



PetaMem Language Server - Manual

Support

PetaMem GmbH

Flurstr. 78

D-90765 Fürth

support@petamem.com

As of: October 12, 2025

- Great care has been taken to ensure the accuracy of the features and techniques presented in this publication. However, PetaMem accepts no responsibility, and offers no warranty whether expressed or implied, for the accuracy of this publication.
- The information in this document is subject to change without notice. PetaMem makes no warranty of any kind in regard to the contents of this document, including, but not limited to, any implied warranties of merchantability quality or fitness for any particular purpose. PetaMem shall not be liable for errors contained in it or for incidental or consequential damages concerning the furnishing, performance or use of this document.
- All trademarks mentioned in this document are the property of their respective owners.

PetaMem Copyright Notice

Copyright © 2002-2016 PetaMem, s.r.o. All rights reserved.

This work is intellectual property of PetaMem, s.r.o. It may not, in whole or in part, be reproduced, published, distributed, performed, displayed, transmitted or incorporated in compilations or derivative works without the express written permission of PetaMem, s.r.o..

Contents

I Administrator	7
1 Before You Begin	9
1.1 Quick Start Guide	9
1.2 System Requirements Overview	10
1.2.1 Hardware Requirements	11
1.2.2 Software Requirements	12
1.2.3 License Management	13
2 Deployment and Configuration	17
2.1 Install and Deinstall	18
2.1.1 Mounting The DVD	18
2.1.2 Install From Directory	19
2.1.3 Deinstalling of a PM Project	19
2.2 Upgrade and Downgrade	20
2.2.1 By Installation	20
2.2.2 By Explicit Switch	21
2.2.3 By Implicit Switch	22
2.3 PMLS Configuration	22
2.3.1 PMLS File System Layout	22
2.3.2 Environment Variables	24
2.3.3 List of Environment Variables	24
2.3.4 UTF-8 Support	26
2.3.5 Server Configuration File	27
Admin Manual Appendices	29

A Licenses	29
A.1 PetaMem Software License	29
A.2 Artistic License	33
A.3 GNU General Public License	36
B Error Messages	41
II User	43
3 First Steps	45
3.1 Server Startup & Shutdown	45
3.1.1 Manual Startup	45
3.1.2 Boot Sequence Startup	46
3.2 Connecting and Disconnecting from the Server	46
3.2.1 Batch Connect	47
3.2.2 Interactive Connect	47
3.3 Command Syntax	48
3.3.1 Positional Parameters	48
3.3.2 Named Parameters	48
4 System Functionality	51
4.1 System Control	51
4.1.1 Ending a Session	51
4.1.2 System Information	51
4.1.3 Formats of client-server communication	52
4.1.4 System Configuration	52
4.2 User Management	53
4.2.1 Creating and Removing Users	53
4.2.2 Users: Information and Disconnecting	53
4.3 User Permissions	54
4.3.1 Setting and Unsetting Permissions	54
4.3.2 User Permissions Information	54
4.4 File System	54

4.4.1	Uploading and Downloading Files	54
4.4.2	File Manipulation	55
4.4.3	Globber	55
4.5	Data Conversion	55
4.5.1	References	56
5	NLP Functionality	57
5.1	Basic Text Functionality	57
5.1.1	Text Metrics	57
5.1.2	Get Unique Tokens	58
5.2	Diacritics Operations	58
5.2.1	References	59
5.3	Language Identification	59
5.3.1	Dict	59
5.3.2	NGram	59
5.3.3	NVect	59
5.3.4	Smart	59
5.3.5	References	59
5.4	Text Summarization	59
5.5	Text Categorization	60
5.6	Spell Check	60
5.7	Morphosyntactic Analysis	60
5.8	Procedural Word Processor	61
5.8.1	References	62
6	NLU Functionality	63
6.1	Lexical Operations	63
6.1.1	Nomenclature	63
6.1.2	Load and Save Lexica	63
6.1.3	Create and Delete Lexica	63
6.1.4	Set Operations on Lexica	64
6.1.5	Adding Entries	64
6.1.6	Finding Entries	65

6.1.7	Inspecting Entries	66
6.1.8	Modifying Entries	67
6.2	Discourse Engine	67
6.2.1	Analysis	68
6.2.2	Generation	69
6.2.3	Memory	70
6.2.4	Internal State Model	70
6.2.5	Usage Tutorial	70
6.2.6	Create and Extend Ontologies	71
6.2.7	Emotional Modelling	77
6.2.8	References	78
6.3	Machine Translation	79
6.4	Information Extraction (IE)	79
7	Advanced Usage	81
7.1	Batch Operation	81
7.2	Remote Server Configuration	81
8	PMSE	83
8.1	About the PetaMem Scripting Environment	83
8.1.1	Conceptual Overview	84
8.1.2	Paths and Directory Structure	84
8.1.3	PMSE Toolset Overview	84
8.2	P_bsd: Basic Statistical Data	88
8.2.1	Reference	88
8.2.2	Examples	90
8.3	P_cct: Corpus Conversion Tool	91
8.3.1	Reference	91
8.3.2	Examples	92
8.4	P_cop: Co-occurrence Processor	93
8.4.1	Reference	93
8.4.2	Examples	94
8.5	P_csp: Comprehensive Statistics Processor	94

8.5.1	Reference	95
8.5.2	Examples	97
8.5.3	Q&A	100
8.6	P_daf: Data Acquisition Framework	100
8.6.1	Reference	100
8.6.2	How to Write an INI File	101
8.6.3	Private Data and Personal INI	102
8.6.4	Extended INI File	102
8.6.5	Hooks	103
8.6.6	Examples	104
8.7	P_dmf: Data Mining Framework	104
8.7.1	Reference	104
8.7.2	File Structure	105
8.7.3	Input Formats	109
8.7.4	Dependencies	109
8.7.5	Input Texts, Encoding	110
8.7.6	Wikimedia Processing	110
8.7.7	Examples	112
8.8	P_dmp: Distance Measures Processor	112
8.8.1	Reference	112
8.8.2	Examples	114
8.8.3	Q&A	114
8.8.4	Distance Functions Characteristics	115
8.9	P_dvf: Data Visualization Framework	120
8.9.1	Reference	120
8.9.2	Examples	123
8.10	P_gnp: Generic N-grams Processor	124
8.10.1	Reference	124
8.10.2	Examples	129
8.10.3	Q&A	130
8.11	P_help: PMSE Helper	132
8.11.1	Reference	132

8.11.2	Examples	132
8.12	P_ici: Intelligent Command Iterator	133
8.12.1	Reference	135
8.12.2	Examples	137
8.13	P_rer: Regular Expression Replacer	138
8.13.1	Reference	139
8.13.2	Examples	140
8.14	P_trt: Text Repair Tool	141
8.14.1	Reference	141
8.14.2	Examples	142
8.15	P_vls: Variable Length Splitter	143
8.15.1	Reference	143
8.15.2	Examples	145
8.16	PMSE: Tutorial	145
8.16.1	Learning by Example	145
8.16.2	Corpora	145
8.16.3	P_csp Interactive	146
8.16.4	P_gnp Interactive	148
8.16.5	Categorization of EMA Texts	152
8.17	PMSE: Cookbook	153
8.17.1	Recipes for PMSE	153
8.17.2	PMSE Crash Course	153
8.17.3	Sentence Segmentation	155
8.17.4	Sub Word N-grams Extraction	158
8.17.5	Probability of Neighbors	161
8.17.6	Co-occurrences	162
8.17.7	Text Categorization	165
8.17.8	PMSE Visualization	167
8.17.9	Visualization of Contingency Tables	170
8.17.10	N-grams Histogram Visualization	172
8.17.11	Macros	176

User Manual Appendices

181

C	Command Reference	181
C.1	?	181
C.2	bot	181
C.3	bot_add	182
C.4	bot_del	183
C.5	bots_list	183
C.6	cat	183
C.7	cfg-client	183
C.8	cfg-get	184
C.9	cfg-reload	185
C.10	cfg-set	185
C.11	compat	185
C.12	convert	186
C.13	cp	186
C.14	dict	186
C.15	doc	187
C.16	e-cpy	187
C.17	e-del	188
C.18	e-describe	188
C.19	e-get_arg	188
C.20	e-get_common	188
C.21	e-get_thull	189
C.22	e-mov	189
C.23	e-new	189
C.24	e-show	189
C.25	e-translate	190
C.26	encode	190
C.27	h	190
C.28	help	190
C.29	help-syntax	191
C.30	l-cpy	192
C.31	l-del	192

C.32 l-info	192
C.33 l-list	193
C.34 l-load	193
C.35 l-mov	193
C.36 l-new	194
C.37 l-reversify	194
C.38 l-save	194
C.39 lc-chart	195
C.40 lc-cpy	195
C.41 lc-del	195
C.42 lc-info	195
C.43 lc-list	196
C.44 lc-mov	196
C.45 lc-new	196
C.46 lcmpr	196
C.47 lcover	196
C.48 li	197
C.49 li-llm	198
C.50 ls	198
C.51 mt	198
C.52 mv	199
C.53 nla-list	199
C.54 nla-morph	199
C.55 nla-number	200
C.56 nla-tag	200
C.57 nlg-list	201
C.58 nlg-morph	201
C.59 nlg-number	201
C.60 nlp-m_test	202
C.61 perm_add	202
C.62 perm_del	202
C.63 perms_list	203

C.64	purge	203
C.65	riter	203
C.66	rm	204
C.67	say	204
C.68	shutdown	205
C.69	sys-info	205
C.70	txt-info	206
C.71	user_add	206
C.72	user_del	206
C.73	users_list	206
C.74	v-cchk	206
C.75	v-get_clique	207
C.76	v-info	207
C.77	v-query	208
C.78	who	208
C.79	whoami	208
C.80	debug	209
C.81	get	209
C.82	put	209
C.83	quit	209
C.84	user	210
D	KR Formalism	211
D.1	Class Logic...	211
D.1.1	...And Beyond	211
D.2	Original Nomenclature...	211
D.2.1	Terms and Formulae	213
D.3	...Our Annotations	213
D.3.1	Truth Value and Probability	214
E	Discourse Macro Reference	217
E.1	Examples	218

III Developer	219
9 Collaborative Development	221
9.1 Subversion	221
9.1.1 SVN Access	221
9.1.2 Development Cycle Tutorial	222
9.2 Priority Of Base Ontologies	223
10 Plugins for Text Categorization	225
10.1 Ngrams filtering	225
10.2 Files filtering	226
10.3 Specifying vector	226
10.4 Functions for getting corpus vector	227
10.5 Functions for getting cluster vector	227
10.6 Functions for computing distance	227
Developer Manual Appendices	229
F API References	229
F.1 Semantic Lexicon	229
F.2 Discourse Engine ChatBot	241
F.3 APIs for PMLIB:: Namespace	243
F.3.1 PMLIB::Association	243
F.3.2 PMLIB::Association::Mi	245
F.3.3 PMLIB::Association::T	246
F.3.4 PMLIB::Config	247
F.3.5 PMLIB::Conv	247
F.3.6 PMLIB::Conv::7z2_	249
F.3.7 PMLIB::Conv::7z2_::p7zip	249
F.3.8 PMLIB::Conv::ace2_	249
F.3.9 PMLIB::Conv::ace2_::ACE_Compression_Software	250
F.3.10 PMLIB::Conv::ar2_	250
F.3.11 PMLIB::Conv::ar2_::GNU	250
F.3.12 PMLIB::Conv::arc2_	251

F.3.13	PMLIB::Conv::arc2_::ARC	251
F.3.14	PMLIB::Conv::arj2_	251
F.3.15	PMLIB::Conv::arj2_::ARJ_Software	252
F.3.16	PMLIB::Conv::azw32txt	252
F.3.17	PMLIB::Conv::azw32txt::calibre	252
F.3.18	PMLIB::Conv::bz22_	253
F.3.19	PMLIB::Conv::bz22_::Seward	253
F.3.20	PMLIB::Conv::cab2_	253
F.3.21	PMLIB::Conv::cab2_::cabextract	254
F.3.22	PMLIB::Conv::chm2pdf	254
F.3.23	PMLIB::Conv::chm2pdf::chm2pdf	254
F.3.24	PMLIB::Conv::cpio2_	255
F.3.25	PMLIB::Conv::cpio2_::GNU	255
F.3.26	PMLIB::Conv::cpio2_::bsdcpio	255
F.3.27	PMLIB::Conv::djvu2txt	256
F.3.28	PMLIB::Conv::djvu2txt::DjVuLibre	256
F.3.29	PMLIB::Conv::doc2txt	256
F.3.30	PMLIB::Conv::doc2txt::antiword	257
F.3.31	PMLIB::Conv::docx2txt	257
F.3.32	PMLIB::Conv::docx2txt::Kumar	257
F.3.33	PMLIB::Conv::dvi2pdf	258
F.3.34	PMLIB::Conv::dvi2pdf::Artifex	258
F.3.35	PMLIB::Conv::epub2txt	258
F.3.36	PMLIB::Conv::epub2txt::calibre	259
F.3.37	PMLIB::Conv::fb22txt	259
F.3.38	PMLIB::Conv::fb22txt::calibre	259
F.3.39	PMLIB::Conv::fodt2txt	260
F.3.40	PMLIB::Conv::fodt2txt::unoconv	260
F.3.41	PMLIB::Conv::gif2png	260
F.3.42	PMLIB::Conv::gif2png::ImageMagick	261
F.3.43	PMLIB::Conv::gz2_	261
F.3.44	PMLIB::Conv::gz2_::GNU	261

E.3.45	PMLIB::Conv::html2txt	262
E.3.46	PMLIB::Conv::html2txt::mech_dump	262
E.3.47	PMLIB::Conv::htmlz2txt	262
E.3.48	PMLIB::Conv::htmlz2txt::calibre	263
E.3.49	PMLIB::Conv::jpg2txt	263
E.3.50	PMLIB::Conv::jpg2txt::tesseract	263
E.3.51	PMLIB::Conv::lha2_	264
E.3.52	PMLIB::Conv::lha2_::Howard	264
E.3.53	PMLIB::Conv::lha2_::Okamoto	264
E.3.54	PMLIB::Conv::lit2txt	265
E.3.55	PMLIB::Conv::lit2txt::calibre	265
E.3.56	PMLIB::Conv::lrf2txt	265
E.3.57	PMLIB::Conv::lrf2txt::calibre	266
E.3.58	PMLIB::Conv::lrz2_	266
E.3.59	PMLIB::Conv::lrz2_::lrzip	266
E.3.60	PMLIB::Conv::lzma2_	267
E.3.61	PMLIB::Conv::lzma2_::XZ_Utils	267
E.3.62	PMLIB::Conv::lzx2_	267
E.3.63	PMLIB::Conv::lzx2_::unlzx	268
E.3.64	PMLIB::Conv::mobi2txt	268
E.3.65	PMLIB::Conv::mobi2txt::calibre	268
E.3.66	PMLIB::Conv::odt2txt	269
E.3.67	PMLIB::Conv::odt2txt::Stosberg	269
E.3.68	PMLIB::Conv::odt2txt::unoconv	269
E.3.69	PMLIB::Conv::ott2txt	270
E.3.70	PMLIB::Conv::ott2txt::unoconv	270
E.3.71	PMLIB::Conv::pbz22_	270
E.3.72	PMLIB::Conv::pbz22_::Gilchrist	271
E.3.73	PMLIB::Conv::pdb2txt	271
E.3.74	PMLIB::Conv::pdb2txt::txt2pdbdoc	271
E.3.75	PMLIB::Conv::pdf2txt	272
E.3.76	PMLIB::Conv::pdf2txt::Poppler	272

F.3.77	PMLIB::Conv::pet2tgz	272
F.3.78	PMLIB::Conv::pet2tgz::Puppy	273
F.3.79	PMLIB::Conv::png2txt	273
F.3.80	PMLIB::Conv::png2txt::tesseract	273
F.3.81	PMLIB::Conv::ps2pdf	274
F.3.82	PMLIB::Conv::ps2pdf::gs	274
F.3.83	PMLIB::Conv::rar2_	274
F.3.84	PMLIB::Conv::rar2_::Roshal	275
F.3.85	PMLIB::Conv::rpm2cpio	275
F.3.86	PMLIB::Conv::rpm2cpio::Troan	275
F.3.87	PMLIB::Conv::rst2pdf	276
F.3.88	PMLIB::Conv::rst2pdf::rst2pdf	276
F.3.89	PMLIB::Conv::rtf2txt	276
F.3.90	PMLIB::Conv::rtf2txt::GNU	277
F.3.91	PMLIB::Conv::rz2_	277
F.3.92	PMLIB::Conv::rz2_::Tridgell	277
F.3.93	PMLIB::Conv::sea2sit	278
F.3.94	PMLIB::Conv::sea2sit::unsea	278
F.3.95	PMLIB::Conv::shar2_	278
F.3.96	PMLIB::Conv::shar2_::sh	279
F.3.97	PMLIB::Conv::sit2_	279
F.3.98	PMLIB::Conv::sit2_::unstuff	279
F.3.99	PMLIB::Conv::snb2txt	280
F.3.100	PMLIB::Conv::snb2txt::calibre	280
F.3.101	PMLIB::Conv::stw2txt	280
F.3.102	PMLIB::Conv::stw2txt::unoconv	281
F.3.103	PMLIB::Conv::sxw2txt	281
F.3.104	PMLIB::Conv::sxw2txt::unoconv	281
F.3.105	PMLIB::Conv::tar2_	282
F.3.106	PMLIB::Conv::tar2_::GNU	282
F.3.107	PMLIB::Conv::tar2_::bsdtar	282
F.3.108	PMLIB::Conv::tcr2txt	283

F.3.109PMLIB::Conv::tcr2txt::calibre	283
F.3.110PMLIB::Conv::tex2pdf	283
F.3.111PMLIB::Conv::tex2pdf::pdflatex	284
F.3.112PMLIB::Conv::texi2pdf	284
F.3.113PMLIB::Conv::texi2pdf::Essex	284
F.3.114PMLIB::Conv::tiff2txt	285
F.3.115PMLIB::Conv::tiff2txt::tesseract	285
F.3.116PMLIB::Conv::troff2txt	285
F.3.117PMLIB::Conv::troff2txt::GNU	286
F.3.118PMLIB::Conv::txtz2txt	286
F.3.119PMLIB::Conv::txtz2txt::calibre	286
F.3.120PMLIB::Conv::uot2txt	287
F.3.121PMLIB::Conv::uot2txt::unoconv	287
F.3.122PMLIB::Conv::wpd2txt	287
F.3.123PMLIB::Conv::wpd2txt::libwpd	288
F.3.124PMLIB::Conv::xar2_	288
F.3.125PMLIB::Conv::xar2_::xar	288
F.3.126PMLIB::Conv::xz2_	289
F.3.127PMLIB::Conv::xz2_::XZ_Utils	289
F.3.128PMLIB::Conv::zip2_	289
F.3.129PMLIB::Conv::zip2_::Spieler	290
F.3.130PMLIB::Conv::zoo2_	290
F.3.131PMLIB::Conv::zoo2_::Dhesi	290
F.3.132PMLIB::Email	291
F.3.133PMLIB::Encoding	291
F.3.134PMLIB::Encoding::List	292
F.3.135PMLIB::Error	292
F.3.136PMLIB::Facade	292
F.3.137PMLIB::Hash	293
F.3.138PMLIB::Hash::Tied	297
F.3.139PMLIB::Help	297
F.3.140PMLIB::ISO	298

F.3.141PMLIB::ISO::11940	299
F.3.142PMLIB::ISO::11941	299
F.3.143PMLIB::ISO::15919	300
F.3.144PMLIB::ISO::15924	300
F.3.145PMLIB::ISO::233	301
F.3.146PMLIB::ISO::259	301
F.3.147PMLIB::ISO::3166	301
F.3.148PMLIB::ISO::3602	303
F.3.149PMLIB::ISO::4217	303
F.3.150PMLIB::ISO::639	304
F.3.151PMLIB::ISO::7098	306
F.3.152PMLIB::ISO::843	307
F.3.153PMLIB::ISO::9	307
F.3.154PMLIB::ISO::9984	307
F.3.155PMLIB::ISO::9985	307
F.3.156PMLIB::IdentificationModeling	307
F.3.157PMLIB::IdentificationModeling::Ngram	308
F.3.158PMLIB::IdentificationModeling::Nvect	308
F.3.159PMLIB::IdentificationModeling::Tfreq	309
F.3.160PMLIB::List	309
F.3.161PMLIB::Logic::Belief	310
F.3.162PMLIB::Loop	312
F.3.163PMLIB::Math::Combinatorics	313
F.3.164PMLIB::Math::Combinatorics::Bell	314
F.3.165PMLIB::Math::Interval	314
F.3.166PMLIB::Math::Iterator	316
F.3.167PMLIB::Math::Matrix	316
F.3.168PMLIB::Math::SetTheory	318
F.3.169PMLIB::Math::Vector	320
F.3.170PMLIB::Metric	320
F.3.171PMLIB::Metric::Block	321
F.3.172PMLIB::Metric::Canberra	321

