pulmonic"
  let PConsonants        = "Pulmonic Consonants"
  let Palatal            = "Palatal"
  let PalatoAlveolar     = "Palato-Alveolar"
  let Place              = "Place"
  let Places             = "Places"
  let Plosive            = "Plosive"
  let Plosives           = "Plosives"
  let Property           = "Property"
  let PostAlveolar       = "Post-Alveolar"
  let PulmonicAffricates = "Pulmonic Affricates"
  let References         = "References"
  let Retroflex          = "Retroflex"
  let Rhotic             = "Rhotic"
  let Rhotics            = "Rhotics"
  let Rounded            = "Rounded"
  let Sampa              = "SAMPA"
  let Segment            = "Segment"
  let Segments           = "Segments"
  let SemiVowel          = "Semivowel"
  let SemiVowels         = "Semivowels"
  let Sibilant           = "Sibilant"
  let Sibilants          = "Sibilants"
  let Sounds             = "Sounds"
  let Stop               = "Stop"
  let Strident           = "Strident"
  let Stridents          = "Stridents"
  let SymbolPairVoiced   = "Where symbols appear in pairs, the one to the right represents a modally voiced
consonant."
  let SymbolPairRounded  = "Where symbols appear in pairs, the one to the right represents a rounded vowel."
  let TapFlap            = "Tap/Flap"
  let TapFlaps           = "Tap/Flaps"
  let Tenuis             = "Tenuis"
  let Text               = "Text"
  let Trill              = "Trill"
  let Trills             = "Trills"
  let Unrounded          = "Unrounded"
  let Uvular             = "Uvular"
  let Velar              = "Velar"
  let Velarized          = "Velarized"
  let Vocoid             = "Vocoid"
  let Vibrant            = "Vibrant"
  let Vibrants           = "Vibrants"
  let Vocoids            = "Vocoids"
  let Voiced             = "Voiced"
  let Voiceless          = "Voiceless"
  let Vowel              = "Vowel"
  let Vowels             = "Vowels"
  let Word               = "Word"
  let Years              = "Years"
end
```

Library.nytril

```
using Format, Units, Math, IO
//========================================================================

with TreeLib
  let GetNodeLabel(node) = node.Data?.SymbolName

  let VisitNodeTaxa(set, node) begin
    set.AddReference(node.Data);
    VisitNodeTaxa(set, each node);
  end

  let GetTaxaLabels(tree) begin
    var set = Type.Dictionary(256);
    VisitNodeTaxa(set, tree);
    return (each set.ValueList).SymbolName;
  end
end
//========================================================================

with Nexus
  let CharacterList = ('A'..'Z' step 1) + ('a'..'z' step 1) + ('0'..'9' step 1)
  let EndMarker     = ";"
  let Missing       = "?"
  let Quote         = "\""

  let Keyword(name) = Span {
    TextColor: Colors.Blue,
    name
  }

  let Comment(text) = Span {
    TextColor: Colors.Green,
    "#"
    text
  }

  let AddLine(name) = Span {
    Keyword(name),
    End: EndMarker,
  }

  let AddValue(name, value) = Span {
    Space,
    Keyword(name),
    "=",
    value,
  }

  let Scope(name) = TextBlock {
    IndentSpace: 2,
    Begin: Span {
      Keyword("begin"),
      Space,
      name,
      EndMarker,
    },
    End: Span {
      Keyword("end"),
      EndMarker,
    },
  }

  let NexusFile = TextBlock {
    Comment("NEXUS"),
  }

  let Newick(node) = Span {
    if (node.Length > 0)
      "(",
      Span {
        Separator: ",",
        Newick(each node)
```

```
      },
      ")",
    end,
    TreeLib.GetNodeLabel(node),
    ":",
    node.Branch
  }

  let ShowTreeLine(ref tree) = Span {
    "tree ",
    tree.SymbolName,
    " = ",
    Newick(tree),
    EndMarker
  }

  let TreeFormat(taxa) = NexusFile {
    Scope("taxa") {
      AddLine("dimensions") {
        AddValue("ntax", taxa.Length),
      },
      AddLine("taxlabels") {
        Space,
        Span {
          Separator: Space,
          taxa,
        },
      },
    }
  }

  let TreeFile(ref tree) = TreeFormat(TreeLib.GetTaxaLabels(tree)) {
    Scope("trees") {
      ShowTreeLine(ref tree),
    }
  }

  let ShowTaxon(maxlength, taxon, range) = Span {
    taxon.Name,
    Space * (maxlength - taxon.Name.Length),
    IPA.ShowCharacter(each taxon.Characters[range])
  }

  let ShowTaxonSet(maxlength, taxa, range) = {
    ShowTaxon(maxlength, each taxa, range),
    Empty,
  }

  let CharacterFile(taxa) begin
    var maxlength = Math.Max((each taxa).Name.Length)+1;
    var total     = taxa[0].Characters.Length;
    return NexusFile {
      Scope("DATA") {
        AddLine("dimensions") {
          AddValue("ntax", taxa.Length),
          AddValue("nchar", total),
        },
        AddLine("format") {
          AddValue("datatype", "STANDARD"),
          AddValue("gap", IPA.GapSegment.Ipa),
          AddValue("missing", Missing),
          AddValue("symbols", Span {Quote, CharacterList[Results.UniqueSegments.IndexRange], Quote}),
        },
        Empty,
        Keyword("matrix"),
        ShowTaxonSet(maxlength, taxa, each ((0..<total) / (70 - maxlength))),
        EndMarker,
      }
    }
  end
end
//============================================================================
```

Languages.nytril

```
let AddLanguage(name, cases=0, words=null) = {
  Name: name,
  Cases: cases,
  Words: words,
}

with Languages
  let Old_Irish          = AddLanguage("Old Irish", 5)
  let Irish              = AddLanguage("Irish", 4)
  let Scots_Gaelic       = AddLanguage("Scots Gaelic", 4)
  let Welsh              = AddLanguage("Welsh")
  let Breton             = AddLanguage("Breton")
  let Cornish            = AddLanguage("Cornish")
  let Latin              = AddLanguage("Latin", 6, WordList.Latin)
  let Nuorese            = AddLanguage("Nuorese")
  let Cagliari           = AddLanguage("Cagliari")
  let Romanian           = AddLanguage("Romanian", 3, WordList.Romanian)
  let Arumanian          = AddLanguage("Arumanian", 3)
  let Catalan            = AddLanguage("Catalan", 0, WordList.Catalan)
  let Portuguese         = AddLanguage("Portuguese", 0, WordList.Portuguese)
  let Spanish            = AddLanguage("Spanish", 0, WordList.Spanish)
  let French             = AddLanguage("French", 0, WordList.French)
  let Provencal          = AddLanguage("Provencal")
  let Walloon            = AddLanguage("Walloon", 0, WordList.Walloon)
  let Ladin              = AddLanguage("Ladin")
  let Romansh            = AddLanguage("Romansh", 0, WordList.Romansh)
  let Friulian           = AddLanguage("Friulian", 0, WordList.Friulian)
  let Italian            = AddLanguage("Italian", 0, WordList.Italian)
  let Gothic             = AddLanguage("Gothic", 5)
  let Old_West_Norse     = AddLanguage("Old West Norse", 4)
  let Icelandic          = AddLanguage("Icelandic", 4)
  let Faroese            = AddLanguage("Faroese", 4)
  let Norwegian          = AddLanguage("Norwegian", 2)
  let Swedish            = AddLanguage("Swedish", 2)
  let Danish             = AddLanguage("Danish", 2)
  let Old_English        = AddLanguage("Old English", 4)
  let English            = AddLanguage("English")
  let Frisian            = AddLanguage("Frisian", 2)
  let Old_High_German    = AddLanguage("Old High German", 5)
  let German             = AddLanguage("German", 4)
  let Luxembourgish      = AddLanguage("Luxembourgish", 3)
  let Swiss_German       = AddLanguage("Swiss German", 3)
  let Dutch              = AddLanguage("Dutch", 2)
  let Flemish            = AddLanguage("Flemish", 2)
  let Afrikaans          = AddLanguage("Afrikaans")
  let Tosk               = AddLanguage("Tosk", 4)
  let Arvanitika         = AddLanguage("Arvanitika", 4)
  let Ancient_Greek      = AddLanguage("Ancient Greek", 5)
  let Modern_Greek       = AddLanguage("Modern Greek", 4)
  let Classical_Armenian = AddLanguage("Classical Armenian", 7)
  let Eastern_Armenian   = AddLanguage("Eastern Armenian", 7)
  let Adapazar           = AddLanguage("Adapazar")
  let Old_Prussian       = AddLanguage("Old Prussian", 5)
  let Lithuanian         = AddLanguage("Lithuanian", 7)
  let Latvian            = AddLanguage("Latvian", 7)
  let Czech              = AddLanguage("Czech", 7)
  let Slovak             = AddLanguage("Slovak", 6)
  let Polish             = AddLanguage("Polish", 7)
  let Lower_Sorbian      = AddLanguage("Lower Sorbian", 6)
  let Upper_Sorbian      = AddLanguage("Upper Sorbian", 6)
  let Ukrainian          = AddLanguage("Ukrainian", 7)
  let Belarusian         = AddLanguage("Belarusian", 6)
  let Russian            = AddLanguage("Russian", 6)
  let Slovenian          = AddLanguage("Slovenian", 6)
  let Macedonian         = AddLanguage("Macedonian")
  let Bulgarian          = AddLanguage("Bulgarian", 2)
  let Serbian            = AddLanguage("Serbian", 7)
  let Old_Church_Slavic  = AddLanguage("Old Church Slavic", 7)
  let Avestan            = AddLanguage("Avestan", 8)
  let Pashto             = AddLanguage("Pashto", 4)
  let Waziri             = AddLanguage("Waziri")
  let Tajik              = AddLanguage("Tajik")
  let Persian            = AddLanguage("Persian")
  let Sogdian            = AddLanguage("Sogdian", 6)
  let Wakhi              = AddLanguage("Wakhi", 4 /* ? */)
```

```
  let Baluchi            = AddLanguage("Baluchi", 3)
  let Kurdish            = AddLanguage("Kurdish", 4)
  let Zazaki             = AddLanguage("Zazaki", 2)
  let Shughni            = AddLanguage("Shughni", 5)
  let Sariqoli           = AddLanguage("Sariqoli", 2)
  let Digor_Ossetic      = AddLanguage("Digor Ossetic", 9)
  let Vedic_Sanskrit     = AddLanguage("Vedic Sanskrit", 8)
  let Nepali             = AddLanguage("Nepali")
  let Assamese           = AddLanguage("Assamese", 6)
  let Oriya              = AddLanguage("Oriya", 3)
  let Bengali            = AddLanguage("Bengali", 4)
  let Bihari             = AddLanguage("Bihari", 5)
  let Marwari            = AddLanguage("Marwari")
  let Hindi              = AddLanguage("Hindi", 3)
  let Urdu               = AddLanguage("Urdu", 3)
  let Sindhi             = AddLanguage("Sindhi", 5)
  let Lahnda             = AddLanguage("Lahnda")
  let Panjabi            = AddLanguage("Panjabi", 5)
  let Gujarati           = AddLanguage("Gujarati", 3)
  let Marathi            = AddLanguage("Marathi", 8)
  let Kashmiri           = AddLanguage("Kashmiri", 5)
  let Singhalese         = AddLanguage("Singhalese", 8)
  let Romani             = AddLanguage("Romani", 3)
  let Tocharian_A        = AddLanguage("Tocharian A", 3)
  let Tocharian_B        = AddLanguage("Tocharian B", 3)
  let Hittite            = AddLanguage("Hittite", 8)
end
//=============================================================================
```

LanguageTree.nytril

```
using Format, Languages
//=============================================================================

with LanguageBranches
  let Branch(branch) = Node {
    Branch: branch
  }

  let Leaf(ref language, branch) = Node {
    Data: ref language,
    Branch: branch,
    Label: language.Name
  }

  let Romance = Branch(1324.32) {
    Leaf(Latin, 0.81),
    Branch(958.67) {
      Branch(755.27) {
        Leaf(Nuorese, 411.20),
        Leaf(Cagliari, 410.52),
      },
      Branch(60.60) {
        Branch(579.53) {
          Leaf(Romanian, 526.10),
          Leaf(Arumanian, 526.97),
        },
        Branch(199.54) {
          Branch(133.85) {
            Branch(333.32) {
              Branch(243.09) {
                Leaf(Walloon, 200.55),
                Leaf(French, 192.63),
              },
              Leaf(Provencal, 437.74),
            },
            Branch(146.55) {
              Branch(256.80) {
                Leaf(Portuguese, 369.61),
                Leaf(Spanish, 369.40),
              },
              Leaf(Catalan, 626.07),
```

```
          }
        },
        Branch(144.66) {
          Branch(178.13) {
            Leaf(Ladin, 582.91),
            Branch(180.58) {
              Leaf(Friulian, 403.47),
              Leaf(Romansh, 403.37),
            }
          },
          Leaf(Italian, 760.59),
        }
      }
    }
  }
}

let Germanic = Branch(443.50) {
  Branch(117.63) {
    Branch(164.08) {
      Branch(870.92) {
        Branch(240.05) {
          Branch(56.52) {
            Leaf(Afrikaans, 220.24),
            Leaf(Flemish, 219.03),
          },
          Leaf(Dutch, 276.86),
        },
        Leaf(Frisian, 517.80),
      },
      Branch(269.41) {
        Leaf(Old_High_German, 0.83),
        Branch(802.11) {
          Branch(89.71) {
            Leaf(Luxembourgish, 228.81),
            Leaf(Swiss_German, 226.19),
          },
          Leaf(German, 314.20),
        }
      }
    },
    Branch(573.40) {
      Leaf(Old_English, 0.81),
      Leaf(English, 1004.03),
    }
  },
  Branch(538.29) {
    Branch(358.92) {
      Leaf(Old_West_Norse, 0.84),
      Branch(383.13) {
        Branch(92.51) {
          Leaf(Faroese, 295.58),
          Leaf(Icelandic, 295.91),
        },
        Leaf(Norwegian, 391.20),
      }
    },
    Branch(633.39) {
      Leaf(Swedish, 497.87),
      Leaf(Danish, 497.35),
    }
  }
}

let ChangA3 = Node {
  Branch(665.23) {
    Branch(503.96) {
      Branch(256.63) {
        Branch(129.51) {
          Branch(1949.74) {
            Branch(910.55) {
              Leaf(Old_Prussian, 1020.97),
              Branch(629.79) {
                Leaf(Lithuanian, 917.14),
                Leaf(Latvian, 917.35),
```

```
      }
    },
    Branch(1229.90) {
      Branch(499.40) {
        Branch(81.74) {
          Leaf(Polish, 646.17),
          Branch(171.47) {
            Branch(403.76) {
              Leaf(Upper_Sorbian, 69.87),
              Leaf(Lower_Sorbian, 71.70),
            },
            Branch(206.18) {
              Leaf(Czech, 267.53),
              Leaf(Slovak, 268.56),
            }
          }
        },
        Branch(129.04) {
          Branch(144.63) {
            Leaf(Ukrainian, 454.98),
            Leaf(Belarusian, 454.65),
          },
          Leaf(Russian, 598.99),
        }
      },
      Branch(71.60) {
        Leaf(Old_Church_Slavic, 194.52),
        Branch(335.15) {
          Branch(154.16) {
            Branch(179.68) {
              Leaf(Macedonian, 486.21),
              Leaf(Bulgarian, 486.75),
            },
            Leaf(Serbian, 665.73),
          },
          Leaf(Slovenian, 820.99),
        }
      }
    }
  },
  Branch(320.14) {
    Branch(333.35) {
      Branch(304.37) {
        Branch(1276.69) {
          Branch(990.60) {
            Leaf(Old_Irish, 0.80),
            Branch(686.97) {
              Leaf(Irish, 495.56),
              Leaf(Scots_Gaelic, 495.92),
            }
          },
          Branch(1041.40) {
            Branch(363.96) {
              Leaf(Cornish, 509.95),
              Leaf(Breton, 767.14),
            },
            Leaf(Welsh, 1130.65),
          }
        },
        Romance
      },
      Branch(1640.21) {
        Germanic,
        Leaf(Gothic, 488.88),
      }
    },
    Branch(3559.09) {
      Leaf(Tosk, 527.11),
      Leaf(Arvanitika, 528.86),
    }
  }
},
Branch(792.34) {
  Branch(2217.67) {
    Leaf(Classical_Armenian, 0.79),
```

```
      Branch(876.25) {
        Leaf(Eastern_Armenian, 650.36),
        Leaf(Adapazar, 650.99),
      }
    },
    Branch(1320.57) {
      Leaf(Ancient_Greek, 0.80),
      Leaf(Modern_Greek, 2423.58),
    }
  }
},
Branch(846.08) {
  Branch(1045.43) {
    Leaf(Avestan, 423.76),
    Branch(673.09) {
      Branch(175.08) {
        Branch(516.83) {
          Branch(203.00) {
            Branch(269.81) {
              Leaf(Wakhi, 1064.02),
              Branch(435.29) {
                Leaf(Shughni, 629.37),
                Leaf(Sariqoli, 629.67),
              }
            },
            Branch(208.53) {
              Branch(601.98) {
                Leaf(Tajik, 523.07),
                Leaf(Persian, 523.46),
              },
              Branch(197.46) {
                Leaf(Baluchi, 928.67),
                Branch(285.40) {
                  Leaf(Zazaki, 643.65),
                  Leaf(Kurdish, 641.77),
                }
              }
            }
          },
          Branch(1108.52) {
            Leaf(Pashto, 427.66),
            Leaf(Waziri, 429.56),
          }
        },
        Leaf(Sogdian, 772.55),
      },
      Leaf(Digor_Ossetic, 2229.15),
    }
  },
  Branch(695.61) {
    Leaf(Vedic_Sanskrit, 0.80),
    Branch(1218.19) {
      Branch(323.84) {
        Leaf(Romani, 1710.04),
        Leaf(Kashmiri, 1709.37),
      },
      Branch(281.72) {
        Leaf(Singhalese, 1752.27),
        Branch(340.38) {
          Leaf(Nepali, 1410.86),
          Branch(176.92) {
            Branch(167.75) {
              Leaf(Bihari, 1066.71),
              Branch(244.83) {
                Leaf(Bengali, 822.16),
                Branch(200.33) {
                  Leaf(Assamese, 622.91),
                  Leaf(Oriya, 621.64),
                }
              }
            },
            Branch(190.09) {
              Branch(125.13) {
                Branch(168.55) {
                  Leaf(Sindhi, 751.50),
```