E.3.173PMLIB::Metric::Chebyshev	322
E.3.174PMLIB::Metric::Correlation	322
E.3.175PMLIB::Metric::Cosine	322
E.3.176PMLIB::Metric::Czekanowski	323
E.3.177PMLIB::Metric::DamerauLevenshtein	323
E.3.178PMLIB::Metric::Dice	323
E.3.179PMLIB::Metric::Discrete	323
E.3.180PMLIB::Metric::Euclid	324
E.3.181PMLIB::Metric::Hamming	324
E.3.182PMLIB::Metric::Jaccard	324
E.3.183PMLIB::Metric::Jaro	324
E.3.184PMLIB::Metric::JaroWinkler	324
E.3.185PMLIB::Metric::Kulezinski	325
E.3.186PMLIB::Metric::Levenshtein	325
E.3.187PMLIB::Metric::Mahalanobis	325
E.3.188PMLIB::Metric::Manhattan	325
E.3.189PMLIB::Metric::Minkowski	326
E.3.190PMLIB::Metric::NeedlemanWunsch	326
E.3.191PMLIB::Metric::Overlap	327
E.3.192PMLIB::Metric::Renkonen	327
E.3.193PMLIB::Metric::RusselRao	327
E.3.194PMLIB::Metric::SmithWaterman	327
E.3.195PMLIB::Metric::SokalSneath	328
E.3.196PMLIB::Metric::Tanimoto	328
E.3.197PMLIB::Metric::Tversky	328
E.3.198PMLIB::Metric::Typo	329
E.3.199PMLIB::Metric::Yule	329
E.3.200PMLIB::PMLIB	329
E.3.201PMLIB::Queue	330
E.3.202PMLIB::Regex	332
E.3.203PMLIB::Scalar	333
E.3.204PMLIB::Script	334

F.3.205PMLIB::Serialize	334
F.3.206PMLIB::Stack	335
F.3.207PMLIB::Stochastic::P	336
F.3.208PMLIB::Stochastic::Sample	337
F.3.209PMLIB::System	338
F.3.210PMLIB::System::HW	338
F.3.211PMLIB::System::HW::Keyboard	339
F.3.212PMLIB::System::OS	340
F.3.213PMLIB::Tag	341
F.3.214PMLIB::Tag::CLAWS5	342
F.3.215PMLIB::Tag::Multext	343
F.3.216PMLIB::Tag::Penn	343
F.3.217PMLIB::Tag::Pmts	344
F.3.218PMLIB::Test	345
F.3.219PMLIB::Text::Filter	350
F.3.220PMLIB::Time	351
F.3.221PMLIB::Timer	351
F.3.222PMLIB::Tree::Nary	351
F.3.223PMLIB::Types	358
F.3.224PMLIB::UpdateCheck	358
F.3.225PMLIB::UpdateCheck::ISO_15924	358
F.3.226PMLIB::UpdateCheck::ISO_3166	359
F.3.227PMLIB::UpdateCheck::ISO_4217	360
F.3.228PMLIB::UpdateCheck::ISO_639_3	361
F.3.229PMLIB::UpdateCheck::PMLIB	362
F.3.230PMLIB::Wiki::Helper	363
F.3.231PMLIB::Wiki::Parser	363
F.3.232PMLIB::f2f	363
F.3.233PMLIB::f2h	364
F.3.234PMLIB::f2l	365
F.3.235PMLIB::f2s	366
F.3.236PMLIB::f2x	366

F.3.237PMLIB::h2f	367
F.3.238PMLIB::h2h	367
F.3.239PMLIB::h2l	369
F.3.240PMLIB::h2s	370
F.3.241PMLIB::h2x	371
F.3.242PMLIB::l2f	371
F.3.243PMLIB::l2h	371
F.3.244PMLIB::l2l	373
F.3.245PMLIB::l2s	380
F.3.246PMLIB::l2x	385
F.3.247PMLIB::s2f	386
F.3.248PMLIB::s2h	386
F.3.249PMLIB::s2l	386
F.3.250PMLIB::s2s	388
F.3.251PMLIB::s2x	394
F.3.252PMLIB::x2f	395
F.3.253PMLIB::x2h	397
F.3.254PMLIB::x2l	398
F.3.255PMLIB::x2s	399
F.3.256PMLIB::x2x	399
F.4 APIs for PMSE:: Namespace	401
F.4.1 PMSE	401
F.4.2 PMSE::Categorization	407
F.4.3 PMSE::Categorization::FilterFiles	412
F.4.4 PMSE::Categorization::FilterFiles::Pareto	412
F.4.5 PMSE::Categorization::FilterFiles::Raw	412
F.4.6 PMSE::Categorization::FilterNgrams	412
F.4.7 PMSE::Categorization::FilterNgrams::Pareto	412
F.4.8 PMSE::Categorization::Vector	413
F.4.9 PMSE::Categorization::Vector::Frequent	413
F.4.10 PMSE::Corpus	414
F.4.11 PMSE::Corpus::BNC	414

F.4.12	PMSE::Corpus::CNK	415
F.4.13	PMSE::Corpus::OANC	416
F.4.14	PMSE::Corpus::PDT	416
F.4.15	PMSE::Corpus::PM	418
F.4.16	PMSE::Corpus::PM::Sentence	419
F.4.17	PMSE::Corpus::Penn	420
F.4.18	PMSE::Corpus::WikiCorpus	420
F.4.19	PMSE::DM	421
F.4.20	PMSE::DM::Aspell	421
F.4.21	PMSE::Library	422
F.4.22	PMSE::Visualize	422
F.4.23	PMSE::Visualize::BinaryTree	423
F.4.24	PMSE::Visualize::Contingency	425
F.4.25	PMSE::Visualize::Cooccurrence	426
F.4.26	PMSE::Visualize::Distance	426
F.4.27	PMSE::Visualize::Dot	427
F.4.28	PMSE::Visualize::FileStat	428
F.4.29	PMSE::Visualize::Histogram	428
F.4.30	PMSE::Visualize::Neighbors	430
E.5	APIs for PMSE Tools APIs	430
G	Regular Expressions Reference	431
G.1	Basics	431
G.2	Metacharacters	432
G.3	Escape Sequences	432
G.4	Quantifiers	433
G.5	Examples	433
G.5.1	Catching common Typos	434
H	ISO639 Reference	435
H.1	Sort by ISO639-1	435
H.2	Sort by ISO639-3	436
H.3	Sort by ISO639-Name	437

List of Figures

1	1 server, N clients configuration of PMLS: an arbitrary number of clients may connect to one PMLS	1
2	M servers, N clients configuration of PMLS: an arbitrary number of clients may connect to several PMLS that do load balancing and client redirection. . .	2
8.1	PMSE top level overview	85
8.2	P_bsd schematic overview	89
8.3	P_cop schematic overview	93
8.4	P_csp schematic overview	95
8.5	Overview of the statistical processes	98
8.6	P_daf schematic overview	101
8.7	P_daf processes overview	105
8.8	P_dmf schematic overview	106
8.9	P_dmf directory structure	107
8.10	P_dmp schematic overview	113
8.11	P_dvf schematic overview	121
8.12	P_gnp - overview of the processes	125
8.13	P_ici schematic overview	133
8.14	P_rer schematic overview	139
8.15	P_trt schematic overview	141
8.16	P_vls schematic overview	144
8.17	Schematic overview of subgrams extraction	159
8.18	The Canterbury Tales, and Other Poems by Geoffrey Chaucer - grammatical co-occurrences for 'say' (filtered input)	164
8.19	Example of clusters visualized by GraphViz	170
8.20	The Canterbury Tales, and Other Poems by Geoffrey Chaucer - three most frequent bigrams	173

8.21 The Canterbury Tales, and Other Poems by Geoffrey Chauce: most frequent
bigrams 175

List of Tables

1.2	minimum and recommended hardware requirements on 32bit and 64bit architectures per PMLS instance .	11
1.4	support status of various versions of the Perl interpreter.	12
2.1	Environment variables defined in <code>pm_bash_env_paths</code> file	25
2.2	Environment variables defined in <code>pm_bash_env_common</code> file	25
2.3	Variables used for running tests in PM projects	26
8.1	<code>.dmf.info</code> example	111
E.1	supported macros for Discourse Engine Ontologies.	218
G.1	Metacharacters in regular expressions.	432
G.2	Escape sequences in regular expressions.	433
G.3	Quantifiers in regular expressions.	433

About PMLS

The PetaMem Language Server (PMLS) is a comprehensive software suite for NLP (natural language processing) and NLU (natural language understanding). As such, it offers a complete functionality in the areas of NLA (natural language analysis), such as tokenizing, tagging, morphosyntactic analysis¹ and semantic inference as well as NLG (natural language generation), such as content determination, discourse planning, Sentence aggregation and other tasks.

The Software was designed with several goals in mind: **scalable architecture**, **strict multilinguality**, **rich functionality**, **hybrid linguistic approach**. We will discuss these goals in the following sections to share some of the philosophy that is behind this product.

Scalable Architecture

Being scalable means, that PMLS will allow for single-user operation on a personal PC or Workstation, but it also can run a multi-server/cluster based multi-user operation to serve requests from thousands of concurrent users.

From a system administrator's point of view, most of the time, PMLS is a regular client-server architecture as seen in figure 1. An arbitrary number of clients may connect to one PMLS. These clients have to adhere to the PMLS protocol, but may be otherwise simple command line clients, web frontends (cgi-scripts), mail-interfaces, standalone applications, scripts for batch text processing etc.

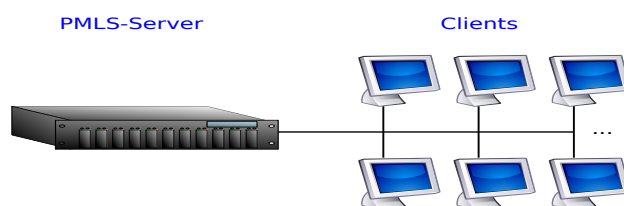


Figure 1: 1 server, N clients configuration of PMLS: an arbitrary number of clients may connect to one PMLS

¹commonly referred to as parsing

All incoming processing requests from the clients are being serialized in the PMLS and processed in the order they arrived. When workload gets too heavy for a single-server, a distributed client-server architecture as seen in figure 2 can be established.

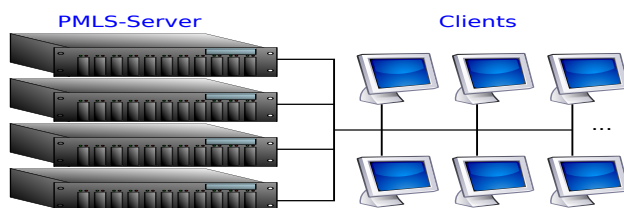


Figure 2: M servers, N clients configuration of PMLS: an arbitrary number of clients may connect to several PMLS that do load balancing and client redirection.

This will involve start-up of several PMLS instances either on some powerful multi-CPU machines and/or a cluster of computers. A client can connect to an arbitrary PMLS server in such a “server cloud” and will get redirected properly to a server that is expected to serve him within the shortest possible time.

Strict Multilinguality

PMLS uses Unicode² to represent characters of the textual data it processes. This means it can handle *“all the characters for all the writing systems of the world, modern and ancient.”* (from the Unicode FAQ)

The native data format for texts processed by PMLS is UTF-8 encoded text.

Furthermore, there is no “hard-coded algorithm” that would be specific to just one given language in the core system. Where such algorithms became necessary, they are implemented as plugins and scheduled from a dispatching framework in the PMLS core.

All language-specific parameters, such as the vocabulary, morphosyntactic and statistical information are covered in the lexica and may be loaded, unloaded and modified at run-time. You can extend PMLS with a new language without having to shutdown the PMLS core.

As of this writing, more than 500 languages are commercially available and several other in preparation. For a detailed list, status and release date please contact our sales department and request the General FactSheet document.

Rich Functionality

Many systems prune their functionality to achieve quickly a very specialised goal. While this approach can lead to results quite fast, these results achieved also often tend to be restricted in terms of functionality and genericity.

²see also <http://www.unicode.org/standard/WhatIsUnicode.html>

The PMLS is the world's most comprehensive and generic NLP/NLU software suite, because its design goals were functional completeness and language independence.

With PMLS you have access to various functional elements such as **language identification**, **text categorization**, **text summarization**, **machine translation**, **diacritics processor**, **morphosyntactic analysis**³, **discourse engine**, **lexical operations** and many more.

Being an integrated NLP/NLU software suite, this package also will help you to achieve your goals faster, because it takes away the integration effort you'd have with several isolated applications. On the other hand, PMLS provides concise API to be integrated into other frameworks or business processes or to integrate them. For a detailed overview of the PMLS NLP/NLU functionality please consult chapter 5, pg. 57. You can browse the API documentation in the developer part of this manual at appendix F, pg. 229.

Hybrid Linguistic Approach

PMLS makes use of both statistical and rule-based algorithms. Morphosyntactic rules are crafted manually, while information about word frequencies, collocations, Ngrams etc. is of statistical nature. Various subsystems use all of these information for analysis and generation.

Take for example the language identification (5.3, pg. 59) functionality. The ngram-based identification method is statistical in nature, the dictionary-based identification method is a rule framework and the top frequency identification method is itself hybrid in nature.

³which includes a stemmer, tagger and lemmatizer

About This Manual

This is the authoritative manual for PMLS, revision **620** and later.

There are three parts:

Part I: Administrator (pp. 9ff.) This part covers all aspects of PMLS administration, such as installing, upgrading, deinstalling, user management, configuration, performance tuning etc.

Part II: User (pp. 45ff.) This part covers PMLS from the user perspective and provides information how to use the system NLP/NLU functionality in everyday operation.

Part III: Developer (pp. 221ff.) This part covers advanced topics and contains reference material that is of interest to software developers who need to extend the PMLS functionality, integrate PMLS into some specific environment or simply need reference material for advanced operation.

This manual should not be used with older versions of PMLS due to the many functional and other differences between the covered version and previous versions. If you are using an earlier release of the PMLS software, please contact PetaMem support to obtain an appropriate copy of this manual for your version. If you did upgrade your PMLS installation, you can find an annotated version of this manual with visual markup of the differences to a given previous version in the [doc/manual/diffs](#) directory.

Also, this manual is available in electronic form in the [doc/manual](#) directory. Both in one page per A4 view format as well as an A5 booklet format - which is the source for the printed manual.

The PMLS software is under constant and fast-paced development, as is this manual. If you are not subscribed to the PetaMem newsletter, please check regularly your client account pages at <http://client.petamem.com/<yoursection>> for any updates.

Should you find errors, inconsistencies or unclear formulations, don't hesitate to contact support@petamem.com.

Conventions Used in This Manual

- **This** is how we refer to files, directories, variables and URLs: “The file [pmlsconf.ini](#) contains the configuration for the PMLS server process and can be found in the [bin](#) directory.”

- This is how we refer to commands and their parameters in text: “Enter `./client_std -help` to get information about the PMLS command line client usage.”
- THIS is how we refer to PetaMem products and tools: “The PetaMem Language Server is called PMLS”
- This is how we emphasize an important action or any other memorable fact.
- *This* is reserved for citations and similar *foreign imported text*

In order to distinguish between input to the operating system shell (such as `bash`⁴ or `tcsh`⁵) or the PMLS input shell, we will refer to those as follows:

```
bash> type a shell command here
```

```
pmls> type a pmls statement here
```

When you enter a command or statement shown in an example, do not type the prompt shown in the example.

Commands for setting shell variables are shown using Bourne shell syntax. For example, the sequence to set the CC environment variable and run the configure command looks like this in Bourne shell syntax:

```
bash> CC=gcc ./configure
```

If you are using `csch` or `tcsh`, you have to issue commands somewhat differently:

```
bash> setenv CC gcc
```

```
bash> ./configure
```

An ellipsis (...) indicates the omission of a section of a statement, typically to provide a shorter version of more complex syntax, alternatively it can also mean an omission in data output, to put the focus on the relevant data.

⁴Bourne again shell - see <http://www.gnu.org/software/bash> - used in most Linux distributions

⁵Enhanced Berkley UNIX C shell - see <http://www.tcsh.org>

Part I

Administrator

Chapter 1

Before You Begin

The beginning is the chiefest part of any work.
Plato

Great effort has been put into making the deployment and upgrading of your PMLS software as easy as possible. The easiest way is described in the following Quick Start Guide, which is worth a try if your system fulfills the requirements and the default factory settings are ok for your application scenario.

If you have specific requirements, skip the quickstart guide and read the details in the subsequent chapters.

1.1 Quick Start Guide

The PetaMem Language Server (PMLS) software you obtained with this manual is compatible to any contemporary UNIX system which meets the requirements mentioned in section 1.2, pg. 10 of this manual. If you are confident, that your system fulfills these requirements, you can try the quick start way of installing PMLS described here:

1. Put the DVD into your DVD-ROM, start the program `./pm_install` and wait until it finishes.
2. Go to the PMLS `bin` directory (normally this is `/opt/PetaMem/PMLS/active/bin`).
3. Type `./estart` in your shell to fire up both server and a CLI client .

After a few seconds, you should see a prompt like

```
pmls>
```

if you do, this means your PMLS server is up and running and a command line client is connected to it. **Congrats!** You can now issue regular PMLS commands. Try the command `info`

```
pmls> info type=sys
```

which should produce an output similar to this

```
handle command... 'info type=sys'  
Vsize: 446.05 MiB ( 467714048 B)  
RSS   : 68618 pages  
PMLS Config Section: server  
  debug           = 0  
  history_cleanup = 1  
  log_level       = 4  
  log_verbose     = 1  
  msg_nosend      = 0  
  msg_verbose     = 5  
  revision        = 521  
  revision_date   = 2010-05-15  
  revision_time   = 12:27:39  
  start           = 2010-06-08 12:28:04  
  time            = 2010-06-08 12:29:05
```

```
Time for command info: 0
```

You now may want to explore the built-in online documentation. Typing

```
pmls> help
```

will print all commands known to the server. Note that not all of these may be available at your installation as their availability depends on license keys obtained and the permissions for the current user. You get detailed help information for a command by typing `help` followed by the commands name. Try this with `li` - the language identification command.

```
pmls> help li
```

To disconnect from the server and shutdown the client, type

```
pmls> shutdown; q
```

1.2 System Requirements Overview

In short, you'll need:

- Hardware equivalent to at least a PC with 1 GHz CPU and 1GB RAM and 5GB free space on your storage system. If you have to make compromises, go for RAM.

- A UNIX-like operating system with Perl 5.10.1 or later installed. The software is being developed and tested on many Linux flavours. Gentoo (<http://www.gentoo.org/>), openSUSE (<http://en.opensuse.org>), Arch (<http://www.archlinux.org/>) and others are the primary OSes used in the software development cycle for this product. Also Debian Linux (<http://www.debian.org>) and SLES (<http://www.suse.com/products/server/>) which are the primary OSes used for testing.
- The Perl interpreter version 5.10.1, 5.12.x, 5.14.x, 5.16.x or later. Versions of Perl older than 5.10.1 are not supported. Versions 5.14.x and 5.16.x are known to run the PMLS, but are not yet certified.
- Some Perl modules from CPAN¹. The file `doc/admin/pmls/needed_modules.txt` lists the required modules. For your convenience, we also ship these modules on the installation DVD.

The detailed hardware and software requirements are as follows:

1.2.1 Hardware Requirements

First, make sure your hardware is sufficient to host PMLS for the supposed operation mode. Most contemporary hardware will easily host one PMLS instance for up to two or three languages. As a rule of thumb, an equivalent to a 32bit x86 with a 1GHz CPU and 1GB RAM should be sufficient. However, please be aware, that on a 64bit architecture the memory requirements nearly double. Table 1.2 covers the hardware requirements in detail.

parameter	32bit: minimum	recommended	64bit: minimum	recommended
CPU	1x1GHz	2x2GHz	1x1GHz	2x2GHz
RAM	1GiB	3GiB	2GiB	6GiB
HD	5GiB	10GiB	5GiB	10GiB

Table 1.2: minimum and recommended hardware requirements on 32bit and 64bit architectures **per PMLS instance**.

The numbers given in table 1.2 are per instance (concurrent for RAM, installed versions for HD-space). That is, if you intend to run 5 PMLS instances on one 64bit server, you should have at least five 1GHz CPU-cores and $5 \times 2 = 10$ GiB RAM for seamless operation². If you intend to keep 3 PMLS releases (different versions) on disk, you should allocate at least $3 \times 5 = 15$ GiB for this.

Please be aware, that PMLS is highly configurable and the min/max numbers mentioned are values that will suit most of the installations. But there are special application cases, where PMLS can be configured to require significantly less but also significantly more memory and CPU resources than mentioned. In any case, the HD requirements are unchanged.

PMLS runs fine on virtual machines. In fact, several pre-installed VM configurations are available for rapid deployment. There are customized versions of PMLS available that run

¹see - <http://search.cpan.org/>

²do not forget some additional memory for the OS requirements

on the ARM architecture.

As the PMLS software is distributed on DVD-ROM, you should have a DVD-ROM device for easy installation (see 2.1.1, pg. 18). However, if there is no such device on your target machine, installation from directory (see 2.1.2, pg. 19) is also possible, but you need at least one DVD-ROM in your network to copy the data.

1.2.2 Software Requirements

Operating System

PMLS server will run on many contemporary UNIX operating systems. There are various clients for Windows, BSD, MacOS and other operating systems. Also, browser-based clients allow usage on any system that has a web browser installed.

System development occurs mostly on Linux, making it naturally the best supported platform. Various Linux distributions are supported, namely Fedora/RedHat, openSUSE/SLES, Gentoo, Debian/Ubuntu.

PMLS has also been deployed on FreeBSD, Solaris and HP-UX.

Perl Infrastructure

The PMLS software suite is implemented in the Perl³ programming language. Perl is part of the base system in all supported Linux distributions and UNIX operating systems. If Perl does not come with your operating system, or not in the required version, please ask your OS vendor for a prebuilt package. Alternatively, Perl is open source and you can compile your own version.

When we say you need “Perl” installed on your system, we precisely mean the perl interpreter and several perl modules.

Perl version	support status
pre-5.10.0	unsupported
5.10.x	supported
5.12.x	supported
post-5.12.2	known to run, not certified yet

Table 1.4: support status of various versions of the Perl interpreter.

If you call the program `pm_install` which is shipped on DVD (see 2.1.1, pg. 18 how to mount the DVD) with the parameter `--test`, it will examine your system and print out the mandatory as well as the optional perl modules missing.

```
bash> ./pm_install --test
```

³see - <http://www.perl.com>

The `pm_install` installation tool should take you through the whole needed installation and upgrade process for any supported distribution. **Please continue with section 2.1.2, pg. 19. Any manual intervention should not be necessary for any supported distribution.**

Where possible, we have included the required Perl modules as part of PMLS. However some modules remain that need to be installed. There are several alternatives how to install the required perl modules on your system:

pre-packaged By far the easiest way is if these modules have been pre-packaged by your OS vendor. E.g. on a RPM-based Linux distribution such as RedHat or openSUSE look for the `perl-module-name.rpm` packages. On Gentoo you can use either `emerge` or the `g-cpan` utility, which provides a convenient way to package build directly from the CPAN repository.

via CPAN shell If you or your system administrator knows the CPAN perl module, installation of additional perl modules is a matter of minutes. You can either type

```
bash> ./cpan
```

or the more verbatim way:

```
bash> perl -MCPAN -e shell
```

any of these will start up the interactive cpan shell which allows you to install additional modules by commands such as e.g. `install Devel::Size`. Simply walk through the list the `./pm_install --test` call has given you and install the missing modules. Because perl modules might depend on other perl modules, by installing one perl module the cpan shell will take care to install all prerequisites.⁴

manually The last resort is a manual install of the required perl modules. However it is the only way if you do not have both an Internet connection and a very bare OS installation. Should your installation require this approach, we have included all mandatory perl modules as source on the DVD. You find them in the `packages/CPAN` directory. Thanks to a standartized way of manually installing perl modules on your unix operating system, even this task is fairly straightforward:

```
bash> tar xzf <perl-package>.tgz
```

```
bash> cd <perl-package>
```

```
bash> perl Makefile.PL
```

```
bash> make && make install
```

Your perl module “perl-package” should now be installed on your system.

1.2.3 License Management

As PetaMem has invested substantial R&D resources to create the PMLS software suite, we seek to protect this intellectual property. It is also our prime concern that protection from

⁴if you're looking for a suitable CPAN mirror, try <ftp://mirror.petamem.com/CPAN/>

unauthorized usage and copying is unobtrusive for the legitimate user.

We have therefore adopted a passive watermark/encryption scheme where every copy of PMLS is marked and encrypted individually, this individual encryption is combined with the software license management. The license file `license.txt` can be found in the PMLS_BIN directory (look in `/opt/PetaMem/PMLS/active/bin`).

A license file contains information about the type of the license (end user, academic license or solution provider), its duration⁵, contact information of the software user and license keys for features and functionality purchased.

As a result, there is no need for on-line authorization, no binding to a specific hardware and no other obstacles for you - the user or administrator. In fact, the PMLS license management will be completely transparent to you during installation and usage (except unlicensed functionality). Please keep in mind that your copy of PMLS has your credentials entangled to it, so make sure there is no unauthorized use, as this is your responsibility and unauthorized copies can be traced back to you.

⁵the date 2099-12-31 23:59:59 means timely unlimited license

A typical license file looks like this:

```
# PetaMem Language Server License
# =====

# Global Information

version      1.0
type         sopr
generated    2010-09-21 15:00:00
expires      2099-12-31 23:59:59

# Contact Information

name         PetaMem GmbH
addr1        Flurstr. 78
city         Fürth
zip          90765
country      Deutschland

tel          +49 911 894 6455
fax          +420 284 680 110
contact      Herr Richard Jelinek
mail         rj@petamem.com

# License class

class        enterprise
concurrent   500

# Feature Keys

PetaMem-ces  910ca80e45cb771088f9381fe58b79f2
PetaMem-deu  6e4fd435796ed6b91c64529a11c4286e
PetaMem-eng  9587d2b0638048b0802b4ac73e4aa183
PetaMem-mul  66bd6d1bf8a0be90afb29e69b04ab086
PetaMem-plugin-li a9c155529753647ee992894a4cd80afe
PetaMem-plugin-mt a7fe9f00e1f61855f601b7a5e3483046
PetaMem-plugin-de 1b6545c78ffc8ba168c7aa3257fb3c04

checksum     uf52wle0GLXSHG005uvqQw
```

The contents of the license file must not be altered, as this corrupts the file and PMLS won't start anymore!

Chapter 2

Deployment and Configuration

*install me in any profession
Save this damn'd profession of writing,
where one needs one's brains all the time.*
Ezra Pound

This chapter describes how to handle the installation(s) of PetaMem (PM) projects¹ on your server. That includes installation, deinstallation, upgrading, downgrading as well as information about the current installation layout.

The central tool for installation and deinstallation purposes is the `pm_install` script which you'll find shipped with every installation media.

```
bash> ./pm_install -h
```

will output (version numbers may vary)² the usage information

```
This is the PetaMem 'pm_install' script rev. 747  
Copyright (C) PetaMem, s.r.o. 2008-present
```

```
Usage: pm_install [options]  
-cpus                number of CPUs pm_install could use  
                    (for performing testsuite)  
-debug              turns on debugging  
-delete <proj>=<rev> remove the revision <rev> for <proj>  
-disable <proj>     deactivate the currently active revision  
-help               prints this usage information and exits  
-list               lists all installed revisions for PetaMem projects  
-set <proj>=<rev>  set the revision <rev> for <proj> as the currently  
                    active and exit  
-source=<path>     set the source directory manually
```

¹There exist 4 PM projects now - namely: PMLS, PMSE, PMLIB, PMWF.

²Also please be aware, that the version of the `pm_install` script is not necessarily the same as the version of the PM project it will install on your server

-test perform only a test of environment

2.1 Install and Deinstall

As for installation, the fastest and easiest way is described in section "Quick Start Guide". However, this will assume many default values and leaves some advanced operation possibilities unused. We will cover the full featureset of `pm_install` here.

2.1.1 Mounting The DVD

After all the prerequisites have been met, the installation itself is a quite simple and straightforward process. Simply insert the DVD into the DVD drive on the machine you want to install given PetaMem project. If the machine has `autofs`³ configured, the DVD will probably be visible in your directory tree within a few seconds like this:

```
bash> df -h

Filesystem                Size      Used Avail Use% Mounted on
/dev/sda2                  9.2G     5.4G   3.4G  62% /
udev                      10M      116K   9.9M   2% /dev
/dev/sda3                  2.9T     166G   2.7T   6% /data
shm                        4.0G         0   4.0G   0% /dev/shm
/dev/sr0                   103M     103M         0 100% /media/PMLS
```

The relevant part of the output is the last line which also shows the directory to which the DVD contents were mounted/mapped. The DVD that comes shipped with PMLS has a Volume-ID 'PMLS' set, so it should be easy to recognise.

However, if you do not have `autofs` configured, you have to mount the DVD manually. If your DVD drive has already an entry in the `/etc/fstab` file, a simple

```
bash> mount /media/cdrom
or
bash> mount /media/dvdrom
```

- whatever the configuration may be - should suffice to see the DVD content under the given directory. On many linux distributions the manual

```
bash> mount -tauto /dev/sr0 /mnt4
```

will work also, and you can see the DVD contents in `/mnt`. Once the DVD is mounted go to the respective directory and issue `pm_install`.

³see - <http://auto-autofs.sourceforge.net/>

⁴on Linux installations, `/dev/sr0` refers very often to the first CD-ROM/DVD drive

2.1.2 Install From Directory

The installation from an arbitrary directory is the same whether this directory is physically on the local disk, a mounted DVD (see previous section), or any imported share (NFS or SMB).

The later might be necessary if the machine where you want to install given PM project does not have any optical drive of its own and gets its software served by some other machine.

There are lots of other possibilities of installation (loopback from an ISO-Image, from USB-stick, from tape and others) - please do not hesitate to contact PetaMem support at support@petamem.com if you have some nonstandard configuration and need help deploying the PetaMem software.

Once you have access to the source directory and have the necessary permissions (root or administrator on the given machine), you can issue the `pm_install` command. A simple

```
bash> ./pm_install
```

in the source directory will do the job, but `pm_install` offers some parameters if you need a nonstandard installation.

In case your data are not in the same directory as the `pm_install` script, you can use the `-source` parameter to fetch them from another directory.

2.1.3 Deinstalling of a PM Project

If you want to disable the currently active revision of PM project, the command

```
bash> pm_install --disable <proj>
```

will perform effectively a “deinstallation” of the project from your server in a way that after applying this command, usually you won’t be able to start up PMLS or work with PMSE. Please be aware, that the software is not really deleted, but simply deactivated to prevent accidental data loss. If all you wanted was to switch between already installed versions, use the `-set <proj>=<rev>` option (see 2.2, pg. 20).

Use this command with caution, because after this PMLS will not start on your server until you again activate a version (either by installing a new one, or by issuing a `pm_install -set <proj>=<rev>` command. Also, keep in mind that by just deactivating a PMLS version, it still takes up space on your HD and with installing more and more versions you could run out of space on your mass storage.

!!! Removal of a PM project version !!!

As `-disable` only performs a deactivation of a PM project version, and this further takes up the space on your mass storage, you might at some point want to completely remove

an obsolete version of the project from your server. The `-delete <proj>=<rev>` option is intended for this, as it will irrevocably remove the specified revision from disk. There are some security measures and side effects:

- The script will refuse to remove the newest revision installed. So if you intend to remove that, you have to do it manually from your OS shell.
- The user is presented with an interactive confirmation message before deletion.
- If the deleted version was the currently active one, the highest available version will be made active.

The PMLS users' [home](#) directories and their contents are not affected by the deletion of a PMLS revision, as they reside in a separate directory. Also, the PMLS data itself can be restored fairly easily from installation media. But any changes you may have done to the *lexica/ontologies* **will be lost** by the `-delete <proj>=<rev>` option. It is because of these data that you should take great care when deleting a revision from mass storage.

2.2 Upgrade and Downgrade

Upgrading and downgrading of a PM project is intuitive and straightforward.

2.2.1 By Installation

Any version you install is being made the currently active version. So if you install a newer version than what was before on your server, you have effectively upgraded a PM project. If you installed an older version than what was active on your server before, you have effectively downgraded a PM project.

Let's look at some examples. Assume you have done a fresh installation of PMLS, so you have just one version installed. This version is also the active one and your [/opt/PetaMem/PMLS](#) directory will look similar to this:

```
root@linux: /> ls -l /opt/PetaMem/PMLS
drwxr-xr-x  9 root root 4096 2011-09-20 16:31 home
lrwxrwxrwx  1 root root  13 2011-09-20 16:31 active -> rev-693
drwxr-xr-x  9 root root 4096 2011-09-20 16:31 rev-693
```

Assume you now obtain new installation media with a newer version. If you simply perform a new installation, the new version gets installed in parallel to the old one and becomes automatically the active one. Your [/opt/PetaMem/PMLS](#) directory will then look similar to this:

```
root@linux: /> ls -l /opt/PetaMem/PMLS
drwxr-xr-x  9 root root 4096 2011-09-20 16:31 home
lrwxrwxrwx  1 root root  13 2011-09-20 16:31 active -> rev-719
drwxr-xr-x  9 root root 4096 2011-09-20 16:31 rev-693
drwxr-xr-x  9 root root 4096 2011-10-25 15:25 rev-719
```

What has happened here is, that the original installation (rev-593) has remained in place, the new installation (rev-619) has been installed in the `PMLS_ROOT` directory and has been

marked as active by putting the “active” link to it. So you can always inspect the active PMLS installation by changing to the `/opt/PetaMem/PMLS/active` directory.

The number of such parallel installations is only limited by the space on your hard disk. You probably have noticed the `home` directory. This contains the home directories of all named PMLS users and will neither change nor be removed when installing new versions or upgrading/downgrading PMLS.

Although it may be obvious, we should mention explicitly, that ex factory higher revision numbers correspond to newer versions of PMLS.

Note: PMSE project has NOT `home` directory.

2.2.2 By Explicit Switch

If you have already several versions of a PM project installed on your server and would like to switch from one to another, you can do so with the `pm_install -set <proj>=<rev>` (see 2.2.2, pg. 21) command, by simply providing the revision of the PM project version that is going to be the active one.

Let’s assume, that after some time you have several revisions of PMLS installed on your server and `PMLS_ROOT` looks something like this:

```
root@linux: /> ls -l /opt/PetaMem/PMLS
drwxr-xr-x  9 root root 4096 2010-09-20 16:31 home
lrwxrwxrwx  1 root root   13 2012-11-24 13:11 active -> rev-811
drwxr-xr-x  9 root root 4096 2011-09-20 16:31 rev-693
drwxr-xr-x  9 root root 4096 2011-10-25 15:25 rev-719
drwxr-xr-x  9 root root 4096 2012-11-24 13:11 rev-811
drwxr-xr-x  9 root root 4096 2013-12-21 14:16 rev-922
```

So there are four revisions, and the active one is 811. There is a newer one (922) and two older revisions (693 and 719). If you want now to upgrade from 811 to 922, you simply issue

```
bash> pm_install -set PMLS=922
```

which will result in a new active link, effectively disabling rev-811 and enabling rev-922:

```
root@linux: /> ls -l /opt/PetaMem/PMLS
drwxr-xr-x  9 root root 4096 2010-09-20 16:31 home
lrwxrwxrwx  1 root root   13 2013-12-21 14:20 active -> rev-922
drwxr-xr-x  9 root root 4096 2011-09-20 16:31 rev-693
drwxr-xr-x  9 root root 4096 2011-10-25 15:25 rev-719
drwxr-xr-x  9 root root 4096 2012-11-24 13:11 rev-811
drwxr-xr-x  9 root root 4096 2013-12-21 14:16 rev-922
```

You do not have to look up the `/opt/PetaMem/PMLS` directory to find out what versions are installed and which one is active. Simply issue the command

```
bash> ./pm_install -list
```

which will return either a message

```
System has no PMLS installation yet.  
System has no PMSE installation yet.  
System has no PMWF installation yet.
```

if there has been no PMLS, PMSE or PMWF installed on this system yet. Otherwise, there will be a list of versions installed and the active one (if any) will be marked with a star. For the above case the output would look like this:

```
Current PMLS installation layout:  
  593  
  619  
  711  
* 822
```

2.2.3 By Implicit Switch

The only situation when the currently active version of a PM project gets switched without explicit user interaction is when `pm_install` tries to maintain a sane (i.e. working) PM project infrastructure on your server. Such a case has been mentioned in the disable/de-installation section (see [2.1.3](#), pg. 19), when deleting a PM project revision - that happened also to be the active revision - would render the system unusable.

`pm_install` then implicitly performs an upgrade by pointing the “active” link to the newest PM project version installed on your server, thus implicitly switching versions.

2.3 PMLS Configuration

After a successful installation of the PM project, there is basically not much work you need to do to ensure a seamless operation. The obvious next step after a fresh new installation would be user management (see [4.2](#), pg. 53). There are of course many knobs and switches you can use to tailor PMLS to your very specific needs and we will discuss them here.

2.3.1 PMLS File System Layout

In order to ease orientation for the PMLS administrator, we will discuss the file hierarchy for an installed version of PMLS. The top-level view on a PMLS installation is described in the following tree-like scheme:

```

PMLS_ROOT--bin
  +-data-+-help
  |      +-lexica-+-a-+-a-+-a
  |      |          |   |   +-...
  |      |          |   |   +-z
  |      |          |   +-...
  |      |          |   +-z
  |      |          +-...
  |      |          +-z
  |      |
  |      +-snlp-+-cs
  |          +-lm
  |          +-top
  |          +-ws
+-doc
+-lib-+-...
  +-PMLS
  +-...

```

bin contains the PMLS binaries (server, clients, startup scripts etc) and the devel subdirectory containing these scripts for development only

data has various data needed for PMLS operation: lexica, statistical information, online help texts

doc here resides all available PMLS documentation

lib Contains the perl modules and libraries that are part of the PMLS software suite.

pmse base directory of the PetaMem scripting environment

Special attention should be paid to the **data/lexica** directory. You will find there three hierarchies of single-letter directories. These form a so-called trie⁵ file structure. This was necessary to be able to cope with the large number of supported languages (PMLS has now full iso639-3 support - see developer manual appendix H, pg. 435).

Say you want to inspect the german ontologies/lexica. As german has the iso639-3 code *deu*, you have to enter the directory **data/lexica/d/e/u**, where you'll find something like that:

```

$ ls -l
total 56
drwxr-xr-x 5 pmls pmls 4096 Sep  8 10:51 base
drwxr-xr-x 4 pmls pmls 4096 Sep  8 10:51 emotion
drwxr-xr-x 4 pmls pmls 4096 Sep  8 10:51 languages
drwxr-xr-x 5 pmls pmls 4096 Sep  8 10:51 petamem
drwxr-xr-x 4 pmls pmls 4096 Sep  8 10:51 sci_all
drwxr-xr-x 4 pmls pmls 4096 Sep  8 10:51 sci_astronomy
drwxr-xr-x 4 pmls pmls 4096 Sep  8 10:51 sci_base
drwxr-xr-x 4 pmls pmls 4096 Sep  8 10:51 sci_biology
drwxr-xr-x 4 pmls pmls 4096 Sep  8 10:51 sci_chemistry

```

⁵see <http://en.wikipedia.org/wiki/Trie>

```
drwxr-xr-x 4 pmls pmls 4096 Sep  8 10:51 sci_geography
drwxr-xr-x 4 pmls pmls 4096 Sep  8 10:51 sci_physics
drwxr-xr-x 4 pmls pmls 4096 Sep  8 10:51 syntax
drwxr-xr-x 3 pmls pmls 4096 Sep  8 10:51 wiki_tr
drwxr-xr-x 3 pmls pmls 4096 Sep  8 10:51 wiktionary
```

These are the present lexica/ontologies for the language you did look-up. The availability of specialized lexica/ontologies will vary depending on language. Also, your installation may contain customer-specific lexica/ontologies that were created by PetaMem or its solution providers or by your staff in-house.

2.3.2 Environment Variables

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer.⁶

The environment definitions for all PetaMem projects to run properly are defined in the file `/opt/PetaMem/conf/active/pm_bash_env_paths`. In order to include these environment definitions add to your bash startup file

```
[[ -f /opt/PetaMem/conf/active/pm_bash_env_paths ]] && \
. /opt/PetaMem/conf/active/pm_bash_env_paths
```

Unless there is a very good reason, the path `/opt/PetaMem` for `PM_ROOT` should not be changed. You can of course install PMLS to any location that fits your needs best, but should at least put a symbolic link from `/opt/PetaMem` to that location. I.e. if you decided to install PMLS under `/disc1/applications/PetaMem` there should be a symbolic link `/opt/PetaMem -> /disc1/applications/PetaMem`. The `pm_install` script should do that for you automatically.

Although testing of new PMLS revisions should probably be done on dedicated machines, you can do so in parallel to a running active PMLS installation, as you can have parallel installations of various revisions on your machine.

In order to switch between the various revisions, you may either use the (cumbersome) way of redefining the environment variables:

```
PMLIB_VERS=rev-344
PMLS_VERS=rev-987
PMSE_VERS=rev-518
```

Or - of course - use the `pm_install -set` option which was designed exactly for this purpose.

If you have already a running PMLS instance on your server, please also make sure you use another port for network connection than the port the currently active PMLS instance is listening on. Also, make sure you use another log path so the logs of the various PMLS instances do not clash.

2.3.3 List of Environment Variables

The following list describes environment variables for all projects.