```
                        Leaf(Marwari, 751.75),
                      },
                      Branch(284.29) {
                        Leaf(Hindi, 633.56),
                        Branch(164.61) {
                          Leaf(Urdu, 470.12),
                          Branch(154.02) {
                            Leaf(Lahnda, 317.19),
                            Leaf(Panjabi, 316.85),
                          }
                        }
                      }
                    },
                    Branch(350.45) {
                      Leaf(Marathi, 694.13),
                      Leaf(Gujarati, 694.02),
                    }
                  }
                }
              }
            }
          }
        }
      },
      Branch(3556.57) {
        Leaf(Tocharian_A, 410.41),
        Leaf(Tocharian_B, 400.03),
      }
    },
    Leaf(Hittite, 2582.56),
  }
end
//=============================================================================
```

IPA.nytril

```
using Format, Units, Type, IPA.FeatureSet
//=============================================================================

with IPA
  with Opens
    let Close     = enum {Name: Lang.Close}
    let NearClose = enum {Name: Lang.NearClose}
    let CloseMid  = enum {Name: Lang.CloseMid}
    let Mid       = enum {Name: Lang.Mid}
    let OpenMid   = enum {Name: Lang.OpenMid}
    let NearOpen  = enum {Name: Lang.NearOpen}
    let Open      = enum {Name: Lang.Open}
  end

  with Backnesses
    let Front     = enum {Name: Lang.Front}
    let NearFront = enum {Name: Lang.NearFront}
    let Central   = enum {Name: Lang.Central}
    let NearBack  = enum {Name: Lang.NearBack}
    let Back      = enum {Name: Lang.Back}
  end

  with Places
    let Bilabial       = enum {Name: Lang.Bilabial}
    let Labial         = enum {Name: Lang.Labial}
    let LabialVelar    = enum {Name: Lang.LabialVelar}
    let LabialPalatal  = enum {Name: Lang.LabialPalatal}
    let LabioDental    = enum {Name: Lang.LabioDental}
    let LinguoLabial   = enum {Name: Lang.LinguoLabial}
    let Dental         = enum {Name: Lang.Dental}
    let Alveolar       = enum {Name: Lang.Alveolar}
    let AlveoloPalatal = enum {Name: Lang.AlveoloPalatal}
    let PostAlveolar   = enum {Name: Lang.PostAlveolar}
    let Retroflex      = enum {Name: Lang.Retroflex}
    let Palatal        = enum {Name: Lang.Palatal}
    let PalatoAlveolar = enum {Name: Lang.PalatoAlveolar}
    let Velar          = enum {Name: Lang.Velar}
```

```
  let Uvular         = enum {Name: Lang.Uvular}
  let Pharyngeal     = enum {Name: Lang.Pharyngeal}
  let Glottal        = enum {Name: Lang.Glottal}
end

with FeatureSet
  let NoFeature   = flag {Name: Lang.NoFeatures}
  let Rounded     = flag {Name: Lang.Rounded}
  let Vocoid      = flag {Name: Lang.Vocoid}
  let Semivowel   = flag {Name: Lang.SemiVowel}
  let LongVowel   = flag {Name: Lang.LongVowel}
  let Voiced      = flag {Name: Lang.Voiced}
  let Velarized   = flag {Name: Lang.Velarized}
  let Ejective    = flag {Name: Lang.Ejective}
  let Pulmonic    = flag {Name: Lang.Pulmonic}
  let Nasal       = flag {Name: Lang.Nasal}
  let Tenuis      = flag {Name: Lang.Tenuis}
  let Lateral     = flag {Name: Lang.Lateral}
  let Sibilant    = flag {Name: Lang.Sibilant}
  let Fricative   = flag {Name: Lang.Fricative}
  let Approximant = flag {Name: Lang.Approximant}
  let Implosive   = flag {Name: Lang.Implosive}
  let Central     = flag {Name: Lang.Central}
  let TapFlap     = flag {Name: Lang.TapFlap}
  let Trill       = flag {Name: Lang.Trill}
  let Stop        = flag {Name: Lang.Stop}
  let Click       = flag {Name: Lang.Click}
  let Affricate   = flag {Name: Lang.Affricate}
  let Vowel       = flag {Name: Lang.Vowel}
  let Rhotic      = flag {Name: Lang.Rhotic}
  let Occlusive   = flag {Name: Lang.Occlusive}
  let Strident    = flag {Name: Lang.Strident}
  let Obstruent   = flag {Name: Lang.Obstruent}
  let Continuant  = flag {Name: Lang.Continuant}
  let Vibrant     = flag {Name: Lang.Vibrant}
  let Liquid      = flag {Name: Lang.Liquid}
end

let Encode(ipa, sampa) = {
  Popup: SegmentPopup,
  Ipa: ipa,
  Sampa: sampa,
  Impossible: false,
  Punctuation: false,
  Diacritic: false,
}

let Diac(description, ipa, sampa) = Encode(ipa, sampa) {
  Diacritic: true,
  Description: description,
}

let Con(features, place, ipa, sampa) = Encode(ipa, sampa) {
  Features: features,
  Place: place,
}

let Vow(features, open, backness, ipa, sampa) = Encode(ipa, sampa) {
  Features: features | Vowel,
  Open: open,
  Backness: backness,
}

let Dia(ipa, sampa) = Encode(ipa, sampa) {
  Diacritic: true
}

let Punct(ipa) = Encode(ipa, ipa) {
  Punctuation: true,
}

let Imp(features, place) = {
  Features: features,
  Place: place,
  Diacritic: false,
```

```
    Punctuation: false,
    Impossible: true,
  }

  let NoSegment     = Encode("?", "!?") {Impossible: true}
  let GapSegment    = Punct("-")
  let LeftSegment   = Punct("(")
  let RightSegment  = Punct(")")
  let SpaceSegment  = Punct(" ")

  with Segments
    //==================================
    // Extra segments found in word list
    //==================================


    let rn        = Con(Nasal Voiced Pulmonic Fricative, Places.Uvular, "ʀ\u0303", "ʀ~")

// French Cold "Froid"  Same as SAMPA "R"?
//    let r_nasal = Con(Nasal Voiced Pulmonic Fricative, Places.Uvular, "ʁ", "ʀ~")

    //==================================
    // Pulmonic Consonants
    //==================================

    // Nasal
    let VlBilabialNasal             = Con(Pulmonic Nasal, Places.Bilabial, "m̥", "m_0")
    let m                           = Con(Voiced Pulmonic Nasal Occlusive, Places.Bilabial, "m", "m")
    let VdLabioDentalNasal          = Con(Voiced Pulmonic Nasal Occlusive, Places.LabioDental, "ɱ", "F")
    let VdLinguoLabioNasal          = Con(Voiced Pulmonic Nasal, Places.LinguoLabial, "n̻", "m_d")
    let VlAlveolarNasal             = Con(Pulmonic Nasal, Places.Alveolar, "n̥", "n_0")
    let n                           = Con(Voiced Pulmonic Nasal Occlusive, Places.Alveolar, "n", "n")
    let VlRetroFlexNasal            = Con(Pulmonic Nasal, Places.Retroflex, "ɳ̊", "n`_0")
    let VdRetroFlexNasal            = Con(Voiced Pulmonic Nasal Occlusive, Places.Retroflex, "ɳ", "n`")
    let VlPalatalNasal              = Con(Pulmonic Nasal, Places.Palatal, "ɲ̊", "J_0")
    let VdPalatalNasal              = Con(Voiced Pulmonic Nasal Occlusive, Places.Palatal, "ɲ", "J")
    let VlVelarNasal                = Con(Pulmonic Nasal, Places.Velar, "ŋ̊", "N_0")
    let nya                         = Con(Voiced Pulmonic Nasal Occlusive, Places.Velar, "ŋ", "N")
    let VdUvularNasal               = Con(Voiced Pulmonic Nasal Occlusive, Places.Uvular, "ɴ", "N\\")

    // Stop
    let p                           = Con(Pulmonic Stop Occlusive, Places.Bilabial, "p", "p")
    let b                           = Con(Voiced Pulmonic Stop Occlusive, Places.Bilabial, "b", "b")
    let VlLabioDentalStop           = Con(Pulmonic Stop, Places.LabioDental, "p̪", "p_d")
    let VdLabioDentalStop           = Con(Voiced Pulmonic Stop, Places.LabioDental, "b̪", "b_d")
    let VlLinguoLabialStop          = Con(Pulmonic Stop, Places.LinguoLabial, "t̻", "")
    let VdLinguoLabialStop          = Con(Voiced Pulmonic Stop, Places.LinguoLabial, "d̻", "")
    let t                           = Con(Pulmonic Stop Occlusive, Places.Alveolar, "t", "t")
    let d                           = Con(Voiced Pulmonic Stop Occlusive, Places.Alveolar, "d", "d")
    let VlRetroflexStop             = Con(Pulmonic Stop Occlusive, Places.Retroflex, "ʈ", "t`")
    let VdRetroflexStop             = Con(Voiced Pulmonic Stop Occlusive, Places.Retroflex, "ɖ", "d`")
    let tya                         = Con(Pulmonic Stop Occlusive, Places.Palatal, "c", "c")
    let VdPalatalStop               = Con(Voiced Pulmonic Stop Occlusive, Places.Palatal, "ɟ", "J\\")
    let k                           = Con(Pulmonic Stop Occlusive, Places.Velar, "k", "k")
    let kw                          = Con(Pulmonic Stop Occlusive, Places.Labial, "kʷ", "k_W")

    let g                           = Con(Voiced Pulmonic Stop Occlusive, Places.Velar, "g", "g")
    let gw                          = Con(Voiced Pulmonic Stop Occlusive, Places.Labial, "gʷ", "g_W")

    let VlUvularStop                = Con(Pulmonic Stop Occlusive, Places.Uvular, "q", "q")
    let VdUvularStop                = Con(Voiced Pulmonic Stop Occlusive, Places.Uvular, "ɢ", "G\\")
    let VdEpiglottalStop            = Con(Pulmonic Stop, Places.Pharyngeal, "ʡ", ">\\")
    let GlottalStop                 = Con(Pulmonic Stop, Places.Glottal, "ʔ", "?")

    // Sibilant Fricative
    let s                           = Con(Pulmonic Sibilant Fricative Strident Obstruent Continuant, Places.
Alveolar, "s", "s")
    let zz                          = Con(Voiced Pulmonic Sibilant Fricative Strident Obstruent Continuant,
Places.Alveolar, "z", "z")
    let shh                         = Con(Pulmonic Sibilant Fricative Strident Obstruent Continuant, Places.
PostAlveolar, "ʃ", "S")
    let gzah                        = Con(Voiced Pulmonic Sibilant Fricative Strident Obstruent Continuant,
Places.PostAlveolar, "ʒ", "Z")
    let VlRetroflexSibFricative     = Con(Pulmonic Sibilant Fricative Strident Obstruent Continuant, Places.
Retroflex, "ʂ", "s`")
```

```
    let VdRetroflexSibFricative      = Con(Voiced Pulmonic Sibilant Fricative Strident Obstruent Continuant,
Places.Retroflex, "ʐ̺", "z`")
    let VlPalatalSibFricative        = Con(Pulmonic Sibilant Fricative Strident Obstruent Continuant, Places.
Palatal, "ɕ", "s\\")
    let VdPalatalSibFricative        = Con(Voiced Pulmonic Sibilant Fricative Strident Obstruent Continuant,
Places.Palatal, "ʑ", "z\\")

    // Fricative
    let VlBilabialFricative          = Con(Pulmonic Fricative Obstruent Continuant, Places.Bilabial, "ɸ",
"p\\")
    let VdBilabialFricative          = Con(Voiced Pulmonic Fricative Obstruent Continuant, Places.Bilabial,
"β", "B")
    let f                            = Con(Pulmonic Fricative Obstruent Continuant Strident, Places.
LabioDental, "f", "f")
    let v                            = Con(Voiced Pulmonic Fricative Obstruent Continuant Strident, Places.
LabioDental, "v", "v")
    let VlLinguoLabialFricative      = Con(Pulmonic Fricative, Places.LinguoLabial, "θ̼", "")
    let VdLinguoLabialFricative      = Con(Voiced Pulmonic Fricative, Places.LinguoLabial, "ð̼", "")
    let th                           = Con(Pulmonic Fricative Obstruent Continuant, Places.Dental, "θ", "T")
    let VdDentalFricative            = Con(Voiced Pulmonic Fricative Obstruent Continuant, Places.Dental, "ð",
"D")
    let VlAlveolarFricative          = Con(Pulmonic Fricative, Places.Alveolar, "θ̪", "")
    let VdAlveolarFricative          = Con(Voiced Pulmonic Fricative, Places.Alveolar, "ð̪", "")
    let VlPostaveolarFricative       = Con(Pulmonic Fricative, Places.PostAlveolar, "ɹ̠̊˔")
    let VdPostalveolarFricative      = Con(Voiced Pulmonic Fricative, Places.PostAlveolar, "ɹ̠̊˔")
    let VdRetroflexFricative         = Con(Voiced Pulmonic Fricative, Places.Retroflex, "ɻ˔","")
    let sh                           = Con(Pulmonic Fricative Obstruent Continuant, Places.Palatal, "ç", "C")
    let VdPalatalFricative           = Con(Voiced Pulmonic Fricative Obstruent Continuant, Places.Palatal, "ʝ"
, "j\\")
    let xha                          = Con(Pulmonic Fricative Obstruent Continuant, Places.Velar, "x", "x")
    let VdVelarFricative             = Con(Voiced Pulmonic Fricative Obstruent Continuant, Places.Velar, "ɣ",
"G")
    let VlUvularFricative            = Con(Pulmonic Fricative Obstruent Continuant Strident, Places.Uvular,
"χ", "X")
    let VdUvularFricative            = Con(Voiced Pulmonic Fricative Obstruent Continuant Strident Rhotic
Liquid, Places.Uvular, "ʁ", "R")
    let VlPharyngealFricative        = Con(Pulmonic Fricative, Places.Pharyngeal, "ħ", "X\\")
    let VdPharyngealFricative        = Con(Voiced Pulmonic Fricative, Places.Pharyngeal, "ʕ", "?\\")
    let h                            = Con(Pulmonic Fricative, Places.Glottal, "h", "h")
    let VdGlottalFricative           = Con(Voiced Pulmonic Fricative, Places.Glottal, "ɦ", "h\\")

    // Approximant
    let VlLabioDentalApproximant     = Con(Pulmonic Approximant, Places.LabioDental, "ʋ̥", "")
    let VdLabioDentalApproximant     = Con(Voiced Pulmonic Approximant Vocoid Approximant, Places.LabioDental,
"ʋ", "v\\")
    let VlPostalveolarApproximant    = Con(Pulmonic Approximant, Places.Alveolar, "ɹ̥", "")
    let VdPostalveolarApproximant    = Con(Voiced Pulmonic Approximant Vocoid Rhotic Liquid, Places.Alveolar,
"ɹ", "r\\")
    let VlRetroflexApproximant       = Con(Pulmonic Approximant, Places.Retroflex, "ɻ̊", "")
    let VdRetroflexApproximant       = Con(Voiced Pulmonic Approximant Vocoid Rhotic Liquid, Places.Retroflex,
"ɻ", "r\\`")
    let VlPalatalApproximant         = Con(Pulmonic Approximant, Places.Palatal, "j̊", "")
    let jg                           = Con(Voiced Pulmonic Approximant Vocoid Semivowel Continuant, Places.
Palatal, "j", "j")
    let VlVelarApproximant           = Con(Pulmonic Approximant, Places.Velar, "ɰ̊", "")
    let VdVelarApproximant           = Con(Voiced Pulmonic Approximant Vocoid Semivowel Continuant, Places.
Velar, "ɰ", "M\\")
    let VdGlottalApproximant         = Con(Voiced Pulmonic Approximant, Places.Glottal, "ʔ̞", "")

    // Tap or Flap
    let VdBilabialDentalFlap         = Con(Voiced Pulmonic TapFlap, Places.Bilabial, "ⱱ̟", "")
    let VdLabioDentalFlap            = Con(Voiced Pulmonic TapFlap Vibrant, Places.LabioDental, "ⱱ", "")
    let VdLingualLabialStop          = Con(Voiced Pulmonic TapFlap, Places.LinguoLabial, "ɾ̼", "")
    let VlAlveolarFlap               = Con(Pulmonic TapFlap, Places.Alveolar, "ɾ̥", "")
    let VdAlveolarTap                = Con(Voiced Pulmonic TapFlap Rhotic Liquid Vibrant, Places.Alveolar, "ɾ"
, "4")
    let VlRetroflexFlap              = Con(Pulmonic TapFlap, Places.Retroflex, "ɽ̊", "")
    let VdRetroflexFlap              = Con(Voiced Pulmonic TapFlap Rhotic Liquid Vibrant, Places.Retroflex,
"ɽ", "r`")
    let VdUvularFlap                 = Con(Voiced Pulmonic TapFlap, Places.Uvular, "ɢ̆", "")
    let VdPharyngealFlap             = Con(Voiced Pulmonic TapFlap, Places.Pharyngeal, "ʡ̆", "")

    // Trill
    let VlBilabialTrill              = Con(Pulmonic Trill, Places.Bilabial, "ʙ̥", "")
    let VdBilabialTrill              = Con(Voiced Pulmonic Trill Vibrant, Places.Bilabial, "ʙ", "B\\")
```

```
    let VlAlveolarTrill             = Con(Pulmonic Trill, Places.Alveolar, "r̥", "")
    let r                           = Con(Voiced Pulmonic Trill Rhotic Liquid Vibrant, Places.Alveolar, "r",
"r")
    let VlRetroflexTrill            = Con(Pulmonic Trill, Places.Retroflex, "ɽ̊r","")
    let VdRetroflexTrill            = Con(Voiced Pulmonic Trill, Places.Retroflex, "ɽr", "")
    let VlUvularTrill               = Con(Pulmonic Trill Rhotic Liquid Vibrant, Places.Uvular, "ʀ", "R\\")
    let VlPharyngealTrill           = Con(Pulmonic Trill, Places.Pharyngeal, "ʜ", "H\\")
    let VdPharyngealTrill           = Con(Voiced Pulmonic Trill, Places.Pharyngeal, "ʢ", "<\\")