⁶https://en.wikipedia.org/wiki/Environment_variable

ENV variable	default value	brief description
\$PM_ROOT	/opt/PetaMem	a root of the installation
\$PMBIN_VERS	active	active path for PM binary
\$PMCFG_VERS	active	active path for PM config
\$PMCFG_ROOT	\$PM_ROOT/conf/\$PMCFG_VERS	PM config active
\$PMLIB_VERS	active	PMLIB active
\$PMLS_VERS	active	PMLS active
\$PMSE_VERS	active	PMSE active
\$PMWF_VERS	active	PMWF active

Table 2.1: Environment variables defined in pm_bash_env_paths file

ENV variable	default value	brief description
\$PERL_RL	Gnu	ReadLine clone
\$PMBIN_ROOT	\$PM_ROOT/bin/\$PMBIN_VERS	PM binaries root
\$PMLIB_ROOT	\$PM_ROOT/PMLIB/\$PMLIB_VERS	PMLIB root
\$PMLIB_BIN	\$PMLIB_ROOT/bin	PMLIB binaries
\$PMLIB_DEVBIN	\$PMLIB_ROOT/devbin	PMLIB development binaries
\$PMLIB_INC	\$PMLIB_ROOT/lib	PMLIB include library
\$PMLS_ROOT	\$PM_ROOT/PMLS/\$PMLS_VERS	PMLS root
\$PMLS_BIN	\$PMLS_ROOT/bin	PMLS binaries
\$PMLS_DEVBIN	\$PMLS_ROOT/devbin	PMLS development binaries
\$PMLS_INC	\$PMLS_ROOT/lib	PMLS libraries
\$PMLS_HOME	\$PM_ROOT/PMLS/home	PMLS libraries
\$PMLS_LEX	\$PMLS_ROOT/data/lexica	PMLS lexica
\$PMLS_SERVER	pmls	server executable name
\$PMLS_LOGDIR	/var/log/\$PMLS_SERVER	server log directory
\$PMLS_PID	/var/run/pmls.pid	PMLS PID file
\$PMSE_ROOT	\$PM_ROOT/PMSE/\$PMSE_VERS	PMSE root
\$PMSE_BIN	\$PMSE_ROOT/bin	PMSE binaries
\$PMSE_DEVBIN	\$PMSE_ROOT/devbin	PMSE development binaries
\$PMSE_INC	\$PMSE_ROOT/lib	PMSE libraries include
\$PMWF_ROOT	\$PM_ROOT/PMWF/\$PMWF_VERS	PMWF root
\$PMWF_DATA	\$PMWF_ROOT/data	PMWF data path
\$PMWF_BIN	\$PMWF_ROOT/bin	PMWF binaries
\$PMWF_DEVBIN	\$PMWF_ROOT/devbin	PMWF development binaries
\$PMWF_INC	\$PMWF_ROOT/lib	PMWF libraries include
\$PMCORP_ROOT	/data/library	root-drectory of the PM Corpus
\$PERL5LIB	\$PMLIB_INC	PMLIB include in Perl libs
\$TEXINPUTS	:\$PMLIB_ROOT/shar/latex	add PM sources for L ^A T _E X

Table 2.2: Environment variables defined in pm_bash_env_common file

The \$PATH⁷ variable is also affected. The following variables are included into \$PATH:

⁷\$PATH is a set of directories where executable programs are located.

```
$PMLIB_BIN $PMLIB_DEVBIN
$PMLS_BIN $PMLS_DEVBIN
$PMSE_BIN $PMSE_DEVBIN
$PMWF_BIN $PMWF_DEVBIN
$PMBIN_ROOT
```

There are also used some other ENV variables in the PMLIB and PMSE projects, mostly for running tests.

ENV variable	default value	brief description
\$PM_CONVERTOR_WARNINGS	0	warnings for non-installed convertors
\$PM_KEEP_TMP_FILES	0	keeps test files
\$PM_INTERNET_TESTS	1	run or skip tests needing internet connection
\$PM_MASTERENV	0	switch master environment on - e. g. activate POD tests
\$PM_NO_FUZZY_TESTS	0	no fuzzy testing
\$PM_PROFLOG	0	run tests repeatably and generate basic profiling stats for each sub
\$PM_T_MYRUN_BACK	0	prints debug data when running PMSE::System::myrun
\$PM_COVERAGE_OPTS	0	opts to generate Devel::Cover reports
\$PM_REPORT_WAIT	0	waits for user interaction

Table 2.3: Variables used for running tests in PM projects

2.3.4 UTF-8 Support

PMLS uses for its internal data representation the UTF-8 encoding. Server and client components accept inputs in UTF-8 encoding and their output is also in UTF-8 encoding. In order to see a correct data representation, make sure your terminal supports UTF-8 and your locale also knows this encoding.

A good test to see if your system correctly supports Unicode, is to examine the filenames in the [data/lexica/r/u/s/petamem/lex](#) directory by doing a

```
bash> ls -l
```

command. The output⁸ should be similar to this:

```
-rw-r--r-- 1 root root 347084 Aug 4 15:45 rus_a.lex
-rw-r--r-- 1 root root 763648 Aug 4 15:45 rus_б.lex
-rw-r--r-- 1 root root 1451532 Aug 4 15:45 rus_b.lex
-rw-r--r-- 1 root root 481874 Aug 4 15:45 rus_r.lex
(...)
-rw-r--r-- 1 root root 18238 Aug 4 15:45 rus_ю.lex
```

⁸see the Cyrillic letters in the filenames

```
-rw-r--r-- 1 root root 58619 Aug 4 15:45 rus_я.lex
```

If you can see the Cyrillic letters on your system, you have a working UTF-8 environment. Else there could be one or more of the following problems:

1. The terminal is not set to UTF-8 encoding or you have the wrong locale.
2. The font used is either not an UTF-8 font, or has missing graphical representation for some Unicode code points.
3. Your file system does not support UTF-8 file names.

Please address at least the first two of these issues before you intend to work with PMLS on non-ASCII languages (other than English and Latin that is).

You can check the locale by entering

```
bash> echo $LANG
```

Which should give you a value like `en_US.utf8`, in any case, the “utf8” part is significant.

2.3.5 Server Configuration File

There is a global INI-style configuration file for PMLS which is read at runtime and which contains all configuration information for the PMLS core as well as all the plugins in one central place. The file is located at `PMLS_ROOT/bin/pmlsconf.ini` and contains a lot of self-explanatory comments.

As most INI-style files, it is divided in different sections. Please inspect the configuration file and the comments therein for further reference. For tutorial purposes, we will provide one important feature of the configuration file here: the startup command definition.

When PMLS starts, either by a system boot script or by manual invocation, the PMLS server process inspects the `pmlsconf.ini` file for its global configuration. One of the first parameters to be inspected, is the 'startcmd' parameter in the '[server]' section. This parameter may be defined multiple times and may contain arbitrary PMLS commands as you'd invoke on the PMLS CLI.

Let's say, you would like PMLS to start up with the default english lexica. The corresponding section in the config file would be:

```
[server]

# server startup commands
startcmd = lex_load lang=eng name=base type=lex
```

You can also change all configuration parameters at runtime, as well as reload the global configuration file. Section 4.1.4, pg. 52 covers the relevant commands in detail.

Appendix (Admin Manual) A

Licenses

Purchasing power is a license to purchase power.
Raoul Vaneigem

The source code of the PetaMem Language Server software suite is an aggregation of about one thousand files with some hundred thousand lines of code. The PMLS core files are covered by the PetaMem Software License, external and system modules are covered by one of four possible licenses:

- PetaMem Software License
- Artistic License
- BSD-style License
- GNU General Public License

For a detailed per-file overview of the licensing and copyright information, please see the doc-directory of your PMLS installation.

In those cases, where 3rd party source code comes under a multi-licensing agreement, PetaMem has adopted the following choice of licensing this 3rd party software:

- If the software offers the choice between the Artistic License and the GNU General Public License, PetaMem choose the Artistic License.
- If the software offers the choice between the Artistic License and a BSD-style License, PetaMem choose the BSD-style License.
- If the software offers the choice between the BSD-style License and the GNU General Public License, PetaMem choose the BSD-style License.

A.1 PetaMem Software License

PetaMem grants you a nonexclusive, nontransferable (except as specified herein) license to use *Software* and accompanying documentation (the "*Documentation*"), subject to the terms and restrictions set forth in this Agreement. You are not permitted to lease, rent, distribute

or sublicense (except as specified herein) *Software* or *Documentation* or to use *Software* or *Documentation* in a time-sharing arrangement or in any other unauthorized manner. Further, no license is granted to you in the human readable code of *Software* (source code). This Agreement does not grant you any rights to patents, copyrights, trade secrets, trademarks, or any other rights with respect to *Software* or *Documentation*.

Subject to the restrictions set forth herein, *Software* is licensed to be used by ... user(s) connected to any personal computer, unix workstation or connected set of these (cluster) owned by or leased to you, for your internal use. You may reproduce one (1) copy of *Software* and *Documentation* for backup or archive purposes. Any such copy of *Software* and *Documentation* must contain PetaMem's and its licensors' proprietary rights and copyright notices in the same form as on the original. You agree not to remove or deface any portion of any legend provided on any licensed program or documentation delivered to you under this Agreement.

Software is licensed to you as a single product. Its component parts may not be separated for use on more than one computer.

1. Assignment; No Reverse Engineering

You may transfer *Software*, *Documentation* and the licenses granted herein to another party in the same country in which you obtained *Software* and *Documentation* if the other party agrees in writing to accept and be bound by the terms and conditions of this Agreement. If you transfer *Software* and *Documentation*, you must at the same time either transfer all copies of *Software* and *Documentation* to the party or you must destroy any copies not transferred. Except as set forth above, you may not assign or transfer your rights under this Agreement.

Modification, reverse engineering, reverse compiling, or disassembly of *Software* is expressly prohibited. However, if you are an European Union ("EU") resident, information necessary to achieve interoperability of *Software* with other programs within the meaning of the EU Directive on the Legal Protection of Computer Programs is available to you from PetaMem upon written request.

2. Export Restrictions

Software, including *Documentation* and all related technical data (and any copies thereof) (collectively "Technical Data"), is subject to Export control laws and may be subject to export or import regulations. You agree that you will not export or re-export the Technical Data (or any copies thereof) or any products utilizing the Technical Data in violation of any applicable laws or regulations of the United States or the country where you legally obtained it. You are responsible for obtaining any licenses to export, re-export or import the Technical Data. In addition to the above, the Product may not be used, exported or re-exported (i) into or to a national or resident of any country to which the U.S. has embargoed; or (ii) to any one on the U.S. Commerce Department's Table of Denial Orders or the U.S. Treasury Department's list of Specially Designated Nationals.

3. Trade Secrets; Title

You acknowledge and agree that the structure, sequence and organization of *Software* are the valuable trade secrets of PetaMem and its suppliers. You agree to hold such

trade secrets in confidence. You further acknowledge and agree that ownership of, and title to, the *Software* and *Documentation* and all subsequent copies thereof regardless of the form or media are held by PetaMem and its suppliers.

4. United States Government Legends

Software, *Documentation* and any other technical data provided hereunder is commercial in nature and developed solely at private expense. *Software* is delivered as "Commercial Computer Software" as defined in DFARS 252.227-7014 (June 1995) or as a commercial item as defined in FAR 2.101(a) and as such is provided with only such rights as are provided in this Agreement, which is PetaMem's standard commercial license for *Software*. Technical data is provided with limited rights only as provided in DFAR 252.227-7015 (November 1995) or FAR 52.227-14 (June 1987), whichever is applicable.

5. Term and Termination

The licenses granted hereunder are perpetual unless terminated earlier as specified below. You may terminate the licenses and this Agreement at any time by destroying the *Software* and *Documentation* together with all copies and merged portions in any form. The licenses and this Agreement will also terminate immediately if you fail to comply with any term or condition of this Agreement. Upon such termination you agree to destroy the *Software* and *Documentation*, together with all copies and merged portions in any form.

6. Limited Warranties And Limitation Of Liability

All warranties and limitations of liability applicable to *Software* are as stated on the Limited Warranty Card or in the product documentation, whether in paper or electronic form, accompanying the *Software*. Such warranties and limitations of liability are incorporated herein in their entirety by this reference.

NEITHER PETAMEM, ITS PARENT, SUBSIDIARIES OR AFFILIATES SHALL BE LIABLE IN ANY WAY FOR LOSS OR DAMAGE OF ANY KIND RESULTING FROM THE USE OF THE PROGRAM OR EDITOR INCLUDING, BUT NOT LIMITED TO, LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES.

7. Equitable Remedies

You hereby agree that PetaMem would be irreparably damaged if the terms of this License Agreement were not specifically enforced, and therefore you agree that PetaMem shall be entitled, without bond, other security, or proof of damages, to appropriate equitable remedies with respect to breaches of this License Agreement, in addition to such other remedies as PetaMem may otherwise have available to it under applicable laws.

8. Governing Law And Severability

This License Agreement shall be deemed to have been made and executed in Germany and any dispute arising hereunder shall be resolved in accordance with the German law, excluding its conflicts of laws principles and excluding the United Nations Convention on Contracts for the International Sale of Goods. You agree that any claim

asserted in any legal proceeding by one of the parties against the other shall be commenced and maintained in any state or federal court located in Germany, Nuremberg, having subject matter jurisdiction with respect to the dispute between the parties. This License Agreement may be amended, altered or modified only by an instrument in writing, specifying such amendment, alteration or modification, executed by both parties. In the event that any provision of this License Agreement shall be held by a court or other tribunal of competent jurisdiction to be unenforceable, such provision will be enforced to the maximum extent permissible and the remaining portions of this License Agreement shall remain in full force and effect. This License Agreement constitutes and contains the entire agreement between the parties with respect to the subject matter hereof and supersedes any prior oral or written agreements.

The parties acknowledge to have read and understand the foregoing License Agreement and agree that the action of installing *Software* is an acknowledgment of their agreement to be bound by the terms and conditions of the License Agreement contained herein. They also acknowledge and agree that this License Agreement is the complete and exclusive statement of the agreement between them and that the License Agreement supersedes any prior or contemporaneous agreement, either oral or written, and any other communications between them.

A.2 Artistic License

August 15, 1997

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications. Definitions

"Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified below.

"Copyright Holder" is whoever is named in the copyright or copyrights for the package.

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.
2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.
3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:
 - (a) place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.
 - (b) use the modified Package only within your corporation or organization.
 - (c) rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page

for each non-standard executable that clearly documents how it differs from the Standard Version.

(d) make other distribution arrangements with the Copyright Holder.

4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:
 - (a) distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.
 - (b) accompany the distribution with the machine-readable source of the Package with your modifications.
 - (c) give non-standard executables non-standard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.
 - (d) make other distribution arrangements with the Copyright Holder.
5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.
6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package via the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.
7. C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.
8. Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.
9. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

A.3 GNU General Public License

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by

the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change. b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License. c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program

(or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for

enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR

THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Appendix (Admin Manual) B

Error Messages

```
Can't connect to port 23000 on localhost
Connection can't be established. Check if server is up and running.
```

This message (issued by a PMLS client process) indicates that the client has not been able to establish a contact with the PMLS server process. There may be different reasons for this to happen:

- there is no network connection between the client and the server. Check your network connection in general (e.g. with the `ping` or `tracpath` commands and als eventually check your firewall settings.
- The server may have crashed due to a problem in its configuration, a software error or any other reason. Check that the server is actually up and running. If not, consult the server log files at `/var/log/pmls/server.log` for possible causes of the crash/shut-down.

Also please be aware, that the port number in this example reflects the default port where PMLS server and client communicate. Should you have defined another port number in your configuration, the error message would reflect that. If the server and client processes are listening on different ports, this could be reason for this error message too.

```
Pid_file already exists for running process (8155)... aborting
```

This warning from the PMLS server process indicates, that there seems already one PMLS process running. The current process thus terminated itself. Most of the time you can ignore this warning (especially if you did type `estart` twice), but if this warning keeps coming up constantly, there might be a more severe reason:

- You forgot to terminate the server the last time. I.e. in the client, you just issued the command `quit` or `q` (terminates only the client), but not `shutdown` (terminates the server).
- There is a stale PID file. Usually this shouldn't be a problem, as PMLS detects stale PID files, but you can remove it manually. It's located at `/var/run/pmls.pid`.

Variable interpolation has cycle. Encountered with <key>, seen: <dump>

This indicates a variable interpolation cycle when loading a specific ontology. Variable interpolation cycles can happen if several variables contain themselves mutually, e.g.

a = %b%

b = %c%

c = %a%

Trying to interpolate any of these variables would result in an endless loop. The Discourse Engine detects such cycles when loading an ontology, terminates execution and issues this error.

Part II

User

Chapter 3

First Steps

*Order and simplification are the first steps toward the mastery of a subject
- the actual enemy is the unknown.*

Thomas Mann - The Magic Mountain

3.1 Server Startup & Shutdown

You can start up the PMLS server process either manually or at boot time. The method you choose depends on your preferences. If you install PMLS on a dedicated machine that will serve as language server most of the time, you will prefer to anchor the PMLS startup at boot time. If the usage pattern of PMLS will be similar to that of an application you start from time to time, you will use the manual start up.

3.1.1 Manual Startup

The PMLS server process is started with the `./pmls` command. Issuing a

```
bash> ./pmls -h
```

will print the options the PMLS server accepts. See the following output (version numbers and dates may vary):

```
This is PMLS version: 578
                    date: 2010-08-12
```

```
Usage: pmls [options]
  -h,help    prints this usage information and exits
  -v,version displays the PMLS version and exits
  -port      specify port number server listen to (default 23000)
```

All UNIX and shell commands for process control apply. The following examples show possible ways how to start up the PMLS server process:

1. in the background

```
bash> ./pmls &
```

2. independent of hangup of the invoking terminal

```
bash> nohup ./pmls &
```

3. with a lower priority

```
bash> nice 20 ./pmls &
```

3.1.2 Boot Sequence Startup

Every UNIX-like operating system has a boot sequence where it starts various processes that are relevant or desired for regular operation. You can add pmls startup to this boot sequence and make sure PMLS is up and working every time your host boots.

Linux systems place their init-scripts in directories like `/etc/init.d` or `/sbin/init.d` or have some admin-interface for administration of this sequence. You will find two example init-scripts in the bin directory ([boot_pmls.suse](#) and [boot_pmls.gentoo](#)).

3.2 Connecting and Disconnecting from the Server

You can connect to a PMLS server process with a client that speaks the PMLS protocol. We ship several different client reference implementations, which you'll find in the `PMLS_ROOT/bin/clients` directory:

`client_std` (Perl) The standard command line client (CLI) you can use for your first steps with PMLS, or for administrative tasks. This is the reference client - also used in development and suitable for batch processing (see section 7.1).

`client_mail` (Perl) A mail client to PMLS. With this client, you can send commands to and receive answers from your PMLS server via mail.

`PMLSCClient.exe` (C#) A CLI client implemented in C#.

`client_std.jar` (JAVA) A CLI client implemented in JAVA.

`Client.php` (PHP) A CLI client implemented in PHP.

If you call `client_std` with the help option, you get the list of acceptable command line parameters:

```
bash> ./client_std -he1
```

Will give you this output:

```
This is the PMLS client revision 522 (c) 2004-2010 PetaMem, s.r.o
```

```
Usage: client_std [options]
```

¹the `-he` is actually necessary, because just `-h` would be ambiguous with the `-host` parameter and `client_std` would complain appropriately - just try it.

```
-cmd=cmd(s)          list of commands to execute (give multiple)
-config=config_file  configuration file to be used
-help                prints this usage information and exits
-host=hostname       machine name with running server to connect to
-logfile=filename    specify file for logging
-no_log              disable logging
-pass=password       specify password for user to be used
-port=portnumber     specify port number to connect to
-user=username       name to be used when connecting to server
-version             print client version and exits
```

The PMLS server process has to run before you try to connect to it, else `client_std` fails with an appropriate error message (see also [Appendix B, pg. 41](#)):

```
Can't connect to port 23000 on localhost
Connection can't be established. Check if server is up and running.
```

3.2.1 Batch Connect

The CLI client can perform a connect without any user interaction as long as you provide username and password on the command line. However, please be aware that this is a security risk, as this information might become visible to other users:

```
bash> client_std -user=someuser -pass=somepass -cmd=<cmd>
```

This mode is especially useful if you want to operate PMLS in batch mode. It will startup a client, connect to the server and issue the `<cmd>` command. See further applications and usage in [section 7.1](#).

Please be aware, that for issuing commands directly at startup of the PMLS server, defining them in the global configuration file is the correct way to do. See [2.3.5, pg. 27](#) and the respective documentation in the global configuration file.

3.2.2 Interactive Connect

If you do not provide the username and/or password on the command line, the CLI client will ask for them:

```
bash> ./client_std

Authentication required. Please supply your username and password.
Username (empty entry will terminate session): admin
Enter password for user 'admin':
```

After this, you should see the PMLS prompt.

3.3 Command Syntax

The syntax of commands/queries in the PMLS shell is defined by the following grammar

```
query := <command>[; command]*:
```

```
command := <keyword>[ parameter]*
```

An alphabetically sorted list and description of command keywords is found in appendix C. The ';' semicolon is used as delimiter between commands. If you want to execute more than one command in a command line separate them with ','.

```
pmls> lex_load lang=deu; lex_load lang=eng
```

will load the German and then the English lexicon².

Some commands take lexical entries as parameters and as PMLS lexica can have phrases as lexical entries, you need to cluster these. This is done with double quotes "<phrase1> <phrase2>" or with underscore <phrase1>_<phrase2>.

```
pmls> dict "new york"
```

is the same as

```
pmls> dict new_york
```

Obviously the underscore version is preferable with two-word phrases and the quotation form has advantages with phrases consisting of three or more words.

3.3.1 Positional Parameters

In the early days of PMLS all commands had positional arguments. This was ok for commands that expected just one or two arguments, but it became increasingly inconvenient if more and optional parameters were involved.

Therefore, PMLS is undergoing a change process towards named parameters (see 3.3.2, pg. 48) where appropriate. As a result, you may encounter some commands with positional parameter syntax side-by-side with some commands with named parameter syntax.

3.3.2 Named Parameters

PMLS is currently in a conversion process from positional to named parameters where appropriate. As this transition is a long term task, you may encounter commands with positional parameters, where you would expect named parameter syntax. Furthermore, within the

²which is equivalent to the more concise `lex_load lang=deu,eng`

next releases these commands may change to this new format, which may create incompatibilities with your application.

The preference for named parameters in more complex commands is because positional parameters may be shorter to type, but they are harder to remember (where the position of named parameters is not relevant). Also, handling of optional parameters gets unnecessary complicated for more complex commands in the positional case.

The help text for every command indicates at first glance whether it expects its parameters in positional or named format. e.g.

SYNOPSIS

```
lex_del <lex>+
```

indicates a positional parameters command, whereas

SYNOPSIS

```
bot_add [name=<name>] [onto=<onto[,onto]*>] lang=<lang[,lang]*>
```

shows named parameters. However, please be aware that some commands that have positional parameter syntax, may transform to named syntax in the future.

Chapter 4

System Functionality

*A linguistic system is a series of differences of sound
combined with a series of differences of ideas.*

Ferdinand De Saussure

4.1 System Control

4.1.1 Ending a Session

There are two commands to “end” your PMLS session: `quit` (C.83, pg. 209) and `shutdown` (C.68, pg. 205). The main difference between these is, that `quit` will just terminate your client, whereas `shutdown` will terminate the server process itself and thus all clients connected to it will get no response after the server has been terminated. If you have a single-user configuration with just one server and one client, and wish to terminate both, make sure you first terminate the server and then the client:

```
pmls> shutdown; q
```

4.1.2 System Information

Another system-relevant command is `info` (??, pg. ??) which gives you information about the system status. Please see section 1.1, pg. 9, for an example output of the `info` command. The `info` command itself can take various parameters to give you even more details about the servers internals. Consult the PMLS online help or the command reference for more information. If you need information about the loaded lexica, simply issue:

```
pmls> info type=lex
```

which should give you some similar output to this:

```

pmls> info type=lex
handle command... 'info type=lex'
Available lexica: main
main:
  atoms      = 55271
  buckets    = 18324/32768
  bytes      = 14579230
  change     = 2010-08-13 08:59:15
  depgraph   = mul-relations mul-languages mul-base
              eng-syntax eng-base deu-syntax deu-base
  entries    = 26850
  start      = 2010-08-13 08:59:15
Mind Units Grand Total: >55271<.

```

This output contains detailed information about the lexica present in the repository: a lexicon named “main” is available, has nearly 27.000 entries and takes up about 14MB of memory. The “depgraph” information lists the lexica “main” does depend on and which therefore have been loaded automatically into memory.

4.1.3 Formats of client-server communication

PMLS supports various serialization formats for client-server communication. User could set input and output formats locally for its instance with the help of `cfg_client` (??, pg. ??) Currently we supports following formats:

```

in formats: auto plaincmd std yaml xml json serverdefault
out formats: same std yaml xml json serverdefault

```

For both in and out formats you could also specify `serverdefault` value, which reads default settings from PMLS configuration file.

You could also specify `?` to see which values are currently supported.

Here is an example which sets input format as `plaincmd` (human readable commands) and output format value reads from the server configuration file:

```

pmls> cfg_get in=plaincmd out=serverdefault

```

4.1.4 System Configuration

The commands `cfg_get` (??, pg. ??) and `cfg_set` (??, pg. ??) allow you to change the global system configuration parameters at runtime, where the command `cfg_reload` (??, pg. ??) allows you to read in the global configuration file at runtime. The latter may prove useful if you have done several changes to the global configuration file with a text editor and would like to make the current running PMLS instance aware of these changes.

```

pmls> cfg_get

```

without any parameters will list the available sections. To find out, what current language is set system-wide for localization, you can issue

```
pmls> cfg_get system lang
```

which may give a similar result to this (english).

```
PMLS Config Variable: system lang -> eng
```

In order to change the system-wide localization variable to german, you could issue the command

```
pmls> cfg_set system lang deu
```

There will be no feedback on that operation. However, you can control the new value by issuing a `cfg_get` command again. After the language has been set, a call to e.g.

```
pmls> help bot_add
```

will present you the german help for this command.

4.2 User Management

PMLS is scalable to serve large numbers of users. There is no distinction between “real users” that is a person who has an account on the machine and interacts with PMLS directly or “virtual users” which can be other applications that use PMLS as back end service. Both are the so called **named users** - users known to the system and their number is only limited by the systems’ hardware capabilities. Then, there is the number of **concurrent users** - which are those of the named users that can be connected to a PMLS server at the same time. This number is defined by the license purchased.

4.2.1 Creating and Removing Users

You can set up new users with the command `user_add` (C.71, pg. 206). Every user created this way, will get his own **home** directory and will get default permissions for commands he may issue to the server.

Users can be removed with the `user_del` (C.72, pg. 206) command. Only the user credentials are removed from the system, but not his **home** directory and the data it may hold. Please take care of this home directory by either deleting or archiving it with system shell commands. You will find the home directory at `/opt/PetaMem/PMLS/home/<username>`.

4.2.2 Users: Information and Disconnecting

For a list of users that are known to the system you can issue the command `users_list` (C.73, pg. 206). The command `who` (C.78, pg. 208) will list the users that are connected to the

system - together with some additional information like their origin IP, idle time etc. For verification purposes you can issue the `whoami` (C.79, pg. 208) command which will tell you your current user identity.

In cases where there is some stale user, or a misbehaving “virtual user” (this could be a wild running frontend application that has a bug or is in development), you can disconnect this user with the command `purge` (C.64, pg. 203). Issuing this command will immediately terminate the corresponding client connection to the PMLS server process.

4.3 User Permissions

User management in PMLS allows also for per-command control of user permissions. Basically, you can set for every user a list of permitted PMLS commands this user is allowed to execute. All other commands are forbidden for this user by default. A “virtual user” (some frontend application for NLP services), most probably shouldn’t have access to administrative functionality such as user management and user permissions themselves.

4.3.1 Setting and Unsetting Permissions

The command `perm_add` (C.61, pg. 202) sets specific permissions for a given user, whereas the command `perm_del` (C.62, pg. 202) removes permissions from a given user.

4.3.2 User Permissions Information

The information about each user and his permissions is held in the file `$PMLS_BIN/.user_auth`.

The command `perms_list` (C.63, pg. 203) shows the permissions for the given user (or current user if no user is given).

4.4 File System

As every user has his own `home`¹ directory, there is need for administration of the files these home directories contain.

4.4.1 Uploading and Downloading Files

Uploading a file to the server is done with the command `put <file>` (C.82, pg. 209). This will transfer given file from your current working directory to your PMLS user home directory at the server. If you want to download a file from the server to the client, simply issue the `get <file>` (C.81, pg. 209) command. This will transfer given file from the PMLS users home directory to your current working directory (where you have started the client).

¹we speak of PMLS own file system structure and the subdirs containing the PMLS user home directories. We do not refer to the Linux/UNIX system users home directories.

4.4.2 File Manipulation

There is a restricted set of operations that allows every PMLS user to list, rename, delete and show files. After you have uploaded some files, you could issue the `ls` (C.50, pg. 198) command:

```
pmls> ls
```

which should give you a similar output like this

Contents of PMLS home directory for user ADMIN

```
cs                223 B    2007-10-02 12:23:22
cs-dor-1191321615 285 B    2007-10-02 12:40:15
cs-dor-1191328449 285 B    2007-10-02 14:34:10
de                 59 B    2008-02-17 09:53:18
english.txt       52.3 KiB 2007-08-25 19:23:03
```

Every file in the PMLS home directory is listed with its name, size², the date and time of creation.

You can delete a file with the `rm` (C.66, pg. 204) command, rename it with the `mv` (C.52, pg. 199) command, copy it with the `cp` (C.13, pg. 186) command and show its contents with the `cat` (C.6, pg. 183) command.

4.4.3 Globbing

Traditionally, in a shell you can use a mechanism called **globbing** to perform file name expansion. This mechanism is often used with wildcards such as `*` or `?`. And these are the rules:

- Files whose first character is a dot (".") are ignored unless this character is explicitly matched.
- An asterisk ("`*`") matches any sequence of any character (including none).
- A question mark ("`?`") matches any one character.
- A square bracket sequence ("`[...]`") specifies a simple character class, like "`[chy0-9]`". Character classes may be negated with a circumflex, as in "`*.[^oa]`", which matches any non-dot files whose names contain a period followed by one character which is neither an "a" nor an "o".

4.5 Data Conversion

With the GDCF³ module, PMLS offers access to a wealth of file formats and encodings.

²We use IEC prefixes - see <http://en.wikipedia.org/wiki/Byte>, therefore the KiB, MiB,...

³General Data Conversion Fabric

4.5.1 References

Relevant commands for data conversion operations are `convert` (C.12, pg. 186) and `encode` (C.26, pg. 190). Please look up the corresponding appendix references.

Chapter 5

NLP Functionality

Words are but air; the pen leaves a mark.
Chinese proverb

5.1 Basic Text Functionality

5.1.1 Text Metrics

You can get exhaustive information about the metrics of a given text by using the `txt-info` (see C.70, pg. 206) command. Given we have a file `cs.iso-8859-2.txt` which is some iso-8859-2 encoded¹ text. We can obtain information about it this way:

```
pmls> txt-info file=cs.iso-8859-2.txt encoding=iso-8859-2
```

```
text-metrics:  
  bytes: 5005  
  chars: 5005  
  words: 730  
  lines: 161  
  sentences: 36  
  paragraphs: 14  
  lang-hypo: cs.iso-8859-2
```

The information lists the number of bytes the text needs in memory/on hard disc, the number of characters in text (which can differ from the number of bytes, as the text might be encoded using several bytes per character), the number of words, number of lines, number of sentences, number of paragraphs and finally a list of language hypotheses about languages this text could be written in (or just one if it's pretty clear to the algorithm). The encoding doesn't need to be given if the examined text is in utf-8 encoding - which is the default:

¹http://en.wikipedia.org/wiki/Character_encoding contains more information about encoding schemes

```
pmls> txt-info file=cs.utf-8.txt
```

```
text-metrics:
  bytes: 5547
  chars: 5005
  words: 730
  lines: 161
  sentences: 36
  paragraphs: 14
  lang-hypo: cs.utf-8
```

This is exactly the same text as above, but just with a different encoding (utf-8). Here we can see the difference between number of bytes and number of characters in text, as UTF-8 encoding uses several bytes for one czech character. We can also observe, that the information about the structure of the text (number of sentences, paragraphs, lines etc.) remains the same.

5.1.2 Get Unique Tokens

To get a list of unique tokens from a text, use the `tgut` (see [??](#), pg. [??](#)) command:

```
pmls> tgut de.txt-ut de.txt
```

Given the file `de.txt` exists in the user's `home` directory, this will create a file named `de.txt-ut` containing the unique tokens found in the file `de.txt`. You can inspect the contents of the `ut`-file with the command `cat` (see [4.4.2](#), pg. [55](#)).

5.2 Diacritics Operations



A diacritic (also diacritical mark, diacritical point, diacritical sign) is an ancillary glyph added to a letter, or basic glyph. The PMLS diacritics processor allows you to handle texts that have diacritics in a way that diacritics can be removed or restored.

1st Fit Reconstruction operation. The most probable alternative of a word (either with or without diacritization) is used. Because the "most probable" alternative (see Restrictions below) is not always the correct one, there is:

Choose Reconstruction operation. If there is more than one alternative, the user is presented all of them to be able to make his own choice. While this requires most interaction on the users side, the preset is equivalent to the '1st Fit' mode of operation and in more than 90% of all cases correct.

Remove Removal operation. All diacritics are removed from a given text. This is the inverse functionality to the reconstruction operations.

5.2.1 References

Relevant command for the Diacritics Operations plugin is `tdiaop` (??, pg. ??). Please look up the corresponding appendix references.

5.3 Language Identification



Language identification is the process of determining which natural language a given content is in. PMLS has several methods to determine the language of a given text. These are dictionary, rule-based heuristic and statistical methods.

5.3.1 Dict

A dictionary based language identification method. The given text is iterated word by word trying to lookup these words in the dictionaries of the system. This method is computationally quite expensive, but suited for short texts, where statistical methods often fail.

The set of supported languages is identical to the union of the supported source languages of all loaded lexica:

5.3.2 NGram

This is the industry standard of language identification. NGram language identification is fast and reliable (if there are good so called "language models") for mid-sized texts.

The supported language/encoding combinations are:

5.3.3 NVect

This is a PetaMem proprietary technology for language identification. It is in some respect a generalization of the NGram method and computationally expensive, so not suited for long texts, but it yields better results than NGram for short texts.

The supported language/encoding combinations are:

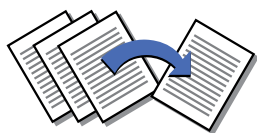
5.3.4 Smart

As the name of this default parameter says, it switches recognition to a mode that tries to be smart, i.e. to apply the best method suitable for a given text. This should be the best choice for all cases.

5.3.5 References

Relevant commands for the Language Identification plugin are `li` (C.48, pg. 197) and `li-llm` (C.49, pg. 198). Please look up the corresponding appendix references.

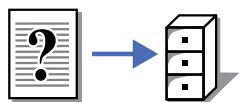
5.4 Text Summarization



Text summarization, also referred to as automatic summarization or document summarization is the creation of a shortened version of a text by a computer program. The product of this procedure still contains the most important points of the original text.

PMLS implements multi-document summarization, but can also handle single-document summarization tasks.

5.5 Text Categorization



Text categorization, also sometimes referred to as text classification or document classification is the task to assign a document to one or more categories, based on its contents. PMLS implements an unsupervised document classification, where the classification is done entirely without reference to external information.

5.6 Spell Check

**Per aspera
at astra**

The PMLS statistical spell checker uses word probabilities, as well as probabilities of co-occurrences and a sophisticated language model to perform spell checking and correction.

The command `tcheck` (see ??, pg. ??) implements a simple spell checking mechanism.

5.7 Morphosyntactic Analysis

PMLS has a part-of-speech tagger implementation that is an extension to and unification of the MULTEX-west and MULTEX-east² tagset.

The relevant command is the `tag` (??, pg. ??) command. You can specify the input to be tagged as text directly on command line (with the `text=` argument) or as file which resides in the PMLS user directory (with the `file=...` argument). The API is the same for all supported languages. If you specify the input text on the command line, the output of the tagging operation is also presented on screen. If you specify it as file, a new file with the tagging information is created in the users PMLS home directory.

```
pmls> tag text="George is living in a house."
```

will yield as output

```
George/NNP is/VBZ living/VBG in/IN a/DET house/NN ./PP
```

You could as well omit the `mode=simple` parameter, because it is the default. If you would like to use the MULTEX-east compatible tagger you first have to load the lexicon corresponding to the language you want to tag. Please see section 6.1, pg. 63, especially 6.1.2, pg. 63 how

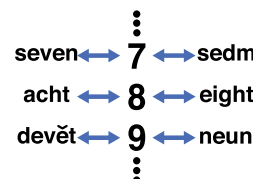
²see - <http://nl.ijs.si/ME/V4/>

to load a lexicon. If you have loaded the corresponding lexicon successfully, you can issue the “multex” tagger:

```
pmls> tag mode=multex file=english.txt
```

will take the content of the file `english.txt` and output is tagged according to the MULTEX specification. Please see section 4.4, pg. 54 on how to upload, download and manipulate files as PMLS user.

5.8 Procedural Word Processor



Morphosyntactic parsing means a system is able to parse both the structure of a sentence as well as the structure of the morphology of a word. However, many languages form words procedurally, which is out of reach for many natural language parsers. PMLS is able to parse these words and to assign a semantic meaning to them. We call this a *Procedural Word Processor* (PWP). As an example to this the handling of numerals has been implemented. PMLS can transform numbers to a textual representation in many languages and it can parse the textual representation to a numeral value. This way, translation of numerals is easily accomplished:

```
pmls> num2txt
```

```
pmls> txt2num
```

will both output the list of languages supported. Something like:

```
Known langs: fra ces por ind afr swe spa nld deu ita
```

The `num2txt` command will also output a range it can operate in when given the language code:

```
pmls> num2txt deu
```

```
Interval: 0 999999999
```

So given a conversion command like

```
pmls> txt2num cs sedm tisíc čtyři sta třicet dva
```

will output `cs: sedm tisíc čtyři sta třicet dva = 7432`. And by entering such a number into a `num2txt` command for German:

```
pmls> num2txt deu 7432
```

we obtain `siebentausendvierhundertzweiunddreissig`. Find that in any dictionary.

5.8.1 References

Relevant commands for the Procedural Word Processor plugin are `txt2num` (??, pg. ??) and `num2txt` (??, pg. ??). Please look up the corresponding appendix references.

Chapter 6

NLU Functionality

The noblest pleasure is the joy of understanding.
Leonardo da Vinci

6.1 Lexical Operations

6.1.1 Nomenclature

PMLS organizes all its knowledge¹ in a central data structure called **The Vault**. This repository itself is a pool of **Lexica**, where each lexicon contains information about a specific domain. Such a lexicon itself can consist of several thousand - not seldom hundreds of thousands - **Entries**. An entry is made up of a **Key** and a **Meaning**. For an in-depth discussion of the nomenclature of a lexical entry see the semantic lexicon API reference section ([F.1](#), pg. 229).

6.1.2 Load and Save Lexica

Lexica can be loaded with the `lex_load` (??, pg. ??) and saved with the `lex_save` (??, pg. ??) command.

Please be careful not to load a lexicon if one of the same name already exists in the repository. The system tries to auto-merge all contents, which is computationally very intensive.

You can control if loading the lexicon was OK by issuing the command `info type=lex` - see more about this in [4.1.2](#), pg. 51.

You can also issue the `lex_list` (??, pg. ??) command to see what lexica are available on disk.

6.1.3 Create and Delete Lexica

Creation and Deletion of lexica can happen in many ways. You can create a new lexicon in the repository “from nothing” with the `lex_new` (??, pg. ??) command. You could also clone

¹about natural language, the world and everything

an already existing lexicon with `lex_clone` (??, pg. ??).

You can delete a lexicon from the repository with `lex_del` (??, pg. ??).

Moving a lexicon within the repository is equivalent to copying the lexicon to a new location and deleting the source lexicon. It is also equivalent to renaming the lexicon and done with the `lex_mov` (??, pg. ??) command.

6.1.4 Set Operations on Lexica

On the topmost level of manipulating lexical data, you can perform set operations on lexica. The command `lex_set` (??, pg. ??) lets you form the union, intersection, difference and symmetric difference between two lexica. Each of these operations is performed on the entries of the respective lexicon. The following command performs an intersection of lexica “eng” and “deu”:

```
pmls> lex_set eng:deu:ise
```

This operation will result in entries like “hat” (the english word for a certain form of head covering, german for “has”) being merged, of course keeping their differentiated meanings untouched. This functionality effectively helps establishing specialized disambiguation lexica.

Please be aware, that the second lexicon mentioned (“deu” in our example) is always the source lexicon and gets destroyed as such. The first lexicon is the target lexicon and remains in the repository, although modified after the operation. Therefore consider saving your own copies of the lexica before performing set operations on them, also `lex_clone` (see ??, pg. ??) is an alternative.

6.1.5 Adding Entries

You can manually add new entries to an existing lexicon. The two relevant commands for this are `lput` and `lrep`. Both have the same syntax:

```
pmls> erep en house=SY(en,V),TR(de(beherbergen))
```

```
pmls> eput en house=SY(en,N),TR(de(haus))
```

The main difference being, that `erep` replaces any entry that might have been in the respective lexicon before, and `eput` merging the entry to an already existing one. Thus, after the first command, `show house` would display

```
,---.      ,-----.      ,---.
|,  |+-|SY(,)|-+-|en |
`---' | `-----' | `---'
      |              | ,---.
      |              +-|N |
```

```

|           `-----'
| ,------.
+-|TR(de(haus))|
| `-----'

```

and after the second command the entry would look like

```

,---. ,-----. ,---.
|, |+-|SY(,)|+-|N |
`---' | `-----' | `---'
|           | ,---.
|           +-|V |
|           | `---'
|           | ,---.
|           +-|en |
|           `---'
| ,-----. ,-----.
+-|TR(de(,))|+-|beherbergen|
| `-----' | `-----'
|           | ,---.
|           +-|haus|
|           `---'

```

effectively representing the ambiguous nature of “house” as building and “to house” as action.

6.1.6 Finding Entries

PMLS offers a variety of methods to find entries within the repository (thus in all the lexica that are currently loaded in the repository). The most versatile of these methods is the `find` command. You can look for entries by exact, approximate or regular expression based search.

```
pmls> find exact house
```

```
pmls> find approx:1 tiger en
```

```
pmls> find regex ar|z en
```

The first invocation will print “house” if such an entry is in the lexicon or output `find: no results.` if no such entry could be found. The approx-invocation will output something like

```

"geiger" "liger" "maiger" "metier" "niger" "stager" "stiver" "tiber" "tier"
"tierce" "tiered" "tiers" "tiger" "tigers" "tigris" "tiler" "tilers" "timer"
"timers" "titer" "untier"

```

The regular expression invocation will print all entries within the english lexicon that end on “ar”.

The “exact” search method has $O(1)$ runtime characteristics, whereas the approximate and regular expression versions have $O(n)$ runtime characteristics.

Another lookup method is the `dict` command. This performs like a simple dictionary. Let us assume you have the lexicon named “en”² loaded, a call of the form:

```
pmls> dict mouse fr:en
```

will return a translation or several possible translations for the word “mouse” found in the en lexicon for the target language fr, “souris” in this case. We could have omitted the en from the command and write just `dict mouse fr` - then the command would have searched in all lexica within the repository to find a match.