    // Lateral Fricative
    let VlAlveolarLateralFricative  = Con(Pulmonic Lateral Fricative Obstruent Continuant Strident Lateral
Liquid, Places.Alveolar, "ɬ", "K")
    let VdAlveolarLateralFricative  = Con(Voiced Pulmonic Lateral Fricative Obstruent Continuant Strident
Lateral Liquid, Places.Alveolar, "ɮ", "K\\")
    let VlRetroflexLateralFricative = Con(Pulmonic Lateral Fricative, Places.Retroflex, "ɭ̊˔", "")
    let VdRetroflexLateralFricative = Con(Voiced Pulmonic Lateral Fricative, Places.Retroflex, "ɭ˔", "")
    let VlAlveolarPalatalFricative  = Con(Pulmonic Lateral Fricative, Places.Palatal, "ʎ̝̊","")
    let VdAlveolarPalatalFricative  = Con(Voiced Pulmonic Lateral Fricative, Places.Palatal, "ʎ̝", "")
    let VlVelarPalatalFricative     = Con(Pulmonic Lateral Fricative, Places.Velar, "ʟ̝̊","")
    let VdVelarPalatalFricative     = Con(Voiced Pulmonic Lateral Fricative, Places.Velar, "ʟ̝", "")

    // LateralApproximant
    let VlAlveolarLateralApproximant = Con(Pulmonic Lateral Approximant, Places.Alveolar, "l̥", "")
    let l                           = Con(Voiced Pulmonic Lateral Approximant Vocoid Rhotic Liquid Lateral,
Places.Alveolar, "l", "l")
    let ssha                        = Con(Voiced Velarized Pulmonic Lateral Approximant, Places.Alveolar, "ɬ"
, "5")
    let VlRetroflexLateral          = Con(Pulmonic Lateral Approximant, Places.Retroflex, "ɭ̊", "")
    let VdRetroflexLateral          = Con(Voiced Pulmonic Lateral Approximant Vocoid Rhotic Liquid Lateral,
Places.Retroflex, "ɭ", "n`")
    let VlPalatalLateral            = Con(Pulmonic Lateral Approximant, Places.Palatal, "ʎ̥", "")
    let yuh                         = Con(Voiced Pulmonic Lateral Approximant Vocoid Rhotic Liquid Lateral,
Places.Palatal, "ʎ", "L")
    let VlVelarLateral              = Con(Pulmonic Lateral Approximant, Places.Velar, "ʟ̥", "")
    let VdVelarLateral              = Con(Voiced Pulmonic Lateral Approximant Vocoid Rhotic Liquid Lateral,
Places.Velar, "ʟ", "L\\")
    let VdUvularLateral             = Con(Voiced Pulmonic Lateral Approximant, Places.Uvular, "ʟ̠", "")

    // Lateral tap/flap
    let VdAlveolarLateralFlap       = Con(Voiced Pulmonic Lateral TapFlap Vibrant Rhotic Liquid, Places.
Alveolar, "ɺ", "l\\")
    let VdRetroflexLateralFlap      = Con(Voiced Pulmonic Lateral TapFlap, Places.Retroflex, "ɭ̆", "")
    let VdPalatalLateralFlap        = Con(Voiced Pulmonic Lateral TapFlap, Places.Palatal, "ʎ̆", "")
    let VdVelarLateralTap           = Con(Voiced Pulmonic Lateral TapFlap, Places.Velar, "ʟ̆", "")


    //==================================
    // Non-Pulmonic Consonants
    //==================================

    // Clicks
    let VlBilabialTenuisClick       = Con(Ejective Tenuis Click Affricate, Places.Bilabial, "ʘ", "O\\")
    let VdBilabialTenuisClick       = Con(Voiced Ejective Tenuis Click Affricate, Places.Bilabial, "ʘ̬", "")
    let VlDentalTenuisClick         = Con(Ejective Tenuis Click Affricate, Places.Dental, "ǀ", "|")
    let VdDentalTenuisClick         = Con(Voiced Ejective Tenuis Click Affricate, Places.Dental, "ǀ̬", "")
    let VlAlveolarTenuisClick       = Con(Ejective Tenuis Click Affricate, Places.Alveolar, "ǃ", "!\\")
    let VdAlveolarTenuisClick       = Con(Voiced Ejective Tenuis Click Affricate, Places.Alveolar, "ǃ̬", "")
    let VlPalatalTenuisClick        = Con(Ejective Tenuis Click Affricate, Places.Palatal, "ǂ", "=\\")
    let VdPalatalTenuisClick        = Con(Voiced Ejective Tenuis Click Affricate, Places.Palatal, "ǂ̬", "")

    let VlBilabialNasalClick        = Con(Ejective Nasal Click Affricate, Places.Bilabial, "ʘ̃", "")
    let VlDentalNasalClick          = Con(Ejective Nasal Click Affricate, Places.Dental, "ǀ̃", "")
    let VlAlveolarNasalClick        = Con(Ejective Nasal Click Affricate, Places.Alveolar, "ǃ̃", "")
    let VlPalatalNasalClick         = Con(Ejective Nasal Click Affricate, Places.Palatal, "ǂ̃", "")

    let VlAlveolarTenuisLateralClick = Con(Ejective Tenuis Lateral Click Affricate, Places.Alveolar, "ǁ", " |
\\ | \\")
    let VdAlveolarTenuisLateralClick = Con(Voiced Ejective Tenuis Lateral Click Affricate, Places.Alveolar,
"ǁ̬", "")

    let VlBilabialImplosiveClick    = Con(Ejective Implosive Click Affricate, Places.Bilabial, "ɓ̥", "")
    let VdBilabialImplosiveClick    = Con(Voiced Ejective Implosive Click Affricate, Places.Bilabial, "ɓ",
"b_<")
    let VlAlveolarImplosiveClick    = Con(Ejective Implosive Click Affricate, Places.Alveolar, "ɗ̥", "")
    let VdAlveolarImplosiveClick    = Con(Voiced Ejective Implosive Click Affricate, Places.Alveolar, "ɗ",
```

```
"d_<")
    let VlRetroflexImplosiveClick    = Con(Ejective Implosive Click Affricate, Places.Retroflex, "ɖˀ", "")
    let VdRetroflexImplosiveClick    = Con(Voiced Ejective Implosive Click Affricate, Places.Retroflex, "ɖ",
"")
    let VlPalatalImplosiveClick      = Con(Ejective Implosive Click Affricate, Places.Palatal, "ʄˀ", "")
    let VdPalatalImplosiveClick      = Con(Voiced Ejective Implosive Click Affricate, Places.Palatal, "ʄ",
"J\\_<")
    let VlVelarImplosiveClick        = Con(Ejective Implosive Click Affricate, Places.Velar, "gˀ", "")
    let VdVelarImplosiveClick        = Con(Voiced Ejective Implosive Click Affricate, Places.Velar, "g", "g_<"
)
    let VlUvularImplosiveClick       = Con(Ejective Implosive Click Affricate, Places.Uvular, "ɢˀ", "")
    let VdUvularImplosiveClick       = Con(Voiced Ejective Implosive Click Affricate, Places.Uvular, "ɢ",
"G\\_<")

    //==================================
    // Pulmonic Affricates
    //==================================

    // Sibilants
    let VlAlveolarAffricate          = Con(Pulmonic Sibilant Affricate Occlusive Strident, Places.Alveolar,
"ts", "")
    let VdAlveolarAffricate          = Con(Voiced Pulmonic Sibilant Affricate Occlusive Strident, Places.
Alveolar, "dz", "")
    let VlPostalveolarAffricate      = Con(Pulmonic Sibilant Affricate Occlusive Strident, Places.
PalatoAlveolar, "t_ʃ", "")
    let VdPostalveolarAffricate      = Con(Voiced Pulmonic Sibilant Affricate Occlusive Strident, Places.
PalatoAlveolar, "d_ʒ", "")
    let VlRetroflexAffricate         = Con(Pulmonic Sibilant Affricate Occlusive Strident, Places.Retroflex,
"tʂ", "")
    let VdRetroflexAffricate         = Con(Voiced Pulmonic Sibilant Affricate Occlusive Strident, Places.
Retroflex, "dʐ", "")
    let VlAlveoloPalatalAffricate    = Con(Pulmonic Sibilant Affricate Occlusive Strident, Places.
AlveoloPalatal, "tɕ", "")
    let VdAlveoloPalatalAffricate    = Con(Voiced Pulmonic Sibilant Affricate Occlusive Strident, Places.
AlveoloPalatal, "dʑ", "")

    // Non-Sibilants
    let VlBilabialNSAffricate        = Con(Pulmonic Affricate Occlusive, Places.Bilabial, "pɸ", "")
    let VdBilabialNSAffricate        = Con(Voiced Pulmonic Affricate Occlusive, Places.Bilabial, "bβ", "")
    let VlLabioDentalNSAffricate     = Con(Pulmonic Affricate Occlusive Strident, Places.LabioDental, "p_f", ""
)
    let VdLabioDentalNSAffricate     = Con(Voiced Pulmonic Affricate Occlusive Strident, Places.LabioDental,
"b_v", "")
    let VlDentalNSAffricate          = Con(Pulmonic Affricate Occlusive, Places.Dental, "t_θ", "")
    let VdDentalNSAffricate          = Con(Voiced Pulmonic Affricate Occlusive, Places.Dental, "d_ð", "")
    let VlAlveolarNSAffricate        = Con(Pulmonic Affricate, Places.Alveolar, "tɹ̝̊""")
    let VdAlveolarNSAffricate        = Con(Voiced Pulmonic Affricate, Places.Alveolar, "dɹ̝", "")
    let VlPalatoAlveolarNSAffricate  = Con(Pulmonic Affricate, Places.PalatoAlveolar, "t_ɹ̝̊ˈ""")
    let VdPalatoAlveolarNSAffricate  = Con(Voiced Pulmonic Affricate, Places.PalatoAlveolar, "d_ɹ̝ˌ""")
    let VlPalatalNSAffricate         = Con(Pulmonic Affricate Occlusive, Places.Palatal, "cç", "")
    let VdPalatalNSAffricate         = Con(Voiced Pulmonic Affricate Occlusive, Places.Palatal, "ɟʝ", "")
    let VlVelarNSAffricate           = Con(Pulmonic Affricate Occlusive, Places.Velar, "kx", "")
    let VdVelarNSAffricate           = Con(Voiced Pulmonic Affricate Occlusive, Places.Velar, "gɣ", "")
    let VlUvularNSAffricate          = Con(Pulmonic Affricate, Places.Uvular, "qχ", "")
    let VdEpiglottalNSAffricate      = Con(Voiced Pulmonic Affricate, Places.Pharyngeal, "ʡʕ", "")
    let VlGlottalNSAffricate         = Con(Pulmonic Affricate, Places.Glottal, "ʔh", "")

    // Lateral
    let VlAlveolarLateralAffricate   = Con(Pulmonic Lateral Affricate, Places.Alveolar, "tɬ̝̊""")
    let VdAlveolarLateralAffricate   = Con(Voiced Pulmonic Lateral Affricate, Places.Alveolar, "dɮ", "")
    let VlRetroflexLateralAffricate  = Con(Pulmonic Lateral Affricate, Places.Retroflex, "tɭ˔", "")
    let VdPalatalLateralAffricate    = Con(Voiced Pulmonic Lateral Affricate, Places.Palatal, "cʎ̝̊""")
    let VlVelarLateralAffricate      = Con(Pulmonic Lateral Affricate, Places.Velar, "kʟ̝̊""")
    let VdVelarLateralAffricate      = Con(Voiced Pulmonic Lateral Affricate, Places.Velar, "gʟ̝", "")

    //==================================
    // Ejective | Affricates
    //==================================

    // Central
    let VlAlveolarEjectiveAffricate       = Con(Ejective Central Affricate, Places.Alveolar, "ts’", "")
    let VlPalatoAlveolarEjectiveAffricate = Con(Ejective Central Affricate, Places.PalatoAlveolar, "t_ʃ’", "")
    let VlRetroflexEjectiveAffricate      = Con(Ejective Central Affricate, Places.Retroflex, "tʂ’", "")
    let VlVelarEjectiveAffricate          = Con(Ejective Central Affricate, Places.Velar, "kx’", "")
    let VlUvularEjectiveAffricate         = Con(Ejective Central Affricate, Places.Uvular, "qχ’", "")
```

```
    // Lateral
    let VlAlveolarLateralEjective         = Con(Ejective Lateral Affricate, Places.Alveolar, "tɬʼ", "")
    let VlPalatalLateralEjective          = Con(Ejective Lateral Affricate, Places.Palatal, "cʎ̝̥̊;""")
    let VlVelarLateralEjective            = Con(Ejective Lateral Affricate, Places.Velar, "kʟ̝̥̊;""")

    let VlLabialVelarApproximant          = Con(Approximant Vocoid Semivowel Continuant, Places.LabialVelar,
"ʍ", "W")

// Pre-existing
//    let VlAlveoloPalatalFricative1      = NewConsonant(CPM(Categories.Other, Places.AlveoloPalatal)
Fricative, "ɕ", "")
//    let VlAlveoloPalatalFricative       = NewConsonant(CPM(Categories.Other, Places.AlveoloPalatal)
Fricative, "ʑ", "")


    let wh                                = Con(Nasal Voiced Approximant Nasal, Places.LabialVelar, "w\u0303",
"w~")
    let w                                 = Con(Voiced Approximant Vocoid Semivowel Continuant, Places.
LabialVelar, "w", "w")

// This conflicts with another segment (VlPharyngealTrill)

//    let VlEpiglottalFricative             = Con(Fricative, Places.Pharyngeal, "ʜ", "H\\")


    let VdLabialPalatalApproximant        = Con(Voiced Approximant Vocoid Semivowel Continuant, Places.
LabialPalatal, "ɥ", "H")

    let SimultaneousSx                    = Con(Sibilant Fricative, Places.PostAlveolar, "ɧ", "x\\")
    let VdEpiglottalFricative             = Con(Voiced Fricative, Places.Pharyngeal, "ʕ", "?\\")
    let VlEpiglottalPlosive               = Con(Ejective, Places.Pharyngeal, "ʡ", "<\\")

    // Ejectives
    let VlBilabialStopEjective            = Con(Ejective Pulmonic Stop, Places.Bilabial, "p\u02BC", "p_>")
    let VlAveolarStopEjective             = Con(Ejective Stop, Places.Alveolar, "t\u02BC", "t_>")
    let VlRetroflexStopEjective           = Con(Ejective Stop, Places.Retroflex, "ʈ\u02BC", "t`_>")
    let VlPalatalStopEjective             = Con(Ejective Stop, Places.Palatal, "c\u02BC", "c_>")
    let VlVelarStopEjective               = Con(Ejective Stop, Places.Velar, "k\u02BC", "k_>")
    let VlUvularStopEjective              = Con(Ejective Stop, Places.Uvular, "q\u02BC", "q_>")
    let VlEpiglottalStopEjective          = Con(Ejective Stop, Places.Pharyngeal, "ʡ\u02BC", ">\\_>")
    let VlBilabialFricativeEjective       = Con(Ejective Fricative, Places.Bilabial, "ɸ\u02BC", "p\\_>")
    let VlLabiodentalFricativeEjective    = Con(Ejective Fricative, Places.LabioDental, "f\u02BC", "f_>")
    let VlDentalFricativeEjective         = Con(Ejective Fricative, Places.Dental, "θ\u02BC", "T_>")
    let VlAlveolarFricativeEjective       = Con(Ejective Fricative, Places.Alveolar, "s\u02BC", "s_>")
    let VlPostalveolarFricativeEjective   = Con(Ejective Fricative, Places.PostAlveolar, "ʃ\u02BC", "S_>")
    let VlRetroflexFricativeEjective      = Con(Ejective Fricative, Places.Retroflex, "ʂ\u02BC", "s`_>")
    let VlPalatalFricativeEjective        = Con(Ejective Fricative, Places.Palatal, "ɕ\u02BC", "s\\_>")
    let VlVelarFricativeEjective          = Con(Ejective Fricative, Places.Velar, "x\u02BC" , "x_>")
    let VlUvularFricativeEjective         = Con(Ejective Fricative, Places.Uvular, "χ\u02BC", "X_>")
    let VlPostalveolarLatFricEjective     = Con(Ejective Lateral Fricative, Places.Alveolar, "ɬ\u02BC", "K_>")

    //=================================
    // Vowels
    //=================================

    let e                         = Vow(Vocoid Continuant, Opens.Close, Backnesses.Front, "i", "i")
    let E                         = Vow(LongVowel Vocoid Continuant, Opens.Close, Backnesses.Front, "iː", "i:"
)
    let eeh                       = Vow(Rounded Vocoid Continuant, Opens.Close, Backnesses.Front, "y", "y")

    let CloseCentralUnrounded     = Vow(NoFeature, Opens.Close, Backnesses.Central, "ɨ", "1")
    let CloseCentralRounded       = Vow(Rounded, Opens.Close, Backnesses.Central, "ʉ", "}")
    let CloseBackUnrounded        = Vow(Vocoid Continuant, Opens.Close, Backnesses.Back, "ɯ", "M")
    let u                         = Vow(Rounded Vocoid Continuant, Opens.Close, Backnesses.Back, "u", "u")

    let NearCloseCentralUnrounded = Vow(NoFeature, Opens.NearClose, Backnesses.Central, "ɪ", "I\\")
    let NearCloseFrontUnrounded   = Vow(NoFeature, Opens.NearClose, Backnesses.NearFront, "ɪ", "I")
    let NearCloseFrontRounded     = Vow(Rounded, Opens.NearClose, Backnesses.NearFront, "ʏ", "Y")
    let NearCloseCentralRounded   = Vow(Rounded, Opens.NearClose, Backnesses.Central, "ʉ", "U\\")

    let NearCloseBackRounded      = Vow(Rounded, Opens.NearClose, Backnesses.NearBack, "ʊ", "U")
    let U                         = Vow(LongVowel Rounded, Opens.Close, Backnesses.Back, "uː", "u:")

    let ay                        = Vow(Vocoid Continuant, Opens.CloseMid, Backnesses.Front, "e", "e")
```