6.1.7 Inspecting Entries

The `show` command can be used to inspect an entry within a lexicon. As it is a visual representation to the complete internal data representation of this entry, you can see all properties at one glance:

```
pmls> show math en3
```

will show - if an english lexicon is loaded - something like

```
,---.  ,-----.  ,---.
|,  |+-|SY(,)|+-|N  |
'----' | '-----' | '----'
      |           | ,---.
      |           +-|en |
      |           '----'
      | ,-----.  ,-----.  ,-----.
+-|TR(,)|+-|cs(|)|+-|matematika|
| '-----' | '-----' | '-----'
      |           | ,-----.
      |           +-|matika|
      |           '-----'
      |           | ,-----.
      |           +-|da(matematik)|
      |           | '-----'
      |           | ,-----.  ,-----.
      |           +-|de(|)|+-|mathe|
      |           | '-----' | '-----'
      |           | ,-----.
      |           +-|mathematisch|
      |           | '-----'
      |           | ,-----.
      |           +-|hu(matek)|
```

²ok - it *is* the english lexicon, but the user should be aware, that in fact the english lexicon could have almost any other name

³if you omit the lexicon specification, the term is searched for in all lexica

```

|           `-----'
| ,-----,
+-|hypernym(science)|
|           `-----'

```

You can see some translation information (TR), some syntactic information (SY) - mainly POS tags and some entries have additional semantic information. The data organization is a N-ary tree. More about the internal data representation formalism in chapter 7, pg. 81.

6.1.8 Modifying Entries

To refine the lexicon, you can modify its entries in many ways. You can copy, rename (i.e. move), delete them. Or you can modify their internal structure in a more sophisticated way. The copy operation command `ecpy` can copy one or more entries within one lexicon or in-between lexica:

```
pmls> ecpy en:phrase1:phrase2:en2
```

will copy “phrase1”⁴ from the english lexicon to “phrase2” in a lexicon named en2. If an entry exists at the target location, it is melted with the copy of the source entry, else it is simply created. We can delete one or more entries with the `edel` command which is very similar in nature:

```
pmls> edel en:house de:maus
```

will delete the entry “house” from the lexicon “en” and the entry “maus” from the lexicon “de”. If you want to move an entry from one location to another, you could of course use the `ecpy` command followed by an appropriate `edel` command. However, it is faster to use `emov` for this:

```
pmls> emov en:phrase1:phrase2:en2
```

6.2 Discourse Engine



The discourse engine integrates in a modular fashion as plugin into the natural language processing framework PMLS offers. This way, the module can access many functions other products in this category (so called “chatbots” or “avatars”) do not have at their disposal. Let alone the grasp of natural language is a complex and multilayer task. The further steps, required for a successful communication such as categorization of the acquired, formulation of an adequate answer or reaction, require additional processes, that are available to the system we review here.

⁴we deliberately call it phrase here as it may contain spaces

6.2.1 Analysis

Regular Expressions

The predominant share of today's available chatbot implementations, works with a simple keyword-based recognition. If one or several keywords are identified in a query, the associated and hardcoded answer is fetched from the database and presented to the user. Often proprietary formalisms are sketched to be able to work with placeholders.

The PMLS Discourse Engine uses for recognition of user input so called **regular expressions** as fundamental mechanism. This is a formal language of the lowest step of the Chomsky-hierarchy (type-3). But aside from this theoretical categorization is an important fact: the power of regular expressions is sufficient to describe the morphology of a natural language. So with this basic feature alone, the PMLS Discourse Engine leaves other available systems behind.

All inputs of the form:

```
Das ist Helmut's Auto
Das ist Martin's Regenschirm
Das ist dem Richard sein Computer
Das ist Alonso's Rennboot
```

can be caught with these very simple rules and trigger an “understanding” answer.

```
i = Das ist (\w+)s (\w+)
i = Das ist dem (\w+) sein (\w+)
o = Vermutlich ist $1 sehr stolz auf sein $2
```

With this, we have barely scratched the surface of possibilities of this formalism. Assume, we'd like to exploit these inputs too:

```
Des is Helmut's Auto
Das is Luis's Regenschirm
Dat ist Marcel's Brosche
```

then we can do so with small modifications of the ontology:

```
i = D[ae][st] ist* (\w+)(ens|s) (\w+)(e*)
i = D[ae]s ist* dem (\w+)(~*) sein (\w+)(e*)
o = Vermutlich ist $1 sehr stolz auf sein$4 $3
```

Even this does not show the full spectrum of possibilities of this basic mechanism of the PMLS Discourse Engine.⁵

Procedural Parser

The possibilities of regular expressions are not sufficient to cover the more abstract aspects of natural language.

One example, what advantage the infrastructure PMLS offers, can be numerals. It is not feasible to have all these in a dictionary - it is hardly likely you would find “Seven-Hundred

⁵Strictly spoken, the PMLS Discourse Engine uses extended regular expressions. More literature and training material is available from PetaMem. Also please see - <http://regex.info>

Fifty-Eight” as an entry in a dictionary. Also the recognition of these numerals is beyond the scope of regular expressions. But PMLS has a parser to recognize numerals (cardinal and ordinal numbers) for more than ten languages. If the user enters “I am thirty-seven years old.”, then the age of the user is interpreted correctly as “37”.

Also, complex terms (see section 6.2.2) are accepted and processed.

Language Identification

Because the Discourse Engine has been designed inherently multilingual, the language identification plays a central role. The system must know, of course, what language the input is, to be able to answer correctly. This is especially important in generic answers (computations, acknowledgements etc.), because these cannot be derived from an input-output rule.

Equally important is the correct language identification if the intention of the user is not recognized at all or if this language is not active in the current instance. Instead of a “I did not understand you” in english or german, this answer should be formulated - together with the information what languages are supported - in the same language the query was formulated.

PMLS supports several methods of language identification and these are of course available to the Discourse Engine. Unfortunately, the available statistical methods are unapt in this case, because they need input of at least 100-150 characters to identify a language with a good hit rate. But there is a dictionary-based method available, that has not this constraint. For this, the lexica have to be held in memory. This is resource- and cost intensive, but the possibility is there.

6.2.2 Generation

Answer Choice

When interacting with a dialogue partner, we would like to be understood. However, equally important is that the opponent does not act mechanistic/deterministic. If we would tell our name a human conversational partner ten times - we certainly would earn an appropriate comment. Also, we do not expect to get *exactly* the same reaction for the same input.

The ontology formalism allows different answer strategies - per rule. From a set of answers either a defined, all cyclic, all random or all random and cyclic (only once a cycle) can be chosen.

Procedural Generator

Exemplary for the procedural abilities of the Discourse Engine we would like to mention its possibility to evaluate terms. As a simple use case, a calculator was implemented. Thus it is possible to issue queries of the form:

```
Berechne mir mal 27 mal 18.
Was ist sieben + drei
Wieviel ist 8 hoch vier
...
```

to the Discourse Engine and to get a correct answer. The basic arithmetic operations, percentages, exponentiation and modulo are supported.

The Discourse Engine also has more procedural abilities. Because of its perception to time and space (in the sense of time elapsed and spatial relations) and ability of qualitative spatial and temporal reasoning, you can issue queries to this system that need deduction capabilities in these domains.

6.2.3 Memory

We expect from a dialogue partner, that he keeps informations from the current talk in mind. If we mention “I am xx years old.”, then we normally expect, that our opponent can answer the later question “How old am I?”.

The PMLS Discourse Engine can remember the previous course of conversation as well as mentioned facts. Here we simulate a simple but generic capacity for remembering. All inputs of the form:

```
Ich bin/habe/kann ...  
[name]/Er/Sie/Es ist/hat/kann
```

can be captured and stored, to be answered in the later course of conversation when queries such as:

```
Was/wie bin/habe/kann ich ...  
was wie ist/hat/kann [name]/Er/Sie/Es ...
```

arise.

6.2.4 Internal State Model

As mentioned in section 6.2.2 a primitive-deterministic acting of a chatbot is counterproductive. An answer strategy that selects randomly one answer from some alternatives is also recognized as such after some time.

The PMLS Discourse Engine has an elaborated internal state model to be able to react adequately to user input. Moreover, the emotions modelled with this are used to steer a visualisation of the chatbot. A total of 24 emotional states is available:

neutral calm, relaxed, pensive, bored, interested, surprised, shamefaced, confident

positive pleased, hopeful, proud, trusting, empathic, satisfied, amused, bold

negative worried, fearful, angry, sad, irritated, disappointed, annoyed, despaired

6.2.5 Usage Tutorial

Create a German chatbot with the `bot_add` command

```
pmls> bot_add lang=deu
```

which will result in the following output

Bot added to pool: Elric [deu(369)] answered: 0/0

We now have a bot that can have conversations in German. Let us greet the bot with

```
pmls> say this="hallo"
```

Guten Tag.

The bot should be able to perform some computations for us

```
pmls> say this="was ist 5 mal 8?"
```

Das Ergebnis ist 40.

Now let's test the memory capabilities of the bot

```
pmls> say this="Mein Auto ist schnell."
```

Verstehe.

```
pmls> say this="Wie ist mein Auto?"
```

schnell.

6.2.6 Create and Extend Ontologies

The Discourse Engine module offers a very powerful formalism to define ontologies and domains of discourse. While this formalism can get as complex as regular programming effort, fast results are possible with just a few keystrokes. We will present all the ingredients necessary to create your own ontologies or to extend the existing ones. As the Discourse Engine uses various subsystems of the PMLS software suite, we will for the sake of brevity just refer to the parts of this documentation mentioning them. First results are possible within 5 minutes, although mastering all possibilities of this component may take several months - or more.

As mentioned in the general overview section, regular expressions are a core technology used for the pattern matching capabilities of the discourse engine and as such are key to any ontology. We suggest the book *Mastering Regular Expressions* by Jeffrey Friedl (see also <http://regex.info>) as a complete reference of regular expressions. A short reference can also be found in appendix G, pg. 431.

.ont Files

An ontology is basically defined by a set of .ont-files that live side-by-side with semantic lexica (.lex files) in the `PMLS_ROOT/data/lexica/<lang>/<ontology>` directories. These files are similar to INI-Files⁶ and as such each file contains of one or more sections of the form

```
[section]
key1 = value1
key2 = value2
```

⁶see http://en.wikipedia.org/wiki/INI_file

As an example let us examine an ontology that catches any user input containing the word 'Aha' and answers with 'Oho':

```
[topic=aha_oho]
o = Oho
i = aha
```

That's it! the key 'i' stands for input, the key 'o' stands for output. If our input contains the string 'aha', then the system will issue the string 'Oho'. The section contains at least the topic and should be - in general - unique per ontology. We will cover that later. Obviously, for 'i' key/value pairs, regular expressions are allowed as values.

The .ont-files extend the simple INI-concept by allowing a richer variety of constructs:

multiple input keys It is possible to define more than just one 'i =' key/value pair. In that case, all input-key/value pairs will be examined and if one of them matches, the corresponding output key/value pair ('o =') will be sent to output.

here documents Lines can have arbitrary length. However, if - for visual reasons - the value is too long for a single line, you can provide a better visual markup by using a here document (Perl style, see http://en.wikipedia.org/wiki/Here_document) for the value.

A more realistic example shall sum up more of the features of an .ont-file:

```
# ABOUT PETAMEM (WHAT/WHO/WHAT DOES)
[topic=q_petamem_what]
o = <<'E00'
<a href="http://www.petamem.com">PetaMem</a> ist ein Anbieter
sprachverarbeitender Systeme. Meine Wenigkeit (Diskurs Engine) ist
beispielsweise so ein (Teil-)Produkt dieser Firma.
E00
i = (wer ist|was macht)( denn)? petamem
i = welcher? (aktivitaet|taetigkeit) (uebt|geht) petamem (aus|nach)
e = hopeful
```

We see two input definitions. The first would match inputs like "wer ist PetaMem", "was macht denn PetaMem" etc., the second matches e.g. "welche tätigkeit übt PetMem aus".

As you may have noticed, all input definitions are lowercase and contain no diacritics. That is, because the Discourse Engine normalizes the user input in that it strips superfluous white spaces, lowercases the input and converts diacritics to an undiacritized form. So if the input definition is `i = taetigkeit`, the real input could be "Tätigkeit", "taetigkeit" or even "tÄtigkeiT" and all would still match.

Also, we can see a multiline-definition of the output using a heredoc ("Here Document"). In this special case it is a non-interpolating heredoc. We will cover the difference between interpolating (`x = <<"END"` and non-interpolating `x = <<'END'` heredocs when covering variable interpolation in .ont-files.

'#' is starting a comment. Contrary to regular INI-Files, the comment doesn't need to be at the start of the line and can also occur as trailing comment on a single key/value line -

except if it is a here document, in which case the comment would be interpreted as part of the value.

The 'e' key is optional and denotes the emotion this particular section will contribute to the chatbots mood if matched. For a list of possible emotions see the section 6.2.4. For a description how the 'e' definition given can affect the internal state model, see section 6.2.7.

Multiple Answers

As we have seen, there can be more than one input definition per section. Same is true for the output definitions. A perfectly valid section might be:

```
[topic=q_bot_name|0=get_bot_name|mode=rand]
o = My name is %0.
o = I'm called %0. #'
o = %0 is my name.
i = (what('s| is)|tell me) your name
i = what name (do you have|were you given)
i = (how|what) do you call yourself
i = who are you
e = pensive
```

We can see four input definitions and three output definitions. As there may be many ways to ask a single thing, there also often are many ways how to provide an answer. This section will let the bot answer to the query about his name in three different ways. The `mode=rand` definition in the section does exactly that: it selects one of the defined answers by random. As it is the default behavior, it can be omitted. If just one answer definition is provided, `mode=rand` is still the default and ok, because the probability to choose 1 answer out of 1 is 100%.

Other possible answer-selection modes are:

constant e.g. `mode=0`, which would always take the answer with the first index (starting at 0). This can be used if there are several answer definitions, but you would like to use just a single one of them and would not like to delete the unused from your ontology.

cycle `mode=cycle` will cycle through the answer definitions. Compared to the `rand-mode`, this has the advantage that the choice of answers is more predictable (for the ontology author) and if a large enough set of possible answers is provided, the user is less likely to be confronted with repetitions.

weighted randomization This mode, defined by `mode=rand(x,y,...,z)`, provides a means of answering by weighted probabilities. So some answers are less probable than others.

An example should clarify the weighted answer randomization:

```
[topic=example|mode=rand(.1,.2,.7)]
o = This answer has a probability of 10%
o = This answer has a probability of 20%
o = This answer has a probability of 70%
i = say something
```

The sum of the single components should always be exactly 1 (100%), also you should always ensure to define the same number of weights as there are answer definitions.

Variable Interpolation

We have mentioned “interpolating” and “non-interpolating” here documents. Interpolation is the process of replacing variables in documents (often templates) with the value they represent. Variable interpolation exists in .ont-files too:

Assume the variable 'USER' has the value 'admin'. If the discourse engine encounters a string '%USER%' in an interpolating context when parsing a .ont-file, the variable is replaced with its value ('admin' here). What is an interpolating context?

- Every single-line value in an .ont-file is in interpolating context
- Every heredoc (multi-line) that has been set into interpolating context (= <<<"<END-MARKER>")

The only non-interpolating context is a heredoc that has been set to non-interpolating context (= <<<'<END-MARKER>').

```
[topic=example]
o = This will interpolate: %VAR1%
o = <<<"E00"
This will interpolate too: %VAR2%
E00
o = <<<'E00'
This will not interpolate: %VAR3%
E00
i = input can have interpolation too: %VAR4%
```

Where are the variables for interpolation defined? In every ontology, there can be a file named vars.ont, which can contain a [vars] section. The key/value pairs in this section define the variables, where the keys are the variable names and the values their contents as the following example illustrates:

```
op_add    = \+|plus|und
op_div    = \|dividiert\s+durch|geteilt\s+durch|durch
op_mul    = \*|x|mal
op_sub    = \-|minus|ohne
operator  = (?<op>%op_add%|%op_sub%|%op_mul%|%op_div%)
```

For such variable definitions, there is an “overlay”-mechanism in place. Variable definitions are read in from the vars.ont file in the corresponding ontology **in the order the ontologies are requested to load**. This means, that while variables are added at a later point, may overwrite already existing variable definitions. This behavior is intentional and does reflect a concept of specialized ontologies, where the more generic ones are loaded first and the more specific ones later.

However, this approach brings also certain risks if a variable is overwritten by a specific ontology and the new content breaks some behavior of the base ontology. Therefore great care must be taken if using this feature.

Variable interpolation may be recurring. In the example above, we see that single specific operators get defined, and then a generic operator is defined as an aggregation of these simpler elements. If an ontology is loaded and the variable `%operator%` is encountered it is expanded to its defined value. This expanded value is again examined for variable interpolation. The variables found are interpolated. This process will repeat until there is no more interpolation and thus allows for arbitrary depth of nested variables.

Because of this nested interpolation mechanism, it is possible to accidentally create cyclic structures. If these are detected, program execution is terminated with a respective error message (see [B](#), pg. 41).

Macro Extension

While regular expressions are a good tool for pattern matching and variable interpolation offers better comfort and maintenance, these mechanisms are still static and make no use of the semantic lexica that PMLS has to offer. Imagine you would like to match a word and all its synonyms. You could of course hand code those manually as a regular expression, or even as a regular expression value of a variable, but this is tiresome and error-prone.

The macro extension capabilities of the Discourse Engine module offer a more comfortable way to account for lexical and semantic knowledge. Assume you would like to catch the word 'beautiful' and all its equivalents. Simply use the EQU macro:

```
[topic=compliment]
o = Thank you.
i = you are EQU<beautiful>
e = pleased
```

The EQU macro will find all equivalents (synonyms) in the PMLS semantic lexicon and build a suitable regular expression to match any of these (and nothing else). This regular expression is then placed instead the EQU<beautiful> call, where it can match.

Another very powerful macro is the EVL<...> macro which accepts any Perl code as argument and evaluates it. For a complete overview of available macros see appendix [E](#), pg. 217 especially table [E.1](#), pg. 218.

Mixing variable interpolation with macro extension is possible and provides a very powerful feature. Should your application require it, you can define variables to have (nested) macros (and - again - variables) as values. You can give macros variables as arguments. While the expressional power of this formalism is unprecedented, it also has its drawbacks. First and foremost, the maintainability of such nested complex definitions suffers. Second, while it is possible to detect cycles in variable definitions, the PMLS does not detect any mutual cycles that may result from e.g. a variable defining a macro which has a variable as its argument and this argument-variable again defines a macro, problems will arise. For the sake of argument see this nonsensical and insane example:

```
zust = zuständig
zequ = EQU<%zust%>
zrev = EVL<reverse %zust% . %requ%>
requ = EQU<%zrev%>
```

If you have such or similar constructs in your ontology definition and it forms a cycle between variable interpolation and macro expansion, the PMLS will hang in an endless loop when reading in this ontology. Best practice should therefore be to use a minimalist approach when defining ontologies. While human language is complex, many matching requirements are met with a surprisingly shallow hierarchy of macros and variables.

Preconditions

So far we have discussed only static means of analyzing user input. Albeit flexible, both variable interpolation as well as macro extension happen when an instance of a PMLS Discourse Engine is created, or if its ontology gets reloaded. During the discourse, there is no dynamic code execution or context sensitive processing that these two methods could provide.

However, during a discourse it is perfectly normal to ask or answer to completely different topics with the same words/phrases. Any discourse system that is basically time invariant (does not account for context and history of the discourse) will fail.

The PMLS discourse engine provides with so called preconditions a turing-complete means to define conditions - and actions - to any part of the discourse, depending on any condition or context you might want to define. What does turing-complete mean? To put it simply, you have a fully fledged programming interface for every single matching rule/topic in your ontology.

With this, you can give different answers depending on the history of the discourse, current emulated mood of the discourse engine, already learned facts, current time or whatever might come to your mind.

Let us inspect a simple example of acquiring time:

```
[topic=q_time|0=get_timeinfo]
o = Die lokale Uhrzeit ist %0.
a = <<'EOA'
given ($capture{time}) {
  when (undef) {
    return $self->pre_pass();
  }
  when (m{morgen}xms) {
    my $tomorrow = $self->get_timeinfo(time() + 86400);
    return $self->pre_final("Morgen um diese Zeit ist es $tomorrow.\n");
  }
  when (m{gestern}xms) {
    my $yesterday = $self->get_timeinfo(time() - 86400);
    return $self->pre_final("Gestern um diese Zeit war es $yesterday.\n");
  }
}
return $self->pre_pass();
EOA
i = \A((uhr)?zeit|datum)\z
i = \Awieviel uhr
```

```
i = wie (spaet|frueh) ist es
i = welche[ns] (tag|datum)? haben wir(<?<time> heute| morgen| gestern gehabt)?
i = welche[ns] (tag|datum)? hatten wir(<?<time> gestern)?( gehabt)?
e = relaxed
```

A precondition is defined by the key 'a'⁷ and the value is Perl code. In the example above, we have used the non-interpolating heredoc 'EOA' contrary to an interpolating context (which would be "EOA"), to prevent any accidental interpolation of defined variables and the occurrence of some %varname% in the Perl code. Of course, if that would be your intention, it is perfectly ok to let variables interpolate into such a block of Perl code.