```
    let CloseMidFrontRounded      = Vow(Rounded Vocoid Continuant, Opens.CloseMid, Backnesses.Front, "ø", "2")
    let MidCentralUnrounded       = Vow(NoFeature, Opens.Mid, Backnesses.Central, "ɘ", "@\\")
    let Schwa                     = Vow(NoFeature, Opens.CloseMid, Backnesses.Central, "ə", "@")
    let ooh                       = Vow(Rounded, Opens.CloseMid, Backnesses.Central, "ɵ", "8")
    let CloseMidBackUnrounded     = Vow(Vocoid Continuant, Opens.CloseMid, Backnesses.Back, "ɤ", "7")
    let oh                        = Vow(Rounded Vocoid Continuant, Opens.CloseMid, Backnesses.Back, "o", "o")

    let MidFrontUnrounded         = Vow(NoFeature, Opens.Mid, Backnesses.Front, "ø̟", "")
    let MidBackUnrounded          = Vow(NoFeature, Opens.Mid, Backnesses.Back, "o̟", "")

    let eh                        = Vow(Vocoid Continuant, Opens.OpenMid, Backnesses.NearFront, "ɛ", "E")
    let ai                        = Vow(LongVowel Vocoid Continuant, Opens.Mid, Backnesses.Front, "eː", "e:")

    let OpenMidNearFrontRounded   = Vow(Rounded Vocoid Continuant, Opens.OpenMid, Backnesses.NearFront, "œ",
"9")
    let OpenMidCentralRounded     = Vow(Rounded, Opens.OpenMid, Backnesses.Central, "ɞ", "3\\")
    let aeh                       = Vow(NoFeature, Opens.OpenMid, Backnesses.Central, "ɜ", "3")
    let OpenMidBackUnrounded      = Vow(Vocoid Continuant, Opens.OpenMid, Backnesses.Back, "ʌ", "V")
    let OpenMidBackRounded        = Vow(Rounded Vocoid Continuant, Opens.OpenMid, Backnesses.Back, "ɔ", "O")
    let O                         = Vow(LongVowel Rounded Vocoid Continuant, Opens.Mid, Backnesses.Back, "oː",
"o:")

    let NearFrontUnrounded        = Vow(NoFeature, Opens.NearOpen, Backnesses.NearFront, "æ", "{")
    let FrontOpenRounded          = Vow(Rounded, Opens.NearOpen, Backnesses.NearFront, "ɶ", "&")
    let OpenMidSchwa              = Vow(Rounded, Opens.NearOpen, Backnesses.Central, "ɐ", "6")

    let OpenCentralUnrounded      = Vow(NoFeature, Opens.Open, Backnesses.Central, "ä", "a_\"")
    let ah                        = Vow(Vocoid Continuant, Opens.Open, Backnesses.NearFront, "a", "a")
    let A                         = Vow(LongVowel Vocoid Continuant, Opens.Open, Backnesses.Central, "aː",
"a:")

    let OpenNearFrontRounded      = Vow(Rounded Vocoid Continuant, Opens.Open, Backnesses.NearFront, "ɶ", "&")
    let OpenBackUnrounded         = Vow(Vocoid Continuant, Opens.Open, Backnesses.Back, "ɑ", "A")
    let OpenBackRounded           = Vow(Rounded Vocoid Continuant, Opens.Open, Backnesses.Back, "ɒ", "Q")

    let uuh                       = Vow(Nasal Rounded, Opens.OpenMid, Backnesses.NearFront, "œ\u0303", "oe*")
// French One "un"
    let ey                        = Vow(Rounded Nasal, Opens.Close, Backnesses.Front, "y\u0303", "y~")
    let aa                        = Vow(Nasal, Opens.Open, Backnesses.NearFront, "a\u0303", "~a")
    let ahn                       = Vow(Nasal Rounded, Opens.NearOpen, Backnesses.Central, "ɐ\u0303", "a*")
// Supposed to be "ɛ˜"as in French Dog "chien"?
    let oon                       = Vow(Nasal Rounded, Opens.OpenMid, Backnesses.Back, "ɔ\u0303", "o*")
// French Fish "poisson"
    let aehn                      = Vow(Nasal, Opens.OpenMid, Backnesses.NearFront, "ɜ\u0303", "3*")
    let ehnn                      = Vow(Nasal, Opens.OpenMid, Backnesses.NearFront, "ɛ\u0303", "E*")
    let uh                        = Vow(Nasal Rounded, Opens.Close, Backnesses.Back, "u\u0303", "u*")
// Portuguese One "um"
    let en                        = Vow(Nasal, Opens.CloseMid, Backnesses.Front, "e\u0303", "e*")
// Portuguese Trail "se*da
    let een                       = Vow(Nasal, Opens.Close, Backnesses.Front, "i\u0303", "i*")
// Portuguese Tongue


// How to show rhotic vowels?
//    let OpenMidCentralRhotic       = Vow(Rhotic, Opens.OpenMid, Backnesses.Central, "ɝ", "")
//    let RhoticSchwa                = Vow(Rhotic, Opens.Open, Backnesses.NearFront, "ɚ", "@`")
  end

  with ImpossibleSegments
    let I10 = Imp(Pulmonic Nasal, Places.Pharyngeal)
    let I11 = Imp(Pulmonic Nasal, Places.Glottal)
    let I12 = Imp(Pulmonic Stop Voiced, Places.Pharyngeal)
    let I13 = Imp(Pulmonic Stop Voiced, Places.Glottal)
    let I14 = Imp(Pulmonic Sibilant Fricative, Places.Bilabial)
    let I15 = Imp(Pulmonic Sibilant Fricative, Places.LabioDental)
    let I16 = Imp(Pulmonic Sibilant Fricative, Places.LinguoLabial)
    let I17 = Imp(Pulmonic Sibilant Fricative, Places.Velar)
    let I18 = Imp(Pulmonic Sibilant Fricative, Places.Uvular)
    let I19 = Imp(Pulmonic Sibilant Fricative, Places.Pharyngeal)
    let I20 = Imp(Pulmonic Sibilant Fricative, Places.Glottal)
    let I21 = Imp(Pulmonic Trill, Places.Velar)
    let I22 = Imp(Pulmonic Trill, Places.Glottal)
    let I23 = Imp(Pulmonic TapFlap, Places.Velar)
    let I24 = Imp(Pulmonic TapFlap, Places.Glottal)
    let I25 = Imp(Pulmonic Lateral Fricative, Places.Bilabial)
```

```
  let I26 = Imp(Pulmonic Lateral Fricative, Places.LabioDental)
  let I27 = Imp(Pulmonic Lateral Fricative, Places.Pharyngeal)
  let I28 = Imp(Pulmonic Lateral Fricative, Places.Glottal)
  let I29 = Imp(Pulmonic Lateral Approximant, Places.Bilabial)
  let I30 = Imp(Pulmonic Lateral Approximant, Places.LabioDental)
  let I31 = Imp(Pulmonic Lateral Approximant, Places.Pharyngeal)
  let I32 = Imp(Pulmonic Lateral Approximant, Places.Glottal)
  let I33 = Imp(Pulmonic Lateral TapFlap, Places.Bilabial)
  let I34 = Imp(Pulmonic Lateral TapFlap, Places.LabioDental)
  let I35 = Imp(Pulmonic Lateral TapFlap, Places.Pharyngeal)
  let I36 = Imp(Pulmonic Lateral TapFlap, Places.Glottal)
  let I37 = Imp(Pulmonic Sibilant Affricate, Places.Bilabial)
  let I38 = Imp(Pulmonic Sibilant Affricate, Places.LabioDental)
  let I39 = Imp(Pulmonic Sibilant Affricate, Places.Velar)
  let I40 = Imp(Pulmonic Sibilant Affricate, Places.Uvular)
  let I41 = Imp(Pulmonic Sibilant Affricate, Places.Pharyngeal)
  let I42 = Imp(Pulmonic Sibilant Affricate, Places.Glottal)
  let I43 = Imp(Pulmonic Lateral Affricate, Places.Bilabial)
  let I44 = Imp(Pulmonic Lateral Affricate, Places.LabioDental)
  let I45 = Imp(Pulmonic Lateral Affricate, Places.Pharyngeal)
  let I46 = Imp(Pulmonic Lateral Affricate, Places.Glottal)
  let I47 = Imp(Ejective Lateral Fricative Affricate, Places.Bilabial)
  let I48 = Imp(Ejective Lateral Fricative Affricate, Places.LabioDental)
  let I49 = Imp(Ejective Lateral Fricative Affricate, Places.Pharyngeal)
  let I50 = Imp(Ejective Central Affricate, Places.Glottal)
  let I51 = Imp(Ejective Lateral Affricate, Places.Bilabial)
  let I52 = Imp(Ejective Lateral Affricate, Places.LabioDental)
  let I53 = Imp(Ejective Lateral Affricate, Places.Pharyngeal)
  let I54 = Imp(Ejective Lateral Affricate, Places.Glottal)
  let I55 = Imp(Ejective Tenuis Click Affricate, Places.Velar)
  let I56 = Imp(Ejective Tenuis Click Affricate, Places.Uvular)
  let I57 = Imp(Ejective Tenuis Click Affricate, Places.Pharyngeal)
  let I58 = Imp(Ejective Nasal Click Affricate, Places.Velar)
  let I59 = Imp(Ejective Nasal Click Affricate, Places.Uvular)
  let I60 = Imp(Ejective Nasal Click Affricate, Places.Pharyngeal)
  let I61 = Imp(Ejective Tenuis Lateral Click Affricate, Places.Bilabial)
  let I62 = Imp(Ejective Tenuis Lateral Click Affricate, Places.LabioDental)
  let I63 = Imp(Ejective Tenuis Lateral Click Affricate, Places.Velar)
  let I64 = Imp(Ejective Tenuis Lateral Click Affricate, Places.Uvular)
  let I65 = Imp(Ejective Tenuis Lateral Click Affricate, Places.Pharyngeal)
end

let AllSegments = Results.UsedSegments + ImpossibleSegments

with DiacriticModifiers
  let UndefinedEscapeCharacter     = Diac("Undefined escape character", null, "*")
  let Nasalized                    = Diac("Nasalized", "\u0303", "_~")
  let Centralized                  = Diac("Centralized", "\u0308", "_\"")
  let Advanced                     = Diac("Advanced", "\u031F", "_+")
  let Retracted                    = Diac("Retracted", "\u0320", "_-")
  let RisingTone                   = Diac("RisingTone ", "\u030C", "_R")
  let Voiceless                    = Diac("Voiceless", "\u0325", "_0")
  let Implosive                    = Diac("Implosive", null, "_<")
  let Syllabic                     = Diac("Syllabic", "\u0329", "_=")
  let Ejective                     = Diac("Ejective", "\u02BC", "_>")
  let Pharyngealized               = Diac("Pharyngealized ", "\u02E4", "_?\\")
  let FallingTone                  = Diac("Falling tone", "\u0302", "_F")
  let NonSyllabic                  = Diac("Non-syllabic", "\u032F", "_^")
  let NoAudibleRelease             = Diac("No audible release", "\u031A", "_}")
  let RhoticHook                   = Diac("Rhotic hook", "\u02DE", "`")
  let AdvancedTongueRoot           = Diac("Advanced tongue root ", "\u0318", "_A")
  let Apical                       = Diac("Apical", "\u033A", "_a")
  let ExtraLowTone                 = Diac("Extra low tone", "\u030F", "_B")
  let LowRisingTone                = Diac("Low rising tone", "\u1DC5", "_B_L")
  let LessRounded                  = Diac("Less rounded", "\u031C", "_c")
  let Dental                       = Diac("Dental", "\u032A", "_d")
  let VelarizedOrPharyngealized    = Diac("Velarized or Pharyngealized", "\u0334", "_e")
  let GlobalFall                   = Diac("Global fall", "\u2198", "<F>")
  let Velarized                    = Diac("Velarized", "\u02E0", "_G")
  let HighTone                     = Diac("High tone", "\u0301", "_H")
  let HighRisingTone               = Diac("High rising tone", "\u1DC4", "_H_T")
  let Aspirated                    = Diac("Aspirated", "\u02B0", "_h")
  let Palatalized                  = Diac("Palatalized", "\u02B2", "_j")
  let CreakyVoiced                 = Diac("Creaky voiced", "\u0330", "_k")
  let LowTone                      = Diac("Low tone", "\u0300", "_L")
```

```
    let LateralRelease              = Diac("Lateral release", "\u02E1", "_l")
    let MidTone                     = Diac("Mid tone", "\u0304", "_M")
    let Laminal                     = Diac("Laminal", "\u033B", "_m")
    let LinguoLabial                = Diac("Linguo-Labial", "\u033C", "_N")
    let NasalRelease                = Diac("Nasal release", "\u207F", "_n")
    let MoreRounded                 = Diac("More rounded", "\u0339", "_O")
    let Lowered                     = Diac("Lowered", "\u031E", "_o")
    let RetractedTongueRoot         = Diac("Retracted tongue root", "\u0319", "_q")
    let GlobalRise                  = Diac("Global rise", "\u2197", "<R>")
    let RisingFallingTone           = Diac("Rising falling tone", "\u1DC8", "_R_F")
    let Raised                      = Diac("Raised", "\u031D", "_r")
    let ExtraHighTone               = Diac("Extra high tone", "\u030B", "_T")
    let BreathyVoiced               = Diac("Breathy voiced", "\u0324", "_t")
    let Voiced                      = Diac("Voiced", "\u032C", "_v")
    let Labialized                  = Diac("Labialized", "\u02B7", "_W")
    let ExtraShort                  = Diac("Extra short", "\u02D8", "_X")
    let MidCentralized              = Diac("Mid-centralized", "\u033D", "_x")
    let Downstep                    = Diac("Down-step", "↓", "!")
    let Upstep                      = Diac("Up-step", "↑", "^")
    let SylableBreak                = Diac("Sylable break", ".", ".")
    let PrimaryStress               = Diac("Primary stress", "ˈ", "\"")
    let SecondaryStress             = Diac("Secondary stress", "ˌ", "%")
    let Long                        = Diac("Long", "ː", ":")
    let HalfLong                    = Diac("Half-long", "ˑ", ":\\")
    let IndeterminacyinFrenchVowels = Diac("Indeterminacy in french vowels", null, "/")
    let BeginNonsegmentalNotation   = Diac("Begin Non-segmental notation", null, "<")
    let Endnonsegmentalnotation     = Diac("End non-segmental notation", "", ">")
    let Voicedepiglottalfricative   = Diac("Voiced epiglottal fricative", "ʢ", "<\\")
    let Postalveolarclick           = Diac("Post-alveolar click", "!", "!\\")
    let MinorGroup                  = Diac("Minor group", " | ", " | ")
    let Dentalclick                 = Diac("Dental click", "|", " | \\")
    let MajorGroup                  = Diac("Major group", "‖", " |  | ")
    let Alveolarlateralclick        = Diac("Alveolar lateral click", "‖", " | \\ | \\")
    let Palatalclick                = Diac("Palatal click", "ǂ", "")
//    let Linkingmark                 = Diac("Linking Mark", "\u203f", "-\\")
    let VoicelessDescender          = Diac("Voiceless descender", "\u030A", "")
    let CombiningMacron             = Diac("Combining macron", "\u0331", "")
    let TieBarBelow                 = Diac("Tie-bar below", "\u035C", "")
    let TieBarAbove                 = Diac("Tie-bar above", "\u0361", "")
    let ReadyMadeCombination        = Diac("Ready made combination", "\u026B", "")
    let Becomes                     = Diac("Becomes", "→", "")
    let Separator                   = Diac("Separator", "", "-")
  end

  //=========================================================================

  let HasFeature(feature, f)    = (feature & f) != NoFeature
  let NotFeature(feature, f)    = (feature & f) == NoFeature
  let HasMask(feature, mask, f) = (feature & mask) == (f & mask)

  //=========================================================================

  let SegmentColumns = 4
  let SegmentSize    = 40 pts

  let ShowSampa(sampa) = Span {
    Style.MonoFamily,
    sampa,
  }

  let SegmentName(segment) = Span {
    if (segment.Features.HasFeature(Vowel))
      segment.Open,
      Space,
      segment.Backness,
    else
      segment.Place,
    end,
    Space,
    Span {
      Separator: Space,
      if (segment.Diacritic)
        segment.Description
      else
        each segment.Features,
```

```
    end
  }
}

let SegmentDisplay(segment, location) = Block {
  ParAlignment: ParAlignments.Center,
  Paragraph {
    LocationMark: location,
    SpaceBefore: 8 pts,
    SpaceAfter: 8 pts,
    TextHeight: SegmentSize,
    Style.IPAFamily,
    SpaceAfter: SegmentSize * 0.125,
    segment.Ipa,
  },
  Paragraph {
    TextHeight: 10 pts,
    SegmentName(segment),
  },
  Paragraph {
    TextHeight: 8 pts,
    ParBackground: 95%,
    Style.MonoFamily,
    TextRadix: 16,
    TextDigits: 4,
    Span {
      Separator: Lang.Separator,
      if (segment.Ipa)
        Type.Integer(each segment.Ipa)
      end
    }
  },
  Paragraph {
    TextHeight: 8 pts,
    if (segment.Sampa and segment.Sampa.Length > 0)
      ParBackground: 90%,
      ShowSampa(segment.Sampa)
    else
      if (segment.Ipa.Length > 0)
        "No Sampa" {TextColor: Colors.White},
        ParBackground: Colors.Red,
      end
    end
  },
}

let SegmentPopup(segment) = Frame {
  Width: 2 inches,
  SegmentDisplay(segment, null)
}

let SegmentCell(ref segment) = Cell {
  Edge: 0.25 pts {Color: Colors.LightGray},
  Padding: 2 pts,
  SegmentDisplay(segment, segment.FullSymbolName)
}

let SegmentRow(segments) = Row {
  SegmentCell(each segments)
}

let AlphaOrder(x, y) begin
  var cl = Math.Compare(x.Ipa.Length, y.Ipa.Length);
  if (cl == 0)
    cl = -Math.Compare(x.Ipa, y.Ipa)
  end
  return cl;
end

let SortedSegments = Results.UsedSegments.Sort(false, AlphaOrder)

let SegmentTable = Block {
  Table {
    Columns: [Metrics.Content.Width / SegmentColumns] * SegmentColumns,
    Style.TitleBar(Lang.IPAListing, SegmentColumns),
```

```
      SegmentRow(each (SortedSegments / SegmentColumns))
    },
    Style.TableNotes
  }

  let ShowIPA(word) = Span {
    (each word.Segments).Ipa
  }

  let ShowCharacter(c) = Span {
    Popup: SegmentPopup.Call(c.Segment),
//    Link: c.Segment.FullSymbolName,
    c.Character
  }

  let ShowSegment(ref segment) = Span {
    Style.IPAFamily,
    Popup: SegmentPopup.Call(segment),
//    Link: segment.FullSymbolName,
    if (segment.Impossible)
      Assert(false, Lang.Impossible),
      TextColor: Colors.Red,
    end,
    if (segment.Diacritic)
      "  ",
    end,
    segment.Ipa
  }

  let SegmentSound(ref segment) begin
    if (segment == SpaceSegment)
      return Space;
    end
    return segment.SymbolName
  end

  let LangHasMeaning(word, meaning) = word.Meaning == meaning
  let FindWordsWithMeaning(ref language, meaning) = language.Words.FindSlice(LangHasMeaning, meaning)

  //=========================================================================
  // Build a dictionary with SAMPA text as the key
  //=========================================================================

  let GatherText(set, ref segment) begin
    if (segment.Sampa.Length > 0)
      set.AddElement(segment.Sampa, ref segment)
    end
  end

  let SampaSet begin
    var set = Type.Dictionary(128);
    GatherText(set, each Segments);
    set.AddElement(" ", ref SpaceSegment);
    return set;
  end

  //=========================================================================
  // Code for Euler segment diagram
  //=========================================================================

  let ChartSize = 6 inches
  let EX(x)     = ChartSize * x * 0.01
  let EY(y)     = ChartSize * y * 0.01

  let Enclosure(x, y, w, h, color) begin
    var size = Size(EX(w), EY(h));
    return Canvas {
      X: EX(x),
      Y: EY(y),
      Size: size,
      Figure {
        Stroke: 1 pts,
        Fill: color,
        Rectangle(Rect(0, size), Size(6 pts))
      }
```

```
    }
  end

  let NameBox(name, angle=0) = Paragraph {
    Transform: Rotate(angle),
    Space,
    name Bold,
    Space,
  }

  let FeatureMask = Vowel Nasal Vocoid LongVowel Semivowel Approximant Vibrant Lateral Affricate Occlusive
Strident Sibilant Obstruent Continuant Fricative Rhotic Liquid Trill TapFlap
  let MatchFeature(segment, data) = HasMask(segment.Features, FeatureMask, data)
  let FeatureSegments(features)   = Results.UsedSegments.FindSlice(MatchFeature, features)

  let FeatureFrame(x, y, width, name, features) = Canvas {
    X: EX(x),
    Y: EY(y),
    Frame {
      Width: EX(width),
      ParAlignment: ParAlignments.Center,
      if (name)
        name Bold,
      end,
      Paragraph {
        Separator: Space,
        IPA.ShowSegment(each FeatureSegments(features))
      }
    }
  }

  let VNameFrame(name, x, y, width, height, color) = Enclosure(x, y, width, height, color) {
    VAlign: VAligns.Center,
    NameBox(name, 90 degrees)
  }

  let FeatureChart = Canvas {
    TextHeight: 16 pts,

    VNameFrame(Lang.Occlusive, 10, 0, 90, 34, Color(255, 238, 238)) {
      HAlign: HAligns.Right,
    },
    VNameFrame(Lang.Continuant, 10, 35, 90, 37, Color(229, 255, 255)) {
      HAlign: HAligns.Right,
    },
    VNameFrame(Lang.Obstruent, 0, 11, 94, 40, Color(238, 238, 255, 50%)),
    VNameFrame(Lang.Vocoid, 0, 52, 78, 21, Color(238, 255, 238, 50%)),
    VNameFrame(Lang.Vibrant, 20, 74, 65, 17, Color(238, 238, 255)),

    Enclosure(11, 1, 83, 9, Color(255, 246, 246)),    // Nasals
    Enclosure(31, 12, 62, 21, Color(246, 242, 250)),  // Affricates
    Enclosure(11, 12, 19, 21, Color(246, 243, 250)),  // Plosives
    Enclosure(11, 36, 80, 14, Color(240, 247, 255)),  // Fricatives

    Enclosure(25, 75, 59, 7, Color(247, 247, 255)) {
      HAlign: HAligns.Left,
      VAlign: VAligns.Center,
      NameBox(Lang.TapFlap),
    },
    Enclosure(25, 83, 53, 7, Color(247, 247, 255)) {
      HAlign: HAligns.Left,
      VAlign: VAligns.Center,
      NameBox(Lang.Trill),
    },

    Enclosure(53, 13, 39, 36, Color(233, 248, 235, 50%)) {
      HAlign: HAligns.Center,
      NameBox(Lang.Strident),
    },

    Enclosure(62, 18, 25, 24, Color(245, 252, 220, 50%)), // Sibilants
    Enclosure(11, 53, 21, 18, Color(240, 255, 247)),      // Vowels
    Enclosure(33, 53, 61, 18, Color(240, 255, 247, 50%)) {
      HAlign: HAligns.Left,
      NameBox(Lang.Approximant),
```

```
  },

  Enclosure(35, 59, 21, 11, Color(248, 255, 225)), // Semivowels

  Enclosure(64, 43, 26, 59, Color(255, 238, 238, 50%)) {
    HAlign: HAligns.Center,
    VAlign: VAligns.Bottom,
    NameBox(Lang.Liquid),
  },

  Enclosure(65, 44, 14, 52, Color(255, 247, 221, 50%)) {
    HAlign: HAligns.Center,
    VAlign: VAligns.Bottom,
    NameBox(Lang.Rhotic),
  },

  Enclosure(80, 44, 9, 44, Color(238, 247, 230, 70%)) {
    HAlign: HAligns.Right,
    VAlign: VAligns.Bottom,
    NameBox(Lang.Lateral, 90 degrees),
  },

  FeatureFrame(11, 1, 84, Lang.Nasal, Nasal Occlusive),
  FeatureFrame(12, 14, 17, Lang.Plosive, Stop Occlusive),
  FeatureFrame(32, 14, 20, Lang.Affricate, Affricate Occlusive),
  FeatureFrame(54, 20, 7, null, Affricate Occlusive Strident),
  FeatureFrame(65, 18, 18, Lang.Sibilant, Affricate Occlusive Strident Sibilant),
  FeatureFrame(14, 38, 22, Lang.Fricative, Fricative Obstruent Continuant),
  FeatureFrame(54, 38, 6, null, Fricative Obstruent Continuant Strident),
  FeatureFrame(64, 36, 20, null, Fricative Obstruent Continuant Strident Sibilant),
  FeatureFrame(66, 44, 10, null, Fricative Obstruent Continuant Strident Rhotic Liquid),
  FeatureFrame(80, 44, 10, null, Fricative Obstruent Continuant Strident Lateral Liquid),
  FeatureFrame(12, 53, 19, Lang.Vowel, Vowel Vocoid Continuant),
  FeatureFrame(36, 60, 19, Lang.SemiVowel, Vocoid Semivowel Approximant Continuant),
  FeatureFrame(57, 62, 5, null, Vocoid Approximant),
  FeatureFrame(66, 60, 10, null, Vocoid Approximant Rhotic Liquid),
  FeatureFrame(82, 58, 5, null, Vocoid Approximant Rhotic Liquid Lateral),
  FeatureFrame(54, 76, 10, null, Vibrant TapFlap),
  FeatureFrame(66, 76, 10, null, Vibrant TapFlap Rhotic Liquid),
  FeatureFrame(77, 77, 10, null, Vibrant TapFlap Rhotic Liquid Lateral),
  FeatureFrame(54, 84, 10, null, Vibrant Trill),
  FeatureFrame(66, 84, 10, null, Vibrant Trill Rhotic Liquid),
}
//=========================================================================
// Code for drift diagram
//=========================================================================

let RowText     = 8 pts
let RowHeight   = 9 pts
let BoxWidth    = 6 pts
let BorderSize  = 0.33 pts
let FeatureList = FeatureSet[1..]

let FeaturePopup(feature) = Frame {
  Width: 2 inches,
  Paragraph {
    feature.Name
  }
}

let RowBox = Canvas {
  Height: RowHeight,
  Width: BoxWidth,
  HAlign: HAligns.Center,
  VAlign: VAligns.Center,
}

let ShowFeature(feature) = Canvas {
  Height: RowHeight,
  VAlign: VAligns.Center,
  feature.Name
}

let FeatureBox(feature, features0, features1) = RowBox {
  if (features0.HasFeature(feature))
```

```
    if (not features1.HasFeature(feature))
      Background: Colors.Red,
    end
  else
    if (features1.HasFeature(feature))
      Background: Colors.Green,
    end
  end
}

let FeatureSegment(ref segment0, ref segment1) = Group {
  Vertical: true,
  VAlign: VAligns.Center,
  FeatureBox(each FeatureList, segment0.Features, segment1.Features),
}

let DriftFeatures(line0, line, index) = Group {
  Vertical: true,
  Group {
    Vertical: true,
    RowBox {
      if (EachIndex == 0)
        ShowSegment(line0.Segments[index]),
      end,
    },
    RowBox {
      ShowSegment(line.Segments[index]),
    },
    BorderB: BorderSize,
  },
  Group {
    FeatureSegment(line0.Segments[index], line.Segments[index]),
  }
}

let SegmentLength(line) = line.Segments.Length
let MinLength(lines)    = Math.Min(SegmentLength(each lines))

let DriftMatrix(lines, index) = Group {
  if (index != 0)
    BorderL: BorderSize,
  end,
  DriftFeatures(lines[0], each lines[1..], index)
}

let DriftChart(lines) = Group {
  TextHeight: RowText,
  VAlign: VAligns.Top,
  Border: BorderSize,
  Group {
    Vertical: true,
    BorderR: BorderSize,
    Group {
      Vertical: true,
      VAlign: VAligns.Center,
      RowBox,
      RowBox,
    },
    Group {
      Vertical: true,
      PaddingR: 3 pts,
      BorderT: BorderSize,
      ShowFeature(each FeatureList)
    },
  },
  DriftMatrix(lines, each 0..<MinLength(lines))
}

let DriftMeaning(meaning) = Group {
  MarginB: 8 pts,
  Vertical: true,
  Paragraph {
    Lang.Meaning, ": ",
    meaning Bold,
  },
```

```
    DriftChart(Results.GetGeneText(meaning, each Results.UsedLanguages)),
  }

  let DriftGroup(meanings) = Group {
    DriftMeaning(each meanings)
  }

  let DriftSection = Block {
    Paragraph {
      Separator: "  ",
      DriftGroup(each Results.UsedMeanings)
    }
  }
end
```

Style.nytril

```
using Type, Format, Units, Math, IO
//=========================================================================

with Metrics
  let MarginL    = 0.75 inches
  let MarginR    = MarginL
  let MarginT    = 0.5 inches
  let MarginB    = 0.4 inches
  let Paper      = Type.Size(8.5 inches, 11 inches)
  let Content    = Type.Size(Paper.Width - MarginL - MarginR, Paper.Height - MarginT - MarginB)
  let TableSpace = 24 pts
  let TreeWidth  = Content.Width

  let BoxSize    = Type.Size(18 pts, 20 pts)
  let CellSize   = Type.Size(BoxSize.Width * 2, BoxSize.Height)
end
//=========================================================================

with Style
  let MainFamily           = {TextFamily: TextFamilies.TimesNewRoman}
  let SansSerif            = {TextFamily: TextFamilies.Calibri}
  let MonoFamily           = {TextFamily: TextFamilies.Consolas}
  let IPAFamily            = {TextFamily: TextFamilies.CambriaMath} // Also can be "Linux Libertine O"

  let ImpossibleBackground = {Background: 80%}
  let Used                 = {TextColor: Colors.Red}
  let TitleBackground      = {Background: 90%}
  let TableEdge            = {Edge: 0.3 pts}
  let ColumnEdge           = {EdgeR: 0.3 pts}
  let SegmentBottom        = {EdgeB: 0.25 pts {Color: 80%}}


  let RowBar(i) = {
    if (i mod 2 != 0)
      Background: 95%
    end
  }

  let WhitePaper = Document {
    Size: Metrics.Paper,
    MainFamily,
    TextHeight: 11.5 pts,
    MarginL: Metrics.MarginL,
    MarginR: Metrics.MarginR,
    MarginT: Metrics.MarginT,
    MarginB: Metrics.MarginB,
  }

  let NormalHeader(text) = Block {
    Span {{TextUppercase: true} text}
  }

  let PageSection = Section {
    SectionBreak: SectionBreaks.NextPage,
    Footer: Block {
      Distance: 0.5 inches,
      ParAlignment: ParAlignments.Center,
```

```
    PageNumber
  },
}

let TableNotes = Paragraph {
  SpaceAfter: Metrics.TableSpace,
}

let TitleWord(text) = Span {
  text[0],
  {TextHeight: 60%} text[1..]
}

let TitleCase(text) = Span {
  if (text)
    TextUppercase: true,
    Separator: Space,
    TitleWord(each text.Split(Space))
  end
}

let Title(text) = Paragraph {
  KeepWithNext: true,
  SpaceBefore: 12 pts,
  SpaceAfter: 6 pts,
  ParAlignment: ParAlignments.Center,
  TextHeight: 18 pts,
  text
}

let HeaderCentered(text) = Paragraph {
  KeepWithNext: true,
  SpaceBefore: 12 pts,
  SpaceAfter: 6 pts,
  ParAlignment: ParAlignments.Center,
  TitleCase(text)
}

let Header1(text) = Paragraph {
  KeepWithNext: true,
  SpaceBefore: 18 pts,
  SpaceAfter: 8 pts,
  TitleCase(text)
}

let Header2(text) = Paragraph {
  KeepWithNext: true,
  SpaceBefore: 12 pts,
  SpaceAfter: 6 pts,
  TextHeight: 14 pts,
  text
}

let Header3(text) = Paragraph {
  KeepWithNext: true,
  Border: 0.25 pts,
  ParBackground: 97%,
  SpaceAfter: 12 pts,
  TextHeight: 14 pts,
  text
}

let SourceFile(ref source) = Block {
  Style.Header3(source.Path.GetFileName),
  Paragraph {
    LeftIndent: 0.25 inches,
    ParAlignment: ParAlignments.Left,
    Style.MonoFamily,
    TextHeight: 8 pts,
    SourceSelection(source)
  }
}

let SourceCodeBlock = TextBlock {
  Style.MonoFamily,
```

```
  TextHeight: 10 pts,
  ParBackground: 97%,
}

let Author(author) = Span {
  Link: author.FullSymbolName,
  author.Title,
}

let Collect(list, node) begin
  if (node.Label)
    list.AddElement(node.Label, node)
  end
  Collect(list, each node);
end

let GetTimes(node) begin
  var list = Type.List(100);
  Collect(list, node);
  return list[1..];
end

let ByYear(x, y) = x.Branch.Compare(y.Branch)

let TimelineNodes = GetTimes(Info.LanguageTree).Sort(true, ByYear)

let TimelineRow(options, node) = Group {
  X: options.Width - options.Width * node.Branch / options.MaxYear,
  Figure {
    Fill: Colors.Green,
    Rectangle(Type.Rect(Type.Point(0), Type.Size(5, options.TextHeight)))
  },
  Paragraph {
    Space,
    node.Label,
  }
}

let ShowTimeline(options) = Group {
  Vertical: true,
  TextHeight: options.TextHeight,
  TimelineRow(options, each TimelineNodes),
  Group {
    Vertical: true,
    Background: 90%,
    ChartAxis {
      Width: options.Width,
      Start: options.MaxYear,
      Stop: 0,
    },
    Frame {
      HAlignment: HAligns.Center,
      "Years"
    }
  }
}

let ShowLanguageTree = Block {
  ShowTree(Info.LanguageTree),
  Header2("Last Branch"),
  ShowTimeline({
    Width: Metrics.TreeWidth,
    TextHeight: 10 pts,
    MaxYear: Math.Max((each TimelineNodes).Branch)
  })
}

let ShowTree(tree) = Group {
  Vertical: true,
  Tree {
    Curvature: 30%,
    Bevel: 20%,
    Marker: {Style.IPAFamily, TextHeight: 4 pts}Chars.Circle {TextColor: Colors.Gray},
    Width: Metrics.TreeWidth,
    ValueLabel: Lang.Years,
```

```
    ValueAxis: ChartAxis,
    tree
  },
}

let ShowAbstract(content) = Block {
  HeaderCentered(content.Title),
  Block {
    content.Body
  }
}

let ShowContent(content) = Block {
  Header1(content.Title),
  Block {
    content.Body
  }
}

let ShowAuthorFull(author) = Block {
  LeftIndent: 0.25 inches,
  Paragraph {
    LocationMark: author.FullSymbolName,
    FirstIndent: -0.25 inches,
    Span {
      Separator: Space,
      if (author.Website)
        Link: author.Website,
      end,
      author.First, author.Middle, author.Last,
    }
  },
  Span {
    Separator: Lang.Separator,
    author.Address
  },
  Span {"{0}: "(Lang.EMail), author.EMail},
  Empty
}

let AppendixRow(appendix) = Paragraph {
  LeftIndent: 20 pts,
  FirstIndent: -20 pts,
  EachIndex + 1,
  ")",
  Tab,
  Span {
    Link: Lang.Appendix + EachIndex,
    appendix.Title
  }
}

let ShowAppendixTable = Block {
  HeaderCentered(Lang.Appendices),
  AppendixRow(each Appendix)
}

let ShowAuthors(authors) = Block {
  HeaderCentered(Lang.Authors),
  ShowAuthorFull(each authors),
}

let ShowReference(ref r) = Paragraph {
  LeftIndent: 0.25 inches,
  FirstIndent: -0.25 inches,
  if (r.Author)
    TitleCase(r.Author),
    if (r.Title)
      Lang.Separator,
      r.Title,
    end,
  else
    r.Title
  end,
  if (r.Year)
```

```
    Lang.Separator,
    r.Year,
  end,
  if (r.Publisher)
    Lang.Separator,
    Italic r.Publisher,
    r.Page,
  end,
  if (r.Link)
    ". : ",
     Span {
       Link: r.Link,
       TextColor: Colors.DarkBlue,
       r.Link
      }
  end,
}

let ShowReferences(references) = Block {
  HeaderCentered(Lang.References),
  ShowReference(each references)
}

let ShowAppendix(appendix) = PageSection {
  Paragraph {
    BorderB: 1 pts,
    ParAlignment: ParAlignments.Center,
    TextHeight: 14 pts,
    SpaceAfter: 8 pts,
    LocationMark: Lang.Appendix + EachIndex,
    "{0} {1} - "(Lang.Appendix, EachIndex+1),
    appendix.Title,
  },
  Block {
    appendix.Content
  }
}

let LanguageRow(lang) = Row {
  Background: ((EachIndex mod 2) == 0 ? Colors.White : 97%),
  lang.Name,
}

let ShowLanguageList(list) = Table {
  PaddingLR: 2,
  Columns: [1.5 inch],
  Row {
    Background: Colors.DarkGray,
    TextColor: Colors.White,
    Lang.Name,
  },
  Edge: 0.25 pts {Color: Colors.DarkGray},
  LanguageRow(each list)
}

let HeaderCell(d, halign=HAligns.Left) = Cell {
  HAlign: halign,
  VAlign: VAligns.Center,
  Style.SansSerif,
  Style.TitleBackground,
  EdgeB: 1 pts,
  Padding: 2 pts,
  d
}

let TitleBar(name, columns) = Row {
  Cell {
    Padding: 2 pts,
    ParAlignment: ParAlignments.Center,
    Background: 40%,
    TextHeight: 16 pts,
    TextColor: Colors.White,
    ColumnSpan: columns,
    name
  }
```