This Perl code is executed only if an input matching rule did match. In our case let's assume the user input would have been

```
pmls> say "this=Welchen haben wir heute?"
```

This would match the fourth input match definition, and therefore trigger the execution of the precondition code in this topic (q_time). As we have asked about today's time (heute), the named capture buffer `time` contains the string "heute".

The code execution starts and first thing it does, it inspects the content of the input capture buffer `time`. In our case it is "heute" and that will not match any of the `when`-clauses, so the `given`-block will be left undone and a `pre_pass`-value is returned. `pre_pass` means, that the precondition code passes execution back to the regular discourse engine code and exits. This regular code will get the string from the 'o' definition and output it. Before output, there will be one variable interpolation of the `%0` variable, which will be assigned the value from the `get_timeinfo` method (this returns the current date/time).

That was pretty much a "nothing done" for the precondition. Compare this to the case where the user input is

```
pmls> say "this=Welches Datum hatten wir gestern?"
```

In this case (fifth input match definition matches), the precondition block is entered, named capture buffer `time` has value "gestern" and this matches the third `when`-clause. The code in this clause simply fetches the current time via the `get_timeinfo` method and subtracts 86400 seconds from it, which is one day. The result is returned via a `pre_final` method which means, that the precondition code takes over control and the regular discourse engine code (which would be executed in case of a `pre_pass` return) is not executed.

6.2.7 Emotional Modelling

Emotional Modelling in the PMLS Discourse Engine allows you to provide the visual representation of the "Chatbot" with a look adequate to the preceding discourse. E.g. if the user did his best to "insult" or "humiliate" the chatbot, one would expect an appropriate reaction. In our product, this reaction is 3-fold:

⁷mnemonics: ante

- The visual representation will show - depending on previous conversation - signs of annoyance, anger, but in positive cases also empathy and pleasure.
- The reactions of the discourse engine - depending on your ontology - will be **different** under different emotional states.
- The emotional state will show a certain inertial behaviour. That is, if the bot is in good mood, it will take more (or worse) statements to make it “angry”. Also, once “angry”, it will behave less cooperative and so it will also take more time with a constructive conversation to make it friendly again.

6.2.8 References

Relevant commands for the Discourse Engine plugin are `bot`(C.2, pg. 181), `bot_add` (C.3, pg. 182), `bot_del` (C.4, pg. 183), `bots_list` (C.5, pg. 183) and `say` (C.67, pg. 204). Please look up the corresponding appendix references.

6.3 Machine Translation



another.

Machine translation, sometimes referred to by the abbreviation MT, is a sub-field of computational linguistics that investigates the use of computer software to translate text or speech from one natural language to

Level 0 This is a very basic form - hardly MT. Basically it is a simple, word-by-word first-fit translation. No morphosyntactical processing or any other sophisticated methods. Using this method is very cheap and its usage domain could be to possibly grasp the meaning of the original text.

Level 1 This mode adds morphological analysis and 2-word phrase lookahead functionality as well as a disambiguation based on statistical occurrence data. There is no syntactical reorder however. Using this method is quite cost efficient, and the results should be more than sufficient to grasp the meaning of the original text. This quality level should be used as minimum if a human-edit is requested.

Level 2 Adds syntax reorder and n-word phrase lookahead to the simple mode. This quality grade provides output comparable with most modern MT systems and could be used in cases where users can accept contemporary MT quality.

Level 3 Adds elaborated disambiguation methods and the processing of procedural words. In real-world texts, results are slightly better than the "Average" quality and could be very well used in intranets where providing multilingual information at lowest cost is necessary.

Level 4 Adds semantic inference, named entity recognition and anaphora resolution. If these terms mean nothing to you, just be aware, that this is the bleeding edge of today's MT technology and will deliver results hitherto unseen from MT systems. The results are of publishable quality for casual content and need only very few human intervention to become perfect.

Level 5 The system proactively searches the internet for relevant texts and dictionaries in addition to all its local features, to provide also good translations for uncovered areas. The system extracts informations from these sources and learns the relevant facts about the topic - thus improving its own knowledge base.

6.4 Information Extraction (IE)

In natural language processing, information extraction (IE) is a type of information retrieval whose goal is to automatically extract structured information, i.e. categorized and contextually and semantically well-defined data from a certain domain, from unstructured machine-readable documents.

Chapter 7

Advanced Usage

*It is a mistake to try to look too far ahead.
The chain of destiny can only be grasped one link at a time.*
Sir Winston Churchill

7.1 Batch Operation

You can use PMLS for automated or scripted (batch-)processing of voluminous data. Most important for this mode of operation is the `-cmd` parameter of the `client_std` PMLS client. Once you have started the PMLS server process you can use your shell scripts `make`¹ files or `makepp`² files to control PMLS via the `client_std -cmd` client.

```
bash> ./client_std -user=username -pass=secretpass  
          -cmd='ladd en de' -cmd='put text.txt' -cmd='li text.txt'
```

This will load the english and German lexica, upload a local file named “text.txt” to the PMLS server and perform a language identification on this file.

A more sophisticated approach to batch processing is the PMSE - the PetaMem Scripting Environment, which is discussed in section ??, pg. ??.

7.2 Remote Server Configuration

As mentioned earlier, there is no need for the PMLS server process and the client being on the same computer. Also, a PMLS server process may serve an arbitrary number of clients from different locations on different operating systems. On the installation medium, you will find several ready-to-run clients for various operating systems in the directory [packages](#). As of this writing, clients for OpenBSD 4.1, SUSE versions 9.2 and 10.1, Solaris 10, Ubuntu 7.04 and RedHat Enterprise Server 4:

¹see - <http://www.gnu.org/software/make/>

²an advanced version of make. See - <http://makepp.sourceforge.net/>

pmls_cliclient-0.51-OPENBSD4.1-i386.gz
pmls_cliclient-0.51-RHES4-i686.gz
pmls_cliclient-0.51-SOLARIS10-i386.gz
pmls_cliclient-0.51-SUSE10.1-x86_64.gz
pmls_cliclient-0.51-SUSE9.2-i586.gz
pmls_cliclient-0.51-UBUNTU7.04-i386.gz

There is also a source-only client in the package [client_std-0.51.zip](#) which you can install on other UNIX operating systems³.

³see [doc/admin/client_std/installation_instructions.txt](#) how to get your client also installed on Windows and Cygwin

Chapter 8

PMSE

*When you have a good script you're almost in more trouble
than when you have a terrible script.*
Robert Downey, Jr.

8.1 About the PetaMem Scripting Environment

The PMSE is a suite of programs and scripts that provide comprehensive functionality for processing large amounts of textual data - so called corpora.

In case you are familiar with the UNIX tools `sed`¹ and `AWK`², you will already know not only the alleged progenitors and inspiration for Perl, but also a large part of the concepts behind batch text processing.

In case you are not be familiar with these tools, never mind: The PMSE will provide you with powerful and concise text processing capabilities.

Let's say you have one (or more) of the following tasks to do:

- translate all of your intranet documentation from one or more source languages to several other target languages and periodically synchronize the translations.
- perform a grammar and spell check on thousands of documents in your company.
- retrieve specific information from internet sources, and to perform actions triggered by the semantic information retrieved (data mining/information retrieval)
- perform various conversions (format, encoding), apply filters, categorize texts etc. on textual data.

then PMSE is your tool and probably soon your friend.

¹see <http://en.wikipedia.org/wiki/Sed>

²see http://en.wikipedia.org/wiki/AWK_programming_language

8.1.1 Conceptual Overview

Image 8.1.1 on page 85 shows an abstract overview of the wide range of processes that can be achieved by PMSE. Starting at the text retrieval, the process-chain continues with statistical data-mining and ends with the display of information. Special position has the `P_ici` script, which can start parallel processing and make working with PMSE scripts highly effective.

8.1.2 Paths and Directory Structure

This section describes predeclared paths and presumed directory structure related to usage of PMSE. The paths are set in the environment by PetaMem configuration files.

PMSE - Root

The root of PMSE is set to be the `PMSE/active/` directory. It can be accessed in bash as:

```
$ cd $PMSE_ROOT
```

PMSE - Binary

The scripts described in the following section 8.1.3 are placed in `$PMSE_ROOT/bin` path which can be accessed from bash as a `$PMSE_BIN` variable.

```
$ cd $PMSE_BIN
```

Data - Library Structure

PMSE is designed to process texts in order to extract statistical data related to language. As we want PMSE to be a processor of multilingual data, we assume a big variability in sources of the data.

We adopted a specific name-space strategy to handle multilingual data and keep it synoptic. The resources are placed in `/data/library/` directory, which can be (in default setting) accessed in bash as

```
$ cd $PMCORP_ROOT
```

command. For information about conversion and manipulation of source files see `P_dmf` section 8.7.

8.1.3 PMSE Toolset Overview

Under the `PMSE_ROOT/bin` directory you will find several scripts that provide generic batch processing functionality and macros which alleviate the job of large scale text data processing. The naming scheme of the central PMSE scripts is `P_xxx`, where `xxx` is a mnemonic for a more verbatim explanation of the functionality of the specific command. In the same directory, a file `README.txt` will give you an overview of the scripts available: The most important scripts to become familiar with are the following:

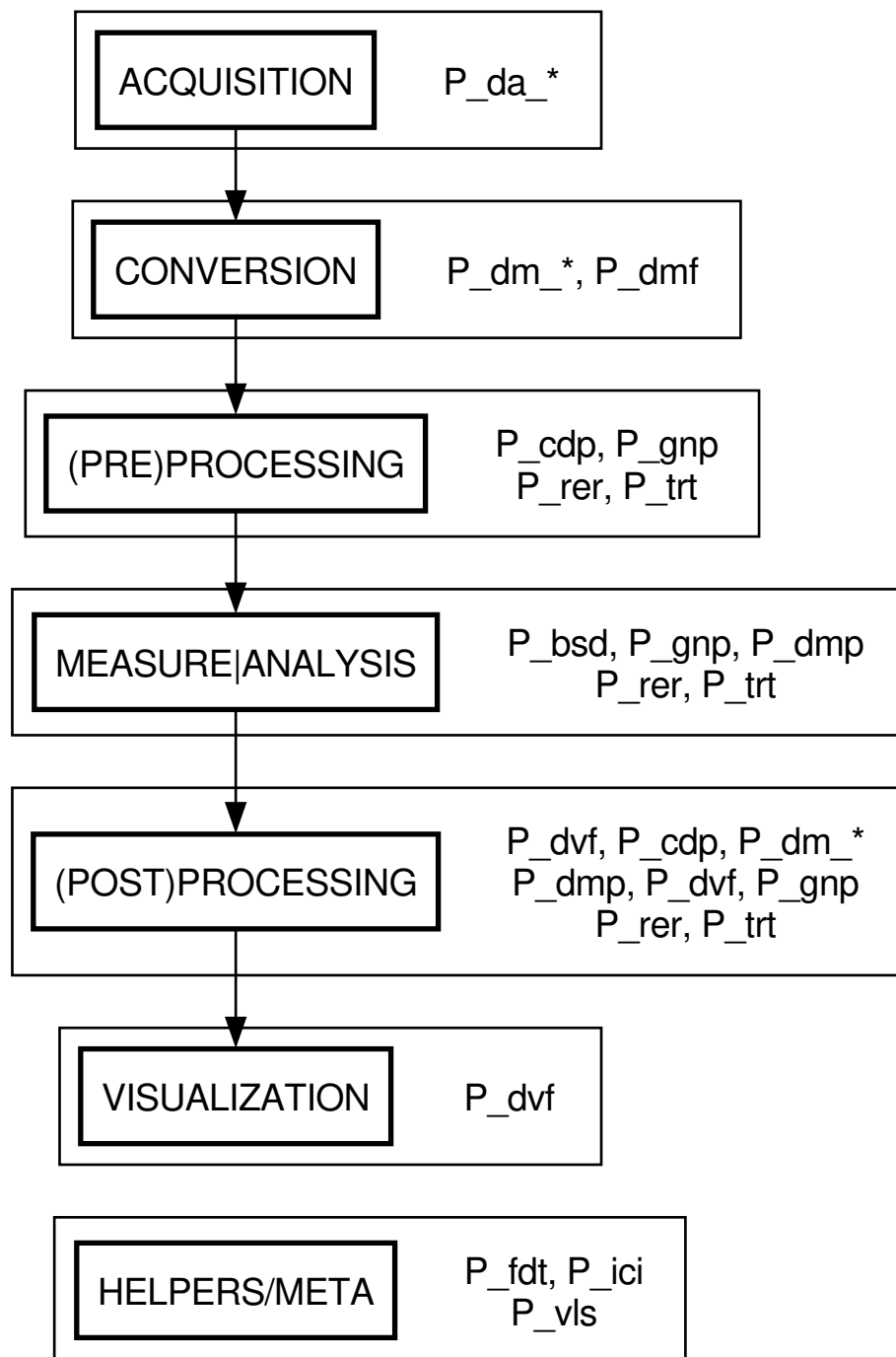


Figure 8.1: PMSE top level overview

P_bsd *Basic Statistical Data*

Get basic statistical data about a corpus

P_cop *Co-occurrences Processor*

Extract co-occurrences for given target from a list of bigrams

P_cct *Corpus Conversion Tool*

Convert corpora and tagsets

P_csp *Comprehensive Statistics Processor*

is the core statistical processing script in PMSE. It provides a wide range of statistical functions for text processing.

P_dmp *Distance Measures Processor*

Computes distance measures for N-gram pairs

P_dmf *Data Mining Framework*

Process files in numerous formats into a plain text automatically

P_dvf *Data Visualization Framework*

Dump and convert data of various formats (Storable, YAML, ...)

P_fdt *File Distribution Tool*

Distribute given files to a set of containers of given size

P_gnp *Generic N-grams Processor*

N-grams processing, calculates various N-grams measurements, creating of contingency tables

P_help *PMSE Helper*

General help for PMSE environment

P_ici *Intelligent Command Iterator*

Script for iterating and parallelization of other scripts

P_rer *Regular Expression Replacer*

The regular expression replacement engine script

P_trt *Text Repair Tool*

Manipulate with a text on more abstract level

P_vls *Variable Length Splitter*

Specify a part of text (part of word list) and cut it out

Every script documents its commandline options via `<cmd> --help`. Also, as all of the scripts are an integral part of the PMSE infrastructure, they share several common options you will see in the help texts:

This is a PMSE script. Generic PMSE options are:

Options:

`--cpu <n>`

set the number of CPUs manually. This is recognized automatically and influences the number of parallel started processes (if possible). With this you can override the autodetect.

`--debug <n>`

Will enable the printout of debug messages. The verbosity threshold for the debug messages is handled in the same as for the regular messages. (see "`--report`").

```
--dry
    dry run facility. Do not actually execute the commands

--help
    print the script specific help and the general PMSE help

--info
    information about the current environment, such as arguments,
    detected values, etc. When this parameter is detected the
    information delivered is current i.e. parameters AFTER this
    parameter are not considered yet. If you want the final
    information for all parameters, place this parameter last
    on the command line

--report <n>
    by default all scripts have no output. If you'd like to know more
    about what's going on, set this option. Setting <n> higher will
    produce more verbose output.

--version
    prints the version number of the script, and of PMSE, and exits.
```

Please note that command line parameter processing is done via the Perl module `Getopt::Long`³ which implies some advanced and tolerant handling. For instance, it doesn't matter whether you provide `-h` or `-help` or `--help` as long as your option name is unambiguous.

There exists also option specific help available for options that require one or more input parameters from a list. This so called self-documenting help may be invoked with question-mark in the position of a argument for given option. In POD, it is marked as:

```
--option <type|?>
```

type may be one of following:

```
A
B
C
...
```

All the most important scripts can be run in interactive mode when `--iact` option is used. In which case the user will be asked to complete options interactively. After execution, he can find the effective CLI in `pmse_env`.

The next sections will cover all PMSE scripts. The structure of the documentation is the same for every script: An introductory and general text, the command reference and examples.

While the examples are specific to the tool described, they often cover useful information about generic options and their usage, pitfalls, as well as tips and tricks. It is therefore

³see <http://search.cpan.org/perldoc?Getopt::Long>

advisable to look through all of them.

After this, the PMSE Tutorial and PMSE Cookbook chapters cover bigger use cases for PMSE.

8.2 P_bsd: Basic Statistical Data

P_bsd provides a basic statistical overview of a text/corpus. It is analogue to the Unix-like program *wc*, but it provides more options and more detailed output.

You can adjust the token definition by the `--delimiter` and `--ifilter` options. See section 8.5 for detailed usage of both of these options.

This script comes in handy, when you need to compare files without scanning them line by line. E.g.:The average number of characters per token or the type-token ratio could help you to detect strangeness in your text.

```
$ P_bsd -?
```

8.2.1 Reference

PMSE Basic Statistical Data Basic Statistical Data script gives an essential overview about the corpus.

USAGE P_bsd [options] file

At least one file (corpus) must be given.

The result is a Perl data structure stored in Storable format in `overview` file in directory specified by `--out` option.

OPTIONS

-delimiter <regex>

The delimiter has the form of Perl regular expression. It enables the tokenizer to dissect text into discrete tokens. If the user doesn't set his own value, the default from the PMLIB tokenizer is taken.

-ifilter [<type>=<regex>|?]*

This option may be provided multiple times (with different content for `type` of course) to define various filters, that are inserted at specific places in the data stream during processing. Valid values for `ifilter` <type> are:

```
+token    tokenizing step: matching tokens will pass
-token    tokenizing step: matching tokens will be blocked
```

<regex> may be any regular expression. Please take care to quote the whole expression, to prevent the shell modifying it. For example: '`<filter>=<regex>`'.

-in <filename>

Input file.

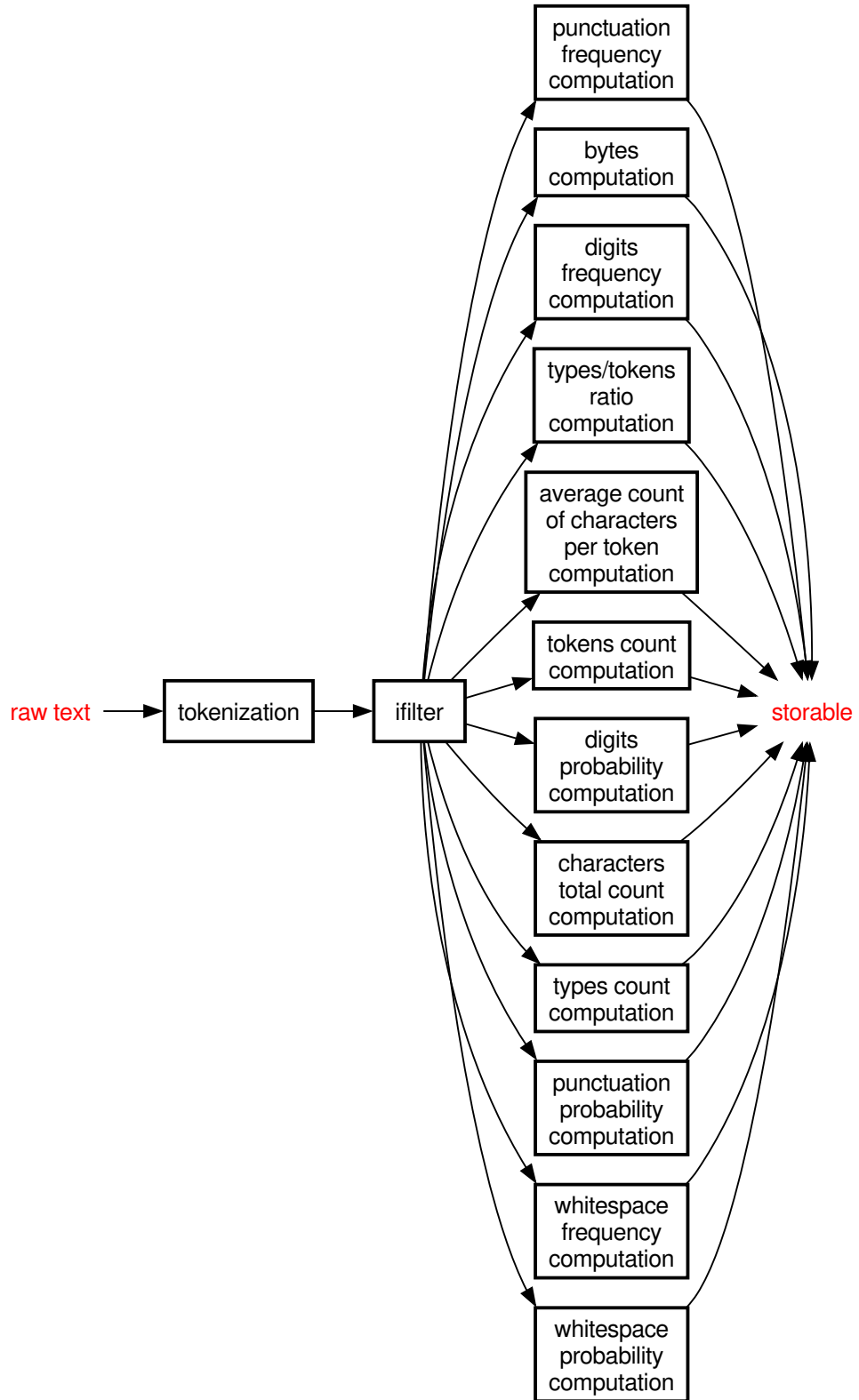


Figure 8.2: P_bsd schematic overview

-out <directoryname>

Defines the output directory.