```
  }
end
//========================================================================

let Logo = Frame {
  Width: 5 inches,
  Height: 0.5 inches,
  Padding: 4 pts,
  Background: Type.Color(51, 66, 81),
  Table {
    Columns: [4.3 inches, 0.6 inches],
    Row {
      Block {
        "Transactions of the" {TextColor: Type.Color(129, 166, 207), Bold, TextHeight: 12 pts},
        "Bayesian Society" {Bold, TextHeight: 20 pts, TextColor: Colors.White}
      },
      Read(Folders.Source FileName("bayes") Extensions.PNG) {Width: 0.5 inches}
    }
  }
}
//========================================================================
```

References.nytril

```
using IO, Format
//========================================================================

with Authors
  with DMGoldstein
    let Title  = "D. M. Goldstein"
    let First  = "David"
    let Middle = "M."
    let Last   = "Goldstein"
    let Address = ["UCLA", "Los Angeles, CA 90095-1543", "USA"]
    let Website = Domain("https://linguistics.ucla.edu") Folder("person") Folder("david-goldstein")
    let EMail   = "dgoldstein@humnet.ucla.edu"
  end

  with JPHuelsenbeck
    let Title  = "J. P. Huelsenbeck"
    let First  = "John"
    let Middle = "P."
    let Last   = "Huelsenbeck"
    let Address = ["UC Berkeley", "3040 Valley Life Sciences Building #3140", "Berkeley, CA 94720-3140", "USA"
]
    let Website = Domain("https://vcresearch.berkeley.edu") Folder("faculty") Folder("john-huelsenbeck")
    let EMail   = "johnh@berkeley.edu"
  end

  with SHMcCreight
    let Title  = "S. H. McCreight"
    let First  = "Shawn"
    let Middle = "H."
    let Last   = "McCreight"
    let Address = ["3060 San Pasqual St.", "Pasadena, CA 91107", "USA"]
    let Website = Domain("https://nytril.com")
    let EMail   = "shawn.mccreight@gmail.com"
  end
end
//========================================================================

let WikipediaLink(name) = Domain("en.wikipedia.org") Folder("wiki") Folder(name)

with References
  with RevBayes
    let Author    = "Höhna, Landis, Heath, Boussau, Lartillot, Moore, Huelsenbeck, Ronquist"
    let Year      = 2016
    let Title     = "RevBayes: Bayesian phylogenetic inference using graphical models and an interactive
model-specification language"
    let Publisher = "Systematic Biology"
    let Pages     = "65:726-736"
    let Link      = Domain("http://www.revbayes.com")
  end
```

```
  with WordLists
    let Author    = null
    let Year      = 2019
    let Title     = "IPA Symbols Chart Complete"
    let Publisher = "InternationalPhoneticAlphabet.org"
    let Link      = Domain("http://www.internationalphoneticalphabet.org") Folder("ipa-charts") Folder("ipa-
symbols-chart-complete")
  end

  with IPAInformation
    let Author    = null
    let Year      = 2019
    let Title     = "IPA Symbols Chart Complete"
    let Publisher = "InternationalPhoneticAlphabet.org"
    let Link      = Domain("http://www.internationalphoneticalphabet.org") Folder("ipa-charts") Folder("ipa-
symbols-chart-complete")
  end

  with IPAWikipedia
    let Author    = null
    let Year      = 2019
    let Title     = "International Phonetic Alphabet"
    let Publisher = "Wikipedia"
    let Link      = WikipediaLink("International_Phonetic_Alphabet")
  end

  with ASJP
    let Author    = "Wichmann, Søren, Eric W. Holman, and Cecil H. Brown (eds.)"
    let Year      = 2018
    let Title     = "The ASJP Database (version 18)"
    let Publisher = "ASJP"
    let Link      = Domain("asjp.clld.org")
  end

  with XSAMPA
    let Author    = null
    let Year      = 2016
    let Title     = "Extended Speech Assessment Methods Phonetic Alphabet"
    let Publisher = "Wikipedia"
    let Link      = WikipediaLink("X-SAMPA")
  end

  with DistinctiveFeature
    let Author    = null
    let Year      = 2020
    let Title     = "Distinctive Feature"
    let Publisher = "Wikipedia"
    let Link      = WikipediaLink("Distinctive_feature")
  end

  with Lunter
    let Author    = "G. A. Lunter, I. Miklós, Y. S. Song, and J. Hein"
    let Year      = 2003
    let Title     = "An Efficient Algorithm for Statistical Multiple Alignment on Arbitrary Phylogenetic
Trees"
    let Publisher = "Journal Of Computational Biology"
  end
end
//============================================================================
```

WordForms.nytril

```
//============================================================================

using WordMeanings

with WordMeanings
  let I        = enum
  let You      = enum
  let We       = enum
  let This     = enum
  let That     = enum
```

```
let Who      = enum
let What     = enum
let Not      = enum
let All      = enum
let Many     = enum
let One      = enum
let Two      = enum
let Big      = enum
let Long     = enum
let Small    = enum
let Woman    = enum
let Man      = enum
let Person   = enum
let Fish     = enum
let Bird     = enum
let Dog      = enum
let Louse    = enum
let Tree     = enum
let Seed     = enum
let Leaf     = enum
let Root     = enum
let Bark     = enum
let Skin     = enum
let Flesh    = enum
let Blood    = enum
let Bone     = enum
let Grease   = enum
let Egg      = enum
let Horn     = enum
let Tail     = enum
let Feather  = enum
let Hair     = enum
let Head     = enum
let Ear      = enum
let Eye      = enum
let Nose     = enum
let Mouth    = enum
let Tooth    = enum
let Tongue   = enum
let Claw     = enum
let Foot     = enum
let Knee     = enum
let Hand     = enum
let Belly    = enum
let Neck     = enum
let Breast   = enum
let Heart    = enum
let Liver    = enum
let Drink    = enum
let Eat      = enum
let Bite     = enum
let See      = enum
let Hear     = enum
let Know     = enum
let Sleep    = enum
let Die      = enum
let Kill     = enum
let Swim     = enum
let Fly      = enum
let Walk     = enum
let Come     = enum
let Lie      = enum
let Sit      = enum
let Stand    = enum
let Give     = enum
let Say      = enum
let Sun      = enum
let Moon     = enum
let Star     = enum
let Water    = enum
let Rain     = enum
let Stone    = enum
let Sand     = enum
let Earth    = enum
let Cloud    = enum
```

```
  let Smoke     = enum
  let Fire      = enum
  let Ash       = enum
  let Burn      = enum
  let Path      = enum
  let Mountain  = enum
  let Red       = enum
  let Green     = enum
  let Yellow    = enum
  let White     = enum
  let Black     = enum
  let Night     = enum
  let Hot       = enum
  let Cold      = enum
  let Full      = enum
  let New       = enum
  let Good      = enum
  let Round     = enum
  let Dry       = enum
  let Name      = enum
end

let Def(meaning, sampa) = {
  Meaning: meaning,
  Sampa: sampa,
  Segments: IPA.SampaSet.FindTokens(sampa, ref IPA.NoSegment)
}
//======================================================================

let WordList.Catalan = [
  Def(I, "Zo"),
  Def(You, "tu"),
  Def(We, "nuzaltr3s"),
  Def(One, "un"),
  Def(Two, "dos"),
  Def(Person, "p3rson3"),
  Def(Fish, "peS"),
//  Def(Dog, "gos"),
  Def(Dog, "k3"),
  Def(Louse, "poL"),
  Def(Tree, "abr3"),
  Def(Leaf, "fuL3"),
  Def(Skin, "peL"),
  Def(Blood, "saN"),
  Def(Bone, "os"),
  Def(Horn, "korn"),
  Def(Horn, "ba53"),
  Def(Ear, "urEL3"),
  Def(Eye, "uL"),
  Def(Nose, "nas"),
  Def(Tooth, "den"),
  Def(Tongue, "LeNgw~3"),
  Def(Knee, "j3noL"),
  Def(Hand, "ma"),
  Def(Breast, "pit"),
  Def(Liver, "fej3"),
  Def(Drink, "bEur3"),
  Def(See, "bEur3"),
  Def(Hear, "s3nti"),
  Def(Die, "muri"),
  Def(Come, "b3ni"),
  Def(Sun, "sol"),
  Def(Star, "3streL3"),
  Def(Water, "aixw~3"),
  Def(Stone, "pe8r3"),
  Def(Fire, "fok"),
  Def(Path, "k3mi"),
  Def(Mountain, "mon"),
  Def(Night, "nit"),
  Def(Full, "plE"),
  Def(New, "nou"),
  Def(Name, "nom"),
]
//======================================================================
```

```
let WordList.French = [
  Def(I, "j3"),
  Def(You, "ti"),
//  Def(You, "vu"),
  Def(We, "nu"),
  Def(This, "s3si"),
  Def(That, "s3la"),
  Def(Who, "ki"),
  Def(What, "kwa"),
  Def(Not, "n3 pa"),
  Def(All, "tu"),
  Def(Many, "boku"),
  Def(One, "oe*"),
  Def(Two, "de"),
  Def(Big, "gra*"),
  Def(Long, "lo*"),
  Def(Small, "p3ti"),
  Def(Woman, "fam"),
  Def(Man, "om"),
  Def(Person, "om"),
  Def(Fish, "pw~aso*"),
  Def(Bird, "wazo"),
  Def(Dog, "Sia*"),
  Def(Louse, "pu"),
  Def(Tree, "arbr3"),
  Def(Seed, "gran"),
  Def(Leaf, "f3y"),
  Def(Root, "rasin"),
  Def(Bark, "ekors"),
  Def(Skin, "po"),
  Def(Flesh, "vy~a*d"),
  Def(Blood, "sa*"),
  Def(Bone, "os"),
  Def(Grease, "grais"),
  Def(Egg, "3f"),
  Def(Horn, "korn"),
  Def(Tail, "ke"),
  Def(Feather, "ply~m"),
  Def(Hair, "S3ve"),
  Def(Head, "t3t"),
  Def(Ear, "ore"),
  Def(Eye, "3y"),
  Def(Nose, "ne"),
  Def(Mouth, "buS"),
  Def(Tooth, "da*"),
  Def(Tongue, "la*g"),
  Def(Claw, "o*gl"),
  Def(Foot, "py~e"),
  Def(Knee, "j3nu"),
  Def(Hand, "ma*"),
  Def(Belly, "va*tr"),
  Def(Neck, "ku"),
  Def(Breast, "pw~atrin"),
  Def(Heart, "k3r"),
  Def(Liver, "fw~a"),
  Def(Drink, "bw~a"),
  Def(Eat, "ma*g"),
  Def(Bite, "mord"),
  Def(See, "vw~a"),
  Def(Hear, "o*ta*dr"),
  Def(Know, "savw~a"),
  Def(Sleep, "dormi"),
  Def(Die, "muri"),
  Def(Kill, "tue"),
  Def(Swim, "naje"),
  Def(Fly, "vw~ale"),
  Def(Walk, "marSe"),
  Def(Come, "v3ni"),
  Def(Lie, "seta*dr"),
  Def(Lie, "etra*da*dE"),
  Def(Sit, "sasw~a"),
  Def(Sit, "etrasi"),
  Def(Stand, "s3l3ve"),
  Def(Stand, "s3t3nird3vu"),
  Def(Give, "done"),
```

```
  Def(Say, "di"),
  Def(Sun, "sole"),
  Def(Moon, "len"),
  Def(Star, "etw~ol"),
  Def(Water, "o"),
  Def(Rain, "plui"),
  Def(Stone, "py~er"),
  Def(Sand, "sabl"),
  Def(Earth, "ter"),
  Def(Cloud, "nuaj"),
  Def(Smoke, "fEme"),
  Def(Fire, "fe"),
  Def(Ash, "sa*dr"),
  Def(Burn, "brule"),
  Def(Path, "rut"),
  Def(Mountain, "mo*taj"),
  Def(Red, "ruj"),
  Def(Green, "ver"),
  Def(Yellow, "jon"),
  Def(White, "bla*"),
  Def(Black, "nw~ar"),
  Def(Night, "nui"),
  Def(Hot, "So"),
  Def(Cold, "fr~wa"),
  Def(Full, "pl3*"),
  Def(New, "nuvo"),
  Def(Good, "bo*"),
  Def(Round, "ro*"),
  Def(Dry, "s3k"),
  Def(Name, "no*"),
]
//======================================================================

let WordList.Friulian = [
  Def(I, "yo"),
  Def(You, "tu"),
  Def(We, "nou"),
//  Def(We, "noaltris"),
  Def(One, "uN"),
  Def(Two, "doi"),
  Def(Person, "pErsoN"),
  Def(Fish, "pes"),
//  Def(Dog, "CaN"),
  Def(Dog, "ky~aN"),
  Def(Louse, "pEdoli"),
  Def(Tree, "arbul"),
  Def(Leaf, "fw~eE"),
  Def(Skin, "py~el"),
  Def(Blood, "saNk"),
  Def(Bone, "vw~es"),
  Def(Horn, "kw~ar"),
  Def(Ear, "oreli"),
  Def(Eye, "voli"),
  Def(Nose, "nas"),
  Def(Tooth, "dint"),
  Def(Tongue, "leNgE"),
  Def(Knee, "zEnoli"),
  Def(Knee, "jEnoli"),
  Def(Hand, "man"),
  Def(Breast, "pet"),
  Def(Liver, "fiat"),
  Def(Liver, "fy~at"),
  Def(Drink, "bevi"),
  Def(See, "viodi"),
  Def(See, "vy~odi"),
  Def(Hear, "sintei"),
  Def(Die, "murei"),
  Def(Come, "vi5ei"),
  Def(Sun, "soreli"),
  Def(Star, "stelE"),
  Def(Water, "agE"),
  Def(Stone, "py~erE"),
//  Def(Fire, "fouk"),
  Def(Fire, "fuk"),
  Def(Path, "stradE"),
```

```
  Def(Mountain, "mont"),
  Def(Mountain, "monta5E"),
  Def(Night, "5ot"),
  Def(Full, "plen"),
//  Def(New, "5ouf"),
  Def(New, "5uf"),
  Def(Name, "non"),
]
//======================================================================

let WordList.Italian = [
  Def(I, "io"),
  Def(You, "tu"),
  Def(We, "noi"),
  Def(One, "uno"),
  Def(Two, "due"),
  Def(Person, "persona"),
  Def(Fish, "peSe"),
  Def(Dog, "kane"),
  Def(Louse, "pidokky~o"),
  Def(Tree, "albero"),
  Def(Leaf, "foLa"),
  Def(Skin, "pElle"),
  Def(Blood, "saNgwe"),
  Def(Bone, "osso"),
  Def(Horn, "korno"),
  Def(Ear, "orekkyo"),
  Def(Eye, "okkyo"),
  Def(Nose, "naso"),
  Def(Tooth, "dante"),
  Def(Tongue, "liNgwa"),
  Def(Knee, "jinokkyo"),
  Def(Hand, "mano"),
  Def(Breast, "pEtto"),
  Def(Liver, "fegato"),
  Def(Drink, "bere"),
  Def(See, "ved"),
  Def(Hear, "ud"),
  Def(Die, "mor"),
  Def(Come, "vEn"),
  Def(Sun, "sole"),
  Def(Star, "stella"),
  Def(Water, "akwa"),
  Def(Stone, "pyEtra"),
  Def(Fire, "fwoko"),
  Def(Path, "sentyaro"),
  Def(Mountain, "monta5a"),
  Def(Night, "notte"),
  Def(Full, "pyEno"),
  Def(New, "nwovo"),
  Def(Name, "nome"),
]
//======================================================================

let WordList.Latin = [
  // David: Comments are placed with two forward slashes

  Def(I, "ego:"),
  Def(You, "tu:"),
  Def(We, "no:s"),
  Def(One, "u:nus"),
  Def(Two, "duo"),
  Def(Person, "perso:na"),
//  Def(Person, "homo", "homo:"),
  Def(Fish, "piskis"),
  Def(Dog, "kanis"),
  Def(Louse, "pedikulus"),
  Def(Tree, "arbor"),
  Def(Leaf, "foly~u*"),   //I don't understand the representation for Leaf
  Def(Skin, "kutis"),
  Def(Blood, "sang_Wis"),
  Def(Bone, "o:s"),
  Def(Horn, "kornu:"),
  Def(Ear, "auris"),
  Def(Eye, "okulus"),
```

```
  Def(Nose, "na:sus"),
  Def(Tooth, "de:ns"),
  Def(Tongue, "liNgw~E"),   //I don't know what E represents here
  Def(Knee, "genu:"),
  Def(Hand, "manus"),
  Def(Breast, "pektus"),
  Def(Breast, "mama"),   //The word for Breast is wrong---mamilla or pectus?
  Def(Liver, "jekur"),
  Def(Drink, "bibere"),
  Def(See, "wide:re"),
  Def(Hear, "audi:re"),
  Def(Die, "mori:"),
  Def(Come, "veni:re"),
  Def(Sun, "so:5"),
  Def(Star, "ste:la"),
  Def(Water, "ak_Wa"),
  Def(Stone, "lapis"),
  Def(Fire, "iNnis"),
  Def(Path, "wia"),
  Def(Mountain, "mo:ns"),
  Def(Night, "noks"),
  Def(Full, "ple:nus"),
  Def(New, "nowus"),
  Def(Name, "no:men"),
]
//=========================================================================

let WordList.Portuguese = [
  Def(I, "eu"),
  Def(You, "tu"),
  Def(We, "noS"),
  Def(One, "u*"),
  Def(Two, "doiS"),
  Def(Person, "pErzon"),
  Def(Fish, "paiS3"),
  Def(Dog, "ka*u*"),
  Def(Louse, "pioLu"),
  Def(Tree, "Ervur3"),
  Def(Leaf, "foLa"),
  Def(Skin, "pEl3"),
  Def(Blood, "sa*x3"),
  Def(Bone, "osu"),
  Def(Horn, "Sifr3"),
  Def(Ear, "oraLa"),
  Def(Eye, "oLu"),
  Def(Nose, "nariS"),
  Def(Tooth, "de*t3"),
  Def(Tongue, "li*gua"),
  Def(Knee, "ZuaLu"),
  Def(Hand, "ma*u"),
  Def(Breast, "saiuS"),
  Def(Liver, "fixa8u"),
  Def(Drink, "b3b"),
  Def(See, "ver"),
  Def(Hear, "ov"),
  Def(Die, "mur"),
  Def(Come, "vir"),
  Def(Sun, "sol"),
  Def(Star, "3Strela"),
  Def(Water, "Egw~a"),
  Def(Stone, "pEdra"),
  Def(Fire, "fogu"),
  Def(Path, "se*da"),
  Def(Mountain, "mo*ta5a"),
  Def(Night, "noyt3"),
  Def(Full, "Seyu"),
  Def(New, "novu"),
  Def(Name, "nom3"),
]
//=========================================================================

let WordList.Romanian = [
  Def(I, "ew"),
  Def(You, "tu"),
  Def(We, "noy"),
```

```
  Def(One, "unu"),
  Def(Two, "doy"),
  Def(Person, "om"),
  Def(Fish, "peSte"),
  Def(Dog, "kaine"),
  Def(Louse, "paduke"),
  Def(Tree, "arbore"),
  Def(Tree, "pom"),
  Def(Leaf, "frunz3"),
  Def(Skin, "pyele"),
  Def(Blood, "s3nje"),
  Def(Bone, "os"),
  Def(Horn, "korn"),
  Def(Ear, "ureke"),
  Def(Eye, "oky"),
  Def(Nose, "nas"),
  Def(Tooth, "dinte"),
  Def(Tongue, "limb3"),
  Def(Knee, "jenuNky"),
  Def(Hand, "m3n3"),
  Def(Breast, "s3n"),
  Def(Liver, "fikat"),
  Def(Drink, "bea"),
  Def(See, "vedea"),
  Def(Hear, "auzy"),
  Def(Die, "mury"),
//  Def(Die, "pieri"),
//  Def(Die, "raposa"),
  Def(Come, "veny"),
  Def(Sun, "soare"),
  Def(Star, "stea"),
//  Def(Star, "steaua"),
  Def(Water, "ap3"),
  Def(Stone, "pyatr3"),
  Def(Fire, "fok"),
  Def(Path, "cale"),
  Def(Mountain, "munte"),
  Def(Night, "noapte"),
  Def(Full, "plin"),
  Def(New, "now"),
  Def(Name, "nume"),

  /*
  Romanian 1
  Def(You, "tu"),
  Def(We, "noi"),
  Def(One, "unu"),
  Def(Two, "doi"),
  Def(Person, "persoan3"),
  Def(Fish, "peSte"),
  Def(Dog, "k3ne"),
  Def(Louse, "p3duke"),
  Def(Tree, "pom"),
  Def(Tree, "arbore"),
  Def(Leaf, "frunz3"),
  Def(Skin, "py~ele"),
  Def(Blood, "s3nje"),
  Def(Bone, "os"),
  Def(Horn, "korn"),
  Def(Ear, "ureke"),
  Def(Eye, "oky~"),
  Def(Nose, "nas"),
  Def(Tooth, "dinte"),
  Def(Tongue, "limb3"),
  Def(Knee, "jenuNky~"),
  Def(Hand, "m3n3"),
  Def(Breast, "py~ept"),
  Def(Breast, "s3n"),
  Def(Liver, "fikat"),
  Def(Drink, "bea"),
  Def(See, "vedea"),
  Def(Hear, "auzy~"),
  Def(Die, "mury~"),
  Def(Come, "veny~"),
  Def(Sun, "soare"),
```

```
  Def(Star, "stea"),
  Def(Water, "ap3"),
  Def(Stone, "py~atr3"),
  Def(Fire, "fok"),
  Def(Path, "k3rare"),
  Def(Mountain, "munte"),
  Def(Night, "noapte"),
  Def(Full, "plin"),
  Def(New, "nou"),
  Def(Name, "nume"),
*/

]
//=============================================================================

let WordList.Romansh = [
  Def(I, "yaw"),
  Def(You, "ti"),
  Def(We, "nus"),
  Def(One, "en"),
  Def(Two, "dus"),
  Def(Person, "k3rSTawn"),
  Def(Fish, "peS"),
  Def(Dog, "Tawn"),
  Def(Louse, "pluL"),
  Def(Tree, "plant3"),
  Def(Leaf, "feL"),
  Def(Skin, "pel"),
  Def(Blood, "saNk"),
  Def(Bone, "os"),
  Def(Horn, "korn3"),
  Def(Ear, "ureL3"),
  Def(Eye, "eL"),
  Def(Nose, "nas"),
  Def(Tooth, "dEnt"),
  Def(Tongue, "lyewNg3"),
  Def(Knee, "Z3neye"),
  Def(Hand, "mawn"),
  Def(Breast, "pET"),
  Def(Liver, "5irom"),
  Def(Drink, "bayv3r"),
  Def(See, "v3zayr"),
  Def(Hear, "udir"),
  Def(Die, "murir"),
  Def(Come, "v35ir"),
  Def(Sun, "suleL"),
  Def(Star, "Stayl3"),
  Def(Water, "aw3"),
  Def(Stone, "krap"),
  Def(Fire, "fyew"),
  Def(Path, "vi3"),
  Def(Mountain, "munto53"),
  Def(Night, "noT"),
  Def(Full, "playn"),
  Def(New, "nof"),
  Def(Name, "num"),
]
//=============================================================================

let WordList.Spanish = [
  Def(I, "yo"),
//  Def(You, "ustet"),
  Def(You, "tu"),
  Def(We, "nosotros"),
  Def(This, "este"),
  Def(That, "ese"),
  Def(That, "akely~a"),
  Def(Who, "kien"),
  Def(What, "ke"),
  Def(Not, "no"),
  Def(All, "todos"),
  Def(Many, "muCos"),
  Def(One, "uno"),
  Def(Two, "dos"),
  Def(Big, "grande"),
```

```
  Def(Long, "largo"),
  Def(Small, "peke5o"),
  Def(Small, "Ciko"),
  Def(Woman, "muher"),
  Def(Man, "ombre"),
  Def(Person, "persona"),
  Def(Fish, "peskado"),
  Def(Fish, "pes"),
  Def(Bird, "ave"),
  Def(Bird, "paharo"),
  Def(Dog, "pero"),
  Def(Louse, "pioho"),
  Def(Tree, "arbol"),
  Def(Tree, "palo"),
  Def(Seed, "semiya"),
  Def(Leaf, "oha"),
  Def(Root, "rais"),
  Def(Bark, "kortesa"),
  Def(Bark, "kaskara"),
  Def(Skin, "piel"),
  Def(Flesh, "karne"),
  Def(Blood, "sangre"),
  Def(Bone, "weso"),
  Def(Grease, "grasa"),
  Def(Egg, "wevo"),
  Def(Horn, "kw~erno"),
  Def(Tail, "kola"),
  Def(Tail, "rabo"),
  Def(Feather, "pluma"),
  Def(Hair, "pelo"),
  Def(Hair, "cabeyo"),
  Def(Head, "kabesa"),
  Def(Ear, "oreha"),
  Def(Eye, "oho"),
  Def(Nose, "naris"),
  Def(Mouth, "boka"),
  Def(Tooth, "diente"),
  Def(Tongue, "lengw~a"),
  Def(Claw, "gara"),
  Def(Foot, "pie"),
  Def(Foot, "pata"),
  Def(Knee, "rodiya"),
  Def(Hand, "mano"),
  Def(Belly, "bariga"),
  Def(Neck, "kw~eyo"),
  Def(Neck, "peskw~eso"),
  Def(Breast, "peCo"),
  Def(Breast, "seno"),
  Def(Heart, "korason"),
  Def(Liver, "igado"),
  Def(Drink, "bebe"),
//  Def(Drink, "toma"),
  Def(Eat, "kome"),
  Def(Bite, "morde"),
  Def(See, "ve"),
  Def(Hear, "oir"),
  Def(Know, "sabe"),
  Def(Know, "konose"),
  Def(Sleep, "dormi"),
  Def(Die, "mori"),
  Def(Kill, "mata"),
  Def(Swim, "nada"),
  Def(Fly, "vola"),
  Def(Walk, "anda"),
  Def(Walk, "kamina"),
  Def(Come, "veni"),
  Def(Lie, "akosta"),
  Def(Lie, "eCa"),
  Def(Sit, "senta"),
  Def(Stand, "esta de pie"),
  Def(Give, "da"),
  Def(Say, "desi"),
  Def(Sun, "sol"),
  Def(Moon, "luna"),
  Def(Star, "estreya"),
```

```
  Def(Water, "agw~a"),
  Def(Rain, "yuvia"),
  Def(Stone, "piedra"),
  Def(Sand, "arena"),
  Def(Earth, "tiera"),
  Def(Cloud, "nube"),
  Def(Smoke, "humo"),
  Def(Fire, "fuego"),
  Def(Ash, "senisa"),
  Def(Burn, "kema"),
  Def(Burn, "arde"),
  Def(Path, "senda"),
  Def(Mountain, "sero"),
  Def(Mountain, "monta5a"),
  Def(Red, "roho"),
  Def(Red, "kolorado"),
  Def(Green, "verde"),
  Def(Yellow, "amariyo"),
  Def(White, "blanko"),
  Def(Black, "negro"),
  Def(Night, "noCe"),
  Def(Hot, "kaliente"),
  Def(Cold, "frio"),
  Def(Full, "yeno"),
  Def(New, "nuevo"),
  Def(Good, "bw~eno"),
  Def(Round, "redondo"),
  Def(Dry, "seko"),
  Def(Name, "nombre"),
]
//============================================================================

let WordList.Walloon = [
  Def(I, "Ce"),
  Def(You, "te"),
  Def(We, "nos"),
  Def(One, "E*"),
  Def(Person, "o*m"),
  Def(Dog, "Ce*"),
  Def(Skin, "pow"),
  Def(Ear, "oreye"),
  Def(Eye, "ui"),
  Def(Drink, "bwEr"),
  Def(Hear, "Sute"),
  Def(Die, "murrir"),
  Def(Come, "vnir"),
  Def(Star, "twEl"),
  Def(Water, "Ew3"),
  Def(Fire, "fE"),
  Def(Path, "vwa*y"),
  Def(Full, "pli*"),
  Def(New, "novEl"),
]
//============================================================================
```

Tables.nytril

```
using Format, Units, IPA, IPA.FeatureSet
//============================================================================

with Results
  let FindMeaning(word, data) = word.Meaning == data;

  let GetGeneText(meaning, ref language) begin
    var words    = language.Words.FindSlice(FindMeaning, meaning, 1);
    var segments = words.Length == 1 ? words[0].Segments : [ref IPA.NoSegment];
    return {
      Language: ref language,
      Meaning: meaning,
      Word: words[0],
      Segments: segments,
      Count: segments.Length
    }
  end
```

```
  let GetMeaningRecord(meaning) begin
    var cells = GetGeneText(meaning, each UsedLanguages);
    return {
       Cells: cells,
       MaxLength: Math.Max((each cells).Count)
    };
  end

  let AddBlank(list, index) begin
    list.AddReference(IPA.GapSegment);
  end

  let AddWord(list, langindex, meaning) begin
    var ma   = WordMeaningArray[EachIndex];
    var cell = ma.Cells[langindex];
    var pad  = ma.MaxLength - cell.Count;
    list.AddReference(IPA.LeftSegment);
    list.AddReference(each cell.Segments);
    if (pad > 0)
      AddBlank(list, each 1..pad);
    end
    list.AddReference(IPA.RightSegment);
  end

  let GetSegments(langindex) begin
    var list = Type.List(150);
    AddWord(list, langindex, each UsedMeanings);
    return list;
  end

  let SegmentToCharacter(ref segment) begin
    if (segment.Punctuation)
      return {
        Character: segment.Ipa,
        Segment: ref segment
      }
    end

    var f = UniqueSegments.FindIndex(DisplayCharacters.SameSegment, {Segment: ref segment});
    if (f.Length == 1)
      return {
        Character: Nexus.CharacterList[f[0]],
        Segment: ref segment
      }
    else
      return {
        Character: '?',
        Segment: ref IPA.NoSegment
      }
    end
  end

  let GetTaxaArray(ref lang) begin
    var segments = GetSegments(EachIndex);
    return {
      Name: lang.SymbolName,
      Segments: segments,
      Characters: SegmentToCharacter(each segments)
    }
  end

  let LangHasWords(ref language) = language.Words != null
  let UsedLanguages              = FindSlice(Languages, LangHasWords)
  let UsedMeanings               = CompleteMeanings.Find
  let WordMeaningArray           = GetMeaningRecord(each UsedMeanings);
  let UniqueSegments             = DisplayCharacters.FindUniqueSegments(UsedLanguages)
  let TaxaArray                  = GetTaxaArray(each UsedLanguages)
  let LanguageTreeFile           = Nexus.TreeFile(Info.LanguageTree)
  let CharacterFile              = Nexus.CharacterFile(TaxaArray)

//  let UsedSegments = UniqueSegments
  let UsedSegments = IPA.Segments

end
```

```
//=========================================================================
// Consonants
//=========================================================================

with MatchingConsonants
  let PulmonicTable = {
    Title: Lang.PConsonants,
    ColWidth: 45 pts,
    Exclude: Affricate Ejective,
    Include: Pulmonic,
    All: NoFeature,
    Manners: [Nasal, Stop, Sibilant Fricative, Fricative, Approximant, TapFlap, Trill, Lateral Fricative,
Lateral Approximant, Lateral TapFlap],
    RowMask: Nasal Stop Sibilant Fricative Approximant TapFlap Trill Lateral Velarized,
    Notes: Lang.SymbolPairVoiced
  }

  let NonPulmonicTable = {
    Title: Lang.NPConsonants,
    ColWidth: 80 pts,
    Exclude: Vowel Pulmonic Central,
    Include: Ejective Click Implosive,
    All: NoFeature,
    Manners: [Ejective Stop, Ejective Fricative, Ejective Lateral Fricative, Click Tenuis, Click Nasal, Click
Tenuis Lateral, Implosive],
    RowMask: Lateral,
    Notes: Lang.SymbolPairVoiced
  }

  let PulmonicAffricatesTable = {
    Title: Lang.PulmonicAffricates,
    ColWidth: 50 pts,
    Exclude: Vowel,
    Include: Affricate Sibilant Lateral,
    All: Pulmonic Affricate,
    RowLabels: [Lang.Sibilant, Lang.NonSibilant, Lang.Lateral],
    Manners: [Sibilant, NoFeature, Lateral],
    RowMask: Pulmonic Affricate Sibilant Fricative Lateral,
  }

  let EjectiveAffricatesTable = {
    Title: Lang.EjectiveAffricates,
    ColWidth: 50 pts,
    Exclude: Vowel Click Implosive Pulmonic,
    Include: Ejective Affricate Central Lateral,
    All: Ejective Affricate,
    Manners: [Central, Lateral],
    RowMask: Pulmonic Approximant Central Lateral,
  }

  let ShowTables begin
    var segments = Type.Dictionary(256);
    return Block {
      MatchingOptions.ShowTable(segments, each [PulmonicTable, NonPulmonicTable]),
      PageBreak,
      MatchingVowels.ShowTable(segments),
      MatchingOptions.ShowTable(segments, each [PulmonicAffricatesTable, EjectiveAffricatesTable]),
      MatchingOther.ShowTable(segments),
    }
  end
end
//=========================================================================

with DisplayWords
  let WordRow(word) = Row {
    Style.RowBar(EachIndex),
    Cell {
      Style.SansSerif,
      Style.TitleBackground,
      word.Meaning
    },
    Cell {
      ShowSampa(word.Sampa)
    },
    Cell {
```

```
      Span {
        ShowSegment(each word.Segments)
      },
    },
    Cell {
      Span {
        Separator: "-",
        SegmentSound(each word.Segments)
      }
    }
  }

  let HeaderCell(d) = Cell {
    Style.SansSerif,
    Style.TitleBackground,
    EdgeB: 1 pts,
    Padding: 2 pts,
    d
  }

  let ShowTable(language) = Block {
    Table {
      Columns: [0.8 inches, 1 inches, 1 inches, Metrics.Content.Width - 2.5 inches],
      Style.TitleBar(language.Name, 4),
      Row {
        HeaderCell(Lang.Meaning),
        HeaderCell(Lang.Sampa),
        HeaderCell(Lang.IPA),
        HeaderCell(Lang.Sounds),
      },
      WordRow(each language.Words)
    },
    Style.TableNotes
  }
end
//============================================================================

with AllWords
  let AddCell(meaning, ref language) = Cell {
    ShowIPA(each IPA.FindWordsWithMeaning(language, meaning))
  }

  let AddRow(ref language, meanings) = Row {
    Cell {
      Style.SansSerif,
      TextColor: Colors.DarkGray,
      language.Name
    },
    AddCell(each meanings, language)
  }

  let MeaningTable(languages, meanings) = Block {
    Table {
      Edge: 0.5,
      Columns: [70 pts] + [54 pts] * meanings.Length,
      Row {
        Style.TitleBackground,
        Style.SansSerif,
        Empty,
        each meanings,
      },
      AddRow(each languages, meanings)
    },
    Paragraph,
  }

  let ShowTable(languages, meanings) = Block {
    TextHeight: 12 pts,
    MeaningTable(languages, each (meanings / 8))
  }
end
//============================================================================

with DisplayCharacters
  let WordMatch(meaning, def) = meaning == def.Meaning
```

```
  let CompleteWord(def)       = Results.UsedMeanings.Contains(WordMatch, def)
  let GetWordList(language)   = language.Words.FindSlice(CompleteWord)


  let CollectWord(set, word) begin
    set.AddReference(each word.Segments);
  end

  let CollectLanguage(set, language) begin
    CollectWord(set, each GetWordList(language));
  end

  let FindUniqueSegments(languagelist) begin
    var set = Type.Dictionary(256);
    CollectLanguage(set, each languagelist);
    return set.ValueList;
  end

  let SameSegment(ref segment, data) = segment == data.Segment

  let CollectLangWord(set, ref segment, word) begin
    if (word.Segments.Contains(SameSegment, {Segment: ref segment}))
      set.AddElement(word.Sampa, word);
    end
  end

  let CollectLanguageWords(set, ref segment, language) begin
    CollectLangWord(set, segment, each GetWordList(language));
  end

  let WordsWithSegment(ref segment) begin
    var set = Type.Dictionary(256);
    CollectLanguageWords(set, segment, each Results.UsedLanguages);
    return set.ValueList;
  end

  let CharacterRow(ref segment) = Row {
    Cell {
      HAlign: HAligns.Center,
      Style.TitleBackground,
      Nexus.CharacterList[EachIndex]
    },
    Cell {
      HAlign: HAligns.Center,
      ShowSegment(segment),
    },
    Cell {
      Style.IPAFamily,
      Span {
        Separator: Lang.Separator,
        ShowIPA(each WordsWithSegment(segment))
      }
    }
  }

  let ShowTable = Block {
    Table {
      Style.TableEdge,
      Columns: [0.5 inches, 0.7 inches, 6 inches],
      Row {
        Style.HeaderCell("Char.", HAligns.Center),
        Style.HeaderCell(Lang.Segment, HAligns.Center),
        Style.HeaderCell("Words containing this segment")
      },
      CharacterRow(each Results.UniqueSegments),
    },
    Style.TableNotes
  }
end
//=============================================================================
// Consonants
//=============================================================================

with MatchingOptions
  let CheckFlags(sflags, rflags, options) = sflags.NotFeature(options.Exclude) and sflags.HasFeature(options.
```