8.2.2 Examples

Basic information retrieval

```
$ P_bsd --out STDOUT --in text.txt
```

Tokenization will be provided by the default PMLIB tokenizer. The script will produce a hash structure with information about text specified in the `--in` option. This hash will be printed on STDOUT:

```
{
  'distribution_punctuation_freq' => 10055,
  'bytes'                        => 297858,
  'distribution_number_freq'     => 264,
  'types_tokens_ratio'          => '0.0479324517972524',
  'average_number_of_token_chars' => '2.55586541844361',
  'tokens_count'                => 116539,
  'distribution_number_prob'     => '0.000886328384666519',
  'chars'                       => 297858,
  'types_count'                 => 5586,
  'distribution_punctuation_prob' => '0.0337576966205373',
  'distribution_whitespace_freq' => 60847,
  'distribution_whitespace_prob' => '0.204281906143196'
}
(waiting for ENTER)
```

The abbreviations mentioned above mean:

- **average_number_of_token_chars** average number of characters per token
- **bytes** length in bytes
- **chars** length in characters
- **distribution_number_freq** frequency of digits in the text
- **distribution_number_prob** probability of digit occurrence
- **distribution_punctuation_freq** frequency of punctuation marks
- **distribution_punctuation_prob** probability of punctuation marks occurrence
- **distribution_whitespace_freq** frequency of whitespace
- **distribution_whitespace_prob** probability of whitespace occurrence
- **tokens_count** count of the tokens in the given text
- **types_count** count of the the types in the given text
- **types_tokens_ratio** type-token ratio

8.3 P_cct: Corpus Conversion Tool

P_cct provides conversion of corpora from source format into Perl data structure. It also converts tagsets into Pmts - the PetaMem Tag Set. Conversion mechanism is implemented in PMSE::Corpus module (and all subsequent modules in given namespace). Conversion of tagsets takes place in PMLIB::Tag namespace. Architecture of corpus and tagset conversion is completely modular and may be extended easily.

```
$ P_cct -?
```

8.3.1 Reference

PMSE Corpus Converter Tool

USAGE P_cct [options]

P_cct converts corpus from original to specified form.

OPTIONS

-convert <void|direction|?>

Execute conversion. May be called without parameter, with specific direction and with ?. If no parameter was specified, default conversion (original tagset 2 pmts) will be used. List of available conversions for given corpus may be listed with ?.

-corpus <corpus name|?>

Specify name of corpus you want to convert. Currently are supported:

```
BNC
Penn
OANC
WikiCorpus
CNK
PDT
```

The complete list of corpora may be listed with ? as a parameter.

-in <filename|directory>

Input files.

Also a directory may be specified, in that case specify also mask.

-mask <regex>

Positive filter for files. Default is:

```
\.txt\z
```

All files in input directory matching this regexp will be processed.

-out <directoryname> or <STDOUT>

Defines the output directory. If not specified, Data will be stored in CWD. Output file will have the same name as original file.

8.3.2 Examples**List available conversions**

```
$ P_cct --corpus ?
```

Basic usage

```
$ P_cct --in OANC --out STDOUT --convert
```

Will find all files in OANC directory and will convert them into internal data structure. Penn-like tagset will be converted to Pmts.

Data conversion

Consider a conversion of OANC⁴ corpus from previous example. Source files of OANC are stored in text format. A sample of such source is displayed below:⁵

```
"/' ' But/CC you/PRP 're/VBP the/DT only/JJ
person/NN who/WP reads/VBZ it/PRP ./.' "' ' That/DT may/MD be/VB
true/JJ ./.' But/CC 15/CD ,/, 000/CD copies/NNS were/VBD printed/VBN ./.'.
```

Resulting data structure will look like this:

```
[
  [
    [",      Oq  ],
    [That,   Dg--s],
    [may,    Voip ],
    [be,     Van  ],
    [true,   Afp  ],
    [.,     Ot   ],
  ],
  [
    [But,    Cc  ],
    [15,     Mc  ],
    [.,     Oy  ],
    [000,    Mc  ],
    [copies, Nc-p],
    [were,   Vais],
    [printed, Vmps],
    [.,     Ot  ],
  ],
],
```

⁴<http://www.americannationalcorpus.org/>

⁵Actually it is a segment of last paragraph of file *ArticleIP_2572.txt*.

],

First level of braces encloses whole converted file, second encloses sentences and the lowest level holds 2-tuples - word and its tag.

8.4 P_cop: Co-occurrence Processor

P_cop extracts co-occurrences for given target from a list of bigrams. The list must be stored in a Storable format. See section 8.17.6 for detailed instructions and explanation of the concept of co-occurrences.

Level of co-occurrences is specified via the `--level` option.

The target (a word with which co-occurs other words) may be specified as a regular expression or a literal. The output is an PMSE object, that can be easily visualized by P_dvf.

PMSE uses the GraphViz software⁶ to display the relation between the target and co-occurring words.

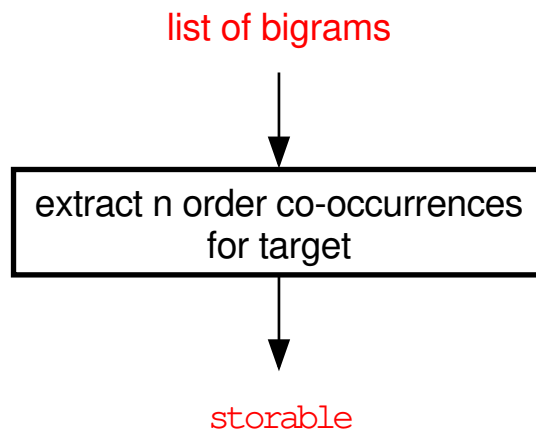


Figure 8.3: P_cop schematic overview

```
$ P_cop -?
```

8.4.1 Reference

PMSE Co-occurrence Processor

USAGE P_cop [OPTIONS]

P_cop extracts co-occurrences from bigrams.

⁶<http://www.graphviz.org/>

OPTIONS**-delimiter <regexp>**

Defines delimiter of tokens in the n-gram. Default is white-space.

-in <filename>

Defines the input file.

It must be a Storable file. Input file must contain bigrams in the format:

```
<bigram> => <value>
```

-level <n>

Level of co-occurrences. Assume this example:

```
target:  A
lvl 1:  A -> B (a co-occurs with B)
lvl 2:  B -> C
lvl 3:  C -> D
```

Co-occurrence A -> D is of order 3. The "deeper" level you go the more co-occurrences you get - and result may become messy. Filtering of input bigrams (and tokens) with high frequency may help here.

Default value of level is 3.

-out <filename>

Defines the name of the output file. The output is a PMSE::Visualize object stored in Storable format.

-query <type>=<target>|?

Type may be:

```
literal
regex
```

It is necessary to specify a word / expression (target) for that we want to extract co-occurring words. The target may be specified as a literal string or as a regular expression. When a literal string is specified, co-occurrences will be extracted exactly for this string. Regular expression may match more words.

8.4.2 Examples

```
$ P_cop --in bigrams.sbl --out coocs --query 'regex=.+ship' --level 2
```

8.5 P_csp: Comprehensive Statistics Processor

P_csp is a tool designed for extraction of tokens and their statistical characteristics, like basic count, frequency or probability. It is also capable to compute histograms of values for given set of tokens.

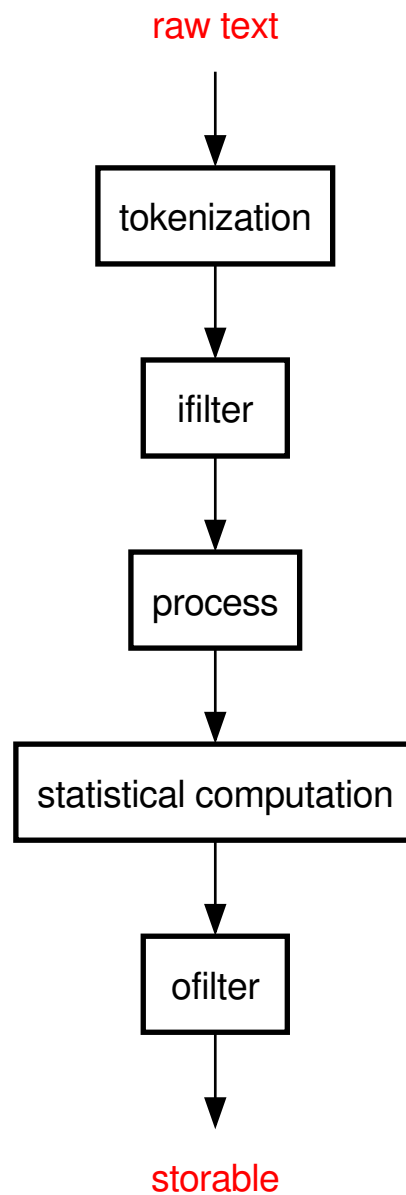


Figure 8.4: P_csp schematic overview

```
$ P_csp -?
```

8.5.1 Reference

PMSE Comprehensive Statistics Processor

USAGE P_csp [options]

P_csp is the core statistical processing script in PMSE. It provides a wide range of statistical functions for text processing.

The process workflow of P_csp is as follows:

```
text(s)-> tokenization -> statistics -> metrics -> output
```

Throughout this processing chain, various hooks are deployed and support extensive data mangling (filters, transformations etc.).

OPTIONS

-action [**<action>** | **?**]*

You can give one or more named actions as a space-separated list to define what processing the script shall perform on the input files. Valid values for **<action>** are:

```
utcount    count unique tokens
utfreq     compute the token frequencies
utprob     compute the token probabilities
```

-bulk **<file>**

With option **-bulk** you can define an INI file, which can be used for histogram / hash filtering and tokens processing. The INI file has a structure of '[section]' and `name=value`, heredoc style can also be used. 'Section' is name of the procedure (process / ofilter), 'name' is the name of the hook (the same as in `--process` and `--ofilter`). 'Value' stores code, which will be executed on the result of the action.

-delimiter **<regex>**

Delimiter has the form of a Perl regular expression. It enables the tokenizer to dissect text into discrete tokens. If the user doesn't set his own value, the default delimiter from the PMLIB tokenizer is taken.

-histogram [**<action>**=**<file>** | **?**]

We can count a distribution (histogram) for the result of each 'action'. Histogram is stored in the file 'file'. Actions could be:

```
utcount    histogram stores distribution of tokens occurrences
utfreq     histogram stores distribution of tokens frequencies
utprob     histogram stores distribution of tokens probabilities
```

'File' is in Perl Storable format.

-ifilter [**<type>**=**<regex>** | **?**]*

This option may be provided multiple times (with different content for `type` of course) to define various filters, that are inserted at specific places during data stream processing. Valid values for `ifilter <type>` are:

```
+token     tokenizing step: matching tokens will pass
-token     tokenizing step: matching tokens will be blocked
+utcount   unique tokens counting step: matching tokens will pass
-utcount   unique tokens counting step:
```

```

        matching tokens will be blocked
+utfreq  unique tokens frequencies step: matching tokens will pass
-utfreq  unique tokens frequencies step:
        matching tokens will be blocked
+utprob  unique tokens probabilities step: matching tokens will pass
-utprob  unique tokens probabilities step:
        matching tokens will be blocked

```

<regex> may be any regular expression. Please take care to quote the whole expression, like '<filter>=<regex>', to prevent the shell modifying it.

-in <filename>

Input file.

-ofilter [<hook>=<code> | ?]

Ofilter can be used for result of each action, which is always a hash, thus with <code> we can affect both keys and values. Hook denotes the specific part of the process where the ofilter is applied. Hook can be:

```

utcount  filter result of action utcount
utfreq   filter result of action utfreq
utprob   filter result of action utprob
_hist    filter result before histogram is made
histogram filter hash with histogram values
        (note: keys and values are both numeric)

```

Code could have the form of \$key =~ m{.*}xmsg or \$value >=< number. You can also insert the name of an INI file via the --bulk option.

-out <directoryname>

Defines the output directory.

-process [<hook>=<subst> | ?]*

<subst> is a Perl substitution operator: s{pattern}{replacement}flags, whereas "pattern" is a regular expression, "replacement" may be a string or even Perl code if the "e" flag is given. Further info see e.g. <http://perldoc.perl.org>

<hook> may be one of the following:

```

_utcount  pre  'utcount' hook
 utcount_ post 'utcount' pre-set building
_utfreq   pre  'utfreq' set building
 utfreq_  post 'utfreq' set building
_utprob   pre  'utprob' set building
 utprob_  post 'utprob' set building

```

8.5.2 Examples

Basic usage

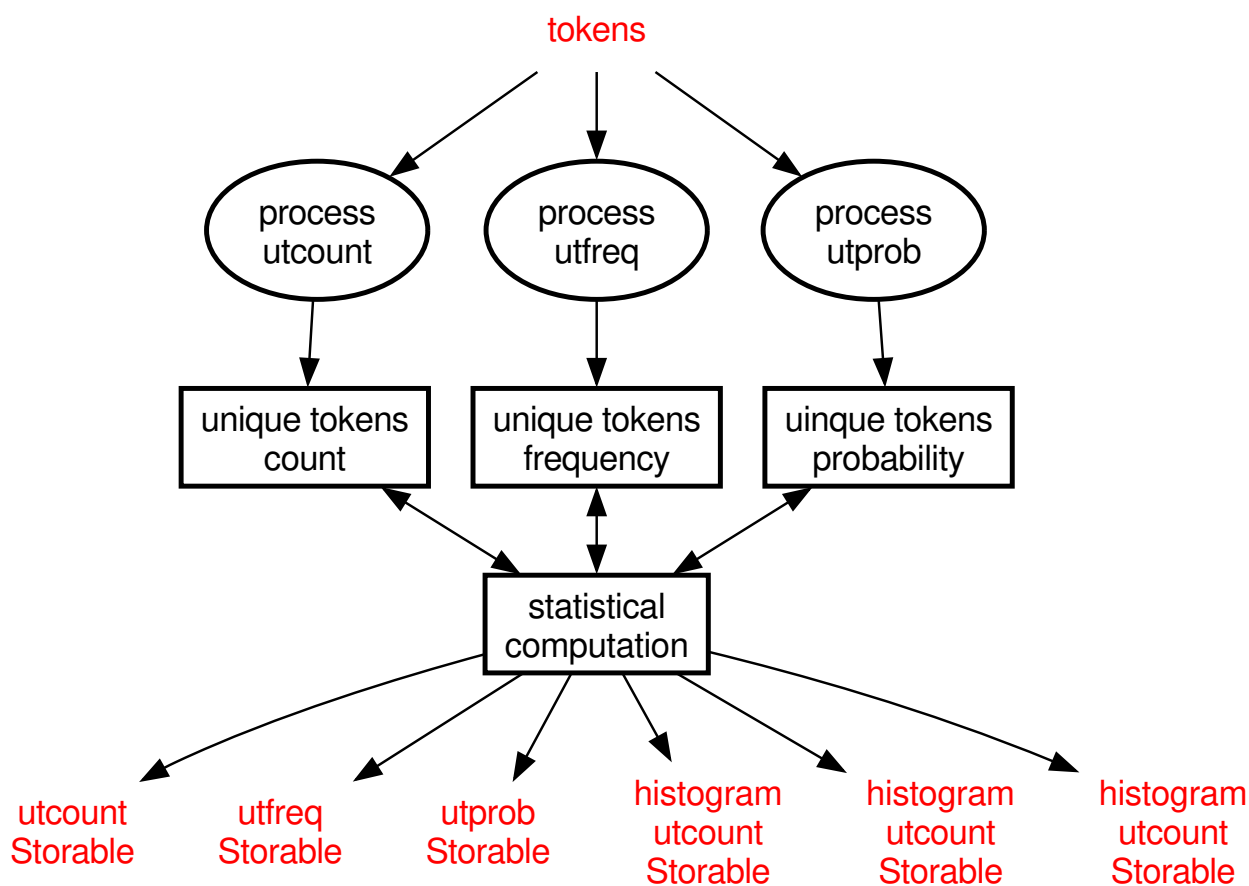


Figure 8.5: Overview of the statistical processes

```
$ P_csp --action utcount --out tokens --in corpus.txt
```

will count all unique tokens (types) from the "corpus.txt" file and the output will be saved in the directory called "tokens". The name of the output file is created from the combination of the --action function, or functions, which were called on the commandline. In this case, the path to the output file will be: "tokens/utcount".

To familiarize yourself with the dependencies of various processing options, please see figure 8.5.2.

Get frequencies

```
$ P_csp --action utfreq --ifilter '+token=\A\p{Alpha}+\p{Digit}* \z' \
> --insort alpha --out cfreq --in corpus.txt
```

will calculate the frequencies of alpha-numeric character tokens from the input file "corpus.txt". The contents of the output file will be sorted by ascending alphabetical order. The

output, in this case, will be stored in the "cfreq/utfreq" file.

Unique count, frequency, probability, lower casing

```
$ P_csp --action utcount utfreq utprob --ifilter '+token=\D+' \
> --delimiter '\s+' --process 's{\A(.+)\z}{lc($1)}xmse' \
> --out corpus_info --in corpus.txt
```

will count unique tokens, their frequency and probability. Tokens (in the original text) are split by whitespace. (--delimiter).

Only non-digits tokens will pass (--ifilter). All tokens will be lowercase (--process). The 3 output files: utcount, utfreq, utprob will be stored in the corpus_info directory.

The output of P_csp is a Perl data structure, further processing can be done by utilizing the P_dvf script.

How to create a bulk file?

The bulk file loaded into P_csp needs to comply with the INI file format: it must have the [section] and 'name=value' structure. Each closing heredoc terminator END must be followed by a newline.

```
[process]
_utcount=s{\d+}{NUMBER}xms
_utfreq=s{(A-Z)}{lc$1}xmse
utprob_=s{\bY.+ \b}{ }xmsi
[ofilter]
_ofilter=<<END
$key =~ m{\pP+}xmsg
$key !~ m{\w}xmsg && $value < 20
END
ofilter_=<<END
$value = 25|32|33
$value > 1500
END
```

Each '[section]' denotes the type of change, where 'name' is a hook, as in --process or --ofilter options. The 'value' contains code, that will be eval'd during processing. You can use the heredoc style to include several lines of code and transformations. A bulk file could be used for multiple filtering and tokens processing.

Creating a histogram

```
$ P_csp --action utcount --out c_utcount \
> --histogram 'utcount=histogram' --in corpus.txt
```

Will calculate the histogram (distribution of occurrences for tokens), the histogram output will be stored in the file 'histogram'.

Creating a histogram and using ofilter

```
$ P_csp --action utcount --out c_utcount \  
> --histogram 'utcount=histogram' --ofilter '_hist=$value > 20' --in corpus.txt
```

This is the same as the example above except that tokens which occur more than 20 times will be deleted. Deleted tokens won't be counted in the histogram. You can also use the `--bulk` option to apply multiple conditions, e.g.: you can delete various words, word forms and tokens represented by classes of regular expressions etc.

8.5.3 Q&A

Something is wrong while P_csp runs on commandline.

If you see this error, it is most probably caused by erroneous shell expansion. You will need to quote your regex `\b(\w)+\pP` or special characters `*`. Or you can escape them, by changing `*` to `*`.

Pay especial attention when using the following characters: `[]<>()*|`

Output file is not readable

The output of `P_csp` is saved in Storable format, which may need to be converted before being read by another script, perhaps using `P_dvf` or some other converter.

8.6 P_daf: Data Acquisition Framework

`P_daf` is a framework for acquisition of various data. The rules (source, destination) for the acquisition are not dependent on `P_daf`. They are specified in a separate INI file.

That allows you to write your own INI files and make your data acquisition automated.

INI files are placed in a `$PMSE_ROOT/cfg/daf.d.` directory. But you can specify alternative path by the `--ini` option.

```
$ P_daf -?
```

8.6.1 Reference

PMSE Data Acquisition Framework

USAGE `P_daf [OPTIONS]`

`P_daf` downloads data from the web.

OPTIONS

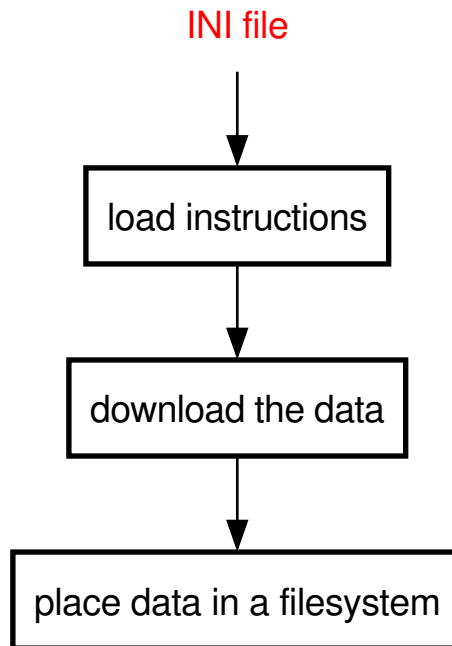


Figure 8.6: P_daf schematic overview

-fetch [<PROJECT> | ?]

Specify project(s) to fetch. ? will give you a list of available projects. If you store INI files in other than default directories, you should specify them in --ini option.

-ini [<DIR>,<FILE>]

Specify path(s) to INI file(s) or dir(s) which contain them. default is:

```
$PMSE_ROOT/cfg/daf.d
```

8.6.2 How to Write an INI