```
Include) and sflags.HasMask(options.RowMask | rflags, rflags)

  let AnyManners(flags, data)             = CheckFlags(data.Features, flags | data.Options.All, data.Options)

  let MatchRow(segment, data)             = segment.Place == data.Place and CheckFlags(segment.Features, data.
Features, data.Options)

  let MatchInclude(segment, data)         = segment.Place == data.Place and segment.Features.NotFeature(data.
Options.Exclude) and
                                            segment.Features.HasFeature(data.Options.Include) and
                                            data.Options.Manners.Contains(AnyManners, {Features: segment.
Features, Options: data.Options})

  let MatchPlace(place, options)          = AllSegments.Contains(MatchInclude, {Place: place, Options: options
})
  let GetPlaces(options)                  = Places.FindSlice(MatchPlace, options)


  let SegmentText(segments, ref segment) begin
    if (not segment.Impossible)
      segments.AddReference(segment)
    end
    return ShowSegment(segment);
  end

  let SegmentBox(segments, ref segment, color) = Canvas {
    HAlign: HAligns.Center,
    Size: Metrics.BoxSize,
    if (segment.Ipa)
      TextHeight: Metrics.BoxSize.Height - 4 pts,
      TextColor: color,
      SegmentText(segments, segment)
    else
      Style.ImpossibleBackground
    end
  }

  let ShowBox(segments, ref segment, color) = SegmentBox(segments, segment, color) {
    if (segment.Features.HasFeature(Voiced))
      X: Metrics.BoxSize.Width
    end
  }

  let MatchError(matches) = matches.Length > 2 or (matches.Length == 2 and matches[0].Features.HasFeature(
Voiced) == matches[1].Features.HasFeature(Voiced))

  let SegmentBlock(segments, matches) = Cell {
    Style.SegmentBottom,
    if (matches.Length > 0)
      if (matches.Length == 1 and matches[0].Impossible)
        Style.ImpossibleBackground
      else
        TextHeight: 1 pts,
        Span {
          Canvas {
            Size: Metrics.CellSize,
            ShowBox(segments, each matches, MatchError(matches) ? Colors.Red : Colors.Black)
          }
        }
      end
    end
  }

  let AddCell(segments, options, place, flags) = SegmentBlock(segments, AllSegments.FindSlice(MatchRow, {Place
: place, Features: flags, Options: options}))

  let AddRow(segments, options, places, flags) begin
    var allflags = flags | options.All;
    return Row {
      Cell {
        VAlign: VAligns.Center,
        EdgeR: 0.5 pts,
        Style.TitleBackground,
        PaddingLR: 2 pts,
        TextHeight: 7 pts,
```

```
        if (options.RowLabels)
          options.RowLabels[EachIndex],
        else
          flags,
        end,
      },
      AddCell(segments, options, each places, allflags)
    }
  end

  let PlaceHeader(place) = Style.HeaderCell(place.Name, HAligns.Center)

  let ShowTable(segments, options) begin
    var places = GetPlaces(options);

    return Block {
      Table {
        Style.TableEdge,
        Columns: [options.ColWidth {EdgeR: 0.5 pts}] +
                 [Metrics.CellSize.Width {HAlign: HAligns.Center, EdgeR: 0.25 pts}] * places.Length,

        Style.TitleBar(options.Title, places.Length+1),
        Row {
          TextHeight: 6 pts,
          Style.HeaderCell(Bold Lang.Manner),
          PlaceHeader(each places)
        },
        AddRow(segments, options, places, each options.Manners),
      },
      Style.TableNotes {
        Lang.ImpossibleShaded,
        Space,
        options.Notes
      },
    }
  end
end
//=============================================================================
// Vowels
//=============================================================================

with MatchingVowels
  let AddBlock(set, matches) = Cell {
    Style.SegmentBottom,
    HAlign: HAligns.Center,
    TextHeight: Metrics.BoxSize.Height,
    if (matches.Length == 2)
      Span {
        MatchingOptions.SegmentText(set, matches[0]),
        " • " {TextColor: Colors.LightGray},
        MatchingOptions.SegmentText(set, matches[1]),
      }
    else
      if (matches.Length == 1)
        MatchingOptions.SegmentText(set, matches[0]),
      end
    end
  }

  let MatchVowelAny(ref segment, data)      = segment.Features.HasMask(Vowel LongVowel Nasal, data.Feature |
Vowel) and segment.Open == data.Open
  let MatchVowelPair(ref segment, data)     = segment.Backness == data.Backness and MatchVowelAny(segment,
data)
  let FindAnyOpen(set, feature, open)       = Results.UsedSegments.Contains(MatchVowelAny, {Open: open,
Feature: feature})
  let AddCell(set, feature, open, backness) = AddBlock(set, Results.UsedSegments.FindSlice(MatchVowelPair, {
Open: open, Feature: feature, Backness: backness}))

  let AddRow(set, feature, open) begin
    if (FindAnyOpen(set, feature, open))
      return Row {
        Cell {
          VAlign: VAligns.Center,
          Style.TitleBackground,
```

```
          open.Name
        },
        AddCell(set, feature, open, each Backnesses)
      };
    else
      return null
    end
  end

  let ShowVowelTable(set, feature, title) = Table {
    HAlign: HAligns.Center,
    Style.TableEdge,
    Columns: [(1 inch){EdgeR: 0.5 pts}] + [Metrics.BoxSize.Width*3 {EdgeR: 0.25 pts}] * Backnesses.Length,
    Style.TitleBar(title, Backnesses.Length+1),
    Row {
      TextHeight: 10 pts,
      Style.HeaderCell(Empty),
      Style.HeaderCell(each Backnesses, HAligns.Center)
    },
    AddRow(set, feature, each Opens)
  }

  let ShowTable(set) = Block {
    ShowVowelTable(set, NoFeature, Lang.Vowels),
    Style.TableNotes {
      Lang.SymbolPairRounded,
    },
    ShowVowelTable(set, LongVowel, Lang.LongVowels),
    Paragraph,
  }
end
//=========================================================================
// Diacritic markers
//=========================================================================

with MatchingDiacritics
  let ColumnDiv = 3

  let AddCell(ref segment) = Cell {
    Paragraph {
      LeftIndent: 0.5 inches,
      FirstIndent: -0.5 inches,
      LocationMark: segment.FullSymbolName,
      ShowSegment(segment) {
        TextHeight: 18 pts,
        Tab,
      },
      segment.Description
    }
  }

  let AddRow(segments) = Row {
    AddCell(each segments)
  }

  let ShowTable = Block {
    Table {
      Style.TableEdge,
      Columns: [Metrics.Content.Width / ColumnDiv] * ColumnDiv,
      Style.TitleBar(Lang.Diacritics, ColumnDiv),
      AddRow(each DiacriticModifiers / ColumnDiv),
    },
    Style.TableNotes
  }
end
//=========================================================================
// Segments not in other lists
//=========================================================================

with MatchingOther
  let MatchOther(ref segment, set) = not set.ContainsReference(segment)

  let ColumnDiv = 2

  let AddCell(ref segment) = {
```

```
    Cell {
      VAlign: VAligns.Center,
      HAlign: HAligns.Center,
      TextHeight: 20 pts,
      LocationMark: segment.FullSymbolName,
      ShowSegment(segment),
    },
    Cell {
      VAlign: VAligns.Center,
      SegmentName(segment)
    }
  }

  let AddRow(segments) = Row {
    AddCell(each segments)
  }

  let ShowTable(set) = Block {
    Table {
      Style.TableEdge,
      Columns: [Metrics.BoxSize.Width, Metrics.Content.Width * 0.5 - Metrics.BoxSize.Width] * ColumnDiv,
      Style.TitleBar(Lang.OtherSegments, ColumnDiv*2),
      AddRow(each (FindSlice(Results.UsedSegments, MatchOther, set) / ColumnDiv)),
    },
    Style.TableNotes
  }
end
//=========================================================================
// Find the list of meanings for which there is a word in every language
//=========================================================================

with CompleteMeanings
  let ContainsMeaning(word, data)    = word.Meaning == data.Meaning
  let WithoutMeaning(language, data) = not language.Words.Contains(ContainsMeaning, data)

  let CollectMeanings(set, ref meaning) begin
    if (not Results.UsedLanguages.Contains(WithoutMeaning, {Meaning: ref meaning}))
      set.AddReference(meaning)
    end
  end

  let Find begin
    var set = Type.Dictionary(128);
    CollectMeanings(set, each WordMeanings);
    return set.ValueList;
  end

  let AddRow(ref meaning) = Row {
    Cell {
      HAlign: HAligns.Center,
      Style.TitleBackground,
      EachIndex+1
    },
    Cell {
      meaning.Name
    }
  }

  let ShowTable = Block {
    Table {
      Style.TableEdge,
      Columns: [0.5 inches, 4 inches],
      Style.TitleBar(Lang.Meanings, 3),
      Row {
        Style.HeaderCell(Empty),
        Style.HeaderCell(Lang.Meaning)
      },
      AddRow(each Results.UsedMeanings),
    },
    Style.TableNotes
  }
end
//=========================================================================
// Segment Tree
//=========================================================================
```

```
with SegmentTree
  let AddNode(name) = Node {
    Bevel: 20%,
    Curvature: 20%,
    Label: name
  }

  let AddSegment(set, ref segment) begin
    set.AddReference(segment);
    return ShowSegment(segment)
  end

  let AddSegments(set, name, func, data) begin
    var segments = Results.UsedSegments.FindSlice(func, data);
    if (segments.Length > 0)
      return Node {
        Label: Frame {
          Width: 3.5 inches,
          Paragraph {
            LeftIndent: 1 inches,
            FirstIndent: -1 inches,
            Span {
              TextColor: Colors.DarkGray,
              name,
              ":\t",
            },
            Span {
              TextHeight: 14 pts,
              Separator: Space,
              AddSegment(set, each segments),
            }
          }
        }
      }
    else
      return null
    end
  end

  let MatchVowel(ref segment, data) = segment.Features.HasFeature(Vowel) and segment.Backness == data.Backness
  let AddVowels(set, backness)      = AddSegments(set, backness.Name, MatchVowel, {Backness: backness})

  let MatchFlags(ref segment, data) = not data.Set.ContainsReference(segment) and segment.Features.HasMask(
data.All, data.Features) and segment.Features.HasMask(data.Other, data.Other)
  let AddFlag(set, all, flags, f)   = AddSegments(set, f.Name, MatchFlags, {Set: set, All: all, Features:
flags, Other: f})

  let AddConsonants(set, name, all, flags) = AddNode(name) {
    AddFlag(set, all, flags, each [Tenuis, Click, Nasal, Ejective, Fricative, Sibilant, Lateral, Stop,
Approximant, TapFlap, Trill, NoFeature]),
  }

  let AddVoicedPairs(set, name, all, flags) = AddNode(name) {
    AddConsonants(set, Lang.Voiced, all Voiced, flags Voiced),
    AddConsonants(set, Lang.Voiceless, all Voiced, flags),
  }

  let ShowTree begin
    var set1 = Type.Dictionary(256);
    var set2 = Type.Dictionary(256);
    var all  = Vowel Pulmonic Affricate;

    return Block {
      Tree {
        Width: 7 inches,
        LabelGap: 3 pts,
        Node {
          AddNode(Lang.Vowels) {
            AddVowels(set1, each Backnesses)
          },
          AddNode(Lang.Affricates) {
            AddVoicedPairs(set1, Lang.NonPulmonic, all, Affricate),
            AddVoicedPairs(set1, Lang.Pulmonic, all, Pulmonic Affricate),
          },
```

```
          AddNode(Lang.Consonants) {
            AddVoicedPairs(set1, Lang.Pulmonic, all, Pulmonic),
            AddVoicedPairs(set1, Lang.NonPulmonic, all, NoFeature),
          },
          AddSegments(set2, Lang.OtherSegments, MatchingOptions.MatchOther, set1)
        }
      }
    }
  end
end
//=========================================================================
// SAMPA table
//=========================================================================

with SAMPAConversion
  let ShowSampaLine(segment) = Span {
    segment.Sampa
  }

  let AddRow(ref segment) = Row {
    Cell {
      ShowSampa(segment.Sampa)
    },
    Cell {
      ShowSegment(segment)
    },
    Cell {
      segment.SymbolName
    }
  }

  let AlphaOrder(x, y) begin
    var cl = Math.Compare(x.Sampa.Length, y.Sampa.Length);
    if (cl == 0)
      cl = -Math.Compare(x.Sampa, y.Sampa)
    end
    return cl;
  end

  let SortedSampa = IPA.SampaSet.Sort(false, AlphaOrder)

  let ShowTable = Block {
    Table {
      Style.TableEdge,
      Columns: [0.75 inches, 0.75 inches, 4 inches],
      Row {
        Style.HeaderCell(Lang.Sampa),
        Style.HeaderCell(Lang.IPA),
        Style.HeaderCell(Lang.Segment)
      },
      AddRow(each SortedSampa),
    },
    Style.TableNotes
  }
end
//=========================================================================
```

Main.nytril

```
using Format, Units, Math, IO

include "English"
include "Library"
include "Languages"
include "LanguageTree"
include "IPA"
include "Style"
include "References"
include "WordForms"
include "Tables"
//=========================================================================

let Main.Run = [
//  Write(RevBayes.SourceFile, Info.RevSourcePath),
```

```
  Write(Results.LanguageTreeFile, Info.LanguageTreePath),
  Write(Results.CharacterFile, Info.CharacterPath),
  Write(WhitePaper, Info.PaperPath Extensions.PDF),
//  Write(WhitePaper, Info.PaperPath Extensions.Word),
  IO.OpenDocument(Info.PaperPath Extensions.PDF),
]
//=========================================================================

with Info
  let MainFolder       = Folders.Source
  let OutputFolder     = MainFolder Folder("Output")
  let PaperPath        = OutputFolder FileName("Paper")
  let LanguageTreePath = OutputFolder FileName("LanguageTree") Extensions.Nexus
  let RevSourcePath    = OutputFolder FileName("Analysis") Extensions.RevBayes

  let CharacterPath    = OutputFolder FileName("Characters") Extensions.Nexus
  let Journal          = "Transactions of the Beysian Society"
  let Publisher        = "The Baysian Society"
  let Title            = "Simulated Feature Evolution using the TKF91 Model"
  let LanguageTree     = LanguageBranches.Romance

  let AuthorList = Span {
    Separator: Lang.Separator,
    LastSeparator: " {0} "(Lang.And),
    Style.Author(each Authors)
  }
end
//=========================================================================

let Watch = WhitePaper

let WhitePaper = Style.WhitePaper {
  Title: Info.Title,
  Author: Info.AuthorList,
//  Description: "Test Description",
//  Comment: "Test Comment",
//  Subject: "Test Subject",
//  Keywords: "Test Keywords",

  Style.PageSection {
    Header: Style.NormalHeader(Info.Journal) {
      Distance: 0.125 inches,
      Even: Style.NormalHeader(Info.Journal),
      First: Block {
        Paragraph {
          ParAlignment: ParAlignments.Center,
          Logo,
        },
      }
    },
    Block {
      ParAlignment: ParAlignments.Center,
      Style.Title(Info.Title),
      Paragraph {
        Info.AuthorList
      },
    },
    Style.ShowAbstract(Abstract),
    Style.ShowContent(each Content),
    Style.ShowAuthors(Authors),
    Style.ShowAppendixTable,
    Style.ShowReferences(References),
  },
  Style.ShowAppendix(each Appendix)
}
//=========================================================================

let AddAppendix(title, content) = {Title: title, Content: content}

let Appendix = [
  AddAppendix(Lang.LanguagePhylogeny, Style.ShowLanguageTree),
//  AddAppendix("Meanings with words in every language", CompleteMeanings.ShowTable),
  AddAppendix("Words in each language by meaning", AllWords.ShowTable(Results.UsedLanguages, Results.
UsedMeanings)),
  AddAppendix("Feature Change", IPA.DriftSection),
```

```
  AddAppendix("Character file", Results.CharacterFile Style.MonoFamily),
  AddAppendix("Segments in the target word list", DisplayCharacters.ShowTable),
  AddAppendix("Segment Groups", MatchingConsonants.ShowTables),
  AddAppendix(Lang.Diacritics, MatchingDiacritics.ShowTable),
  AddAppendix("Euler Feature Diagram", IPA.FeatureChart),
  AddAppendix(Lang.IPAFullName, IPA.SegmentTable),
  AddAppendix("Feature Tree", SegmentTree.ShowTree),
  AddAppendix("SAMPAConversion", SAMPAConversion.ShowTable),
  AddAppendix("Word Lists by Language", DisplayWords.ShowTable(each Results.UsedLanguages)),
  AddAppendix("Language Tree File", Results.LanguageTreeFile Style.MonoFamily),
  AddAppendix(Lang.NytrilSourceCode, Style.SourceFile(each System.SourceList)),
]
//=========================================================================

with Abstract
  let Title = Lang.Abstract
  let Body  = Block {
    Paragraph {
      "It all started in a little town called Madrid..."
    }
  }
end
//=========================================================================

with Content.Introduction
  let Title = Lang.Introduction
  let Body  = Block {
    Paragraph {
      "In this paper, we attempt to do the impossible!"
    }
  }
end
//=========================================================================

with Content.Methods
  let Title = Lang.Methods
  let Body  = Block {
    Paragraph {
      "We used any and all means necessary."
    }
  }
end
//=========================================================================

with Content.Conclusion
  let Title = Lang.Conclusion
  let Body  = Block {
    Paragraph {
      "Vene Vidi Vici"
    }
  }
end
//=========================================================================
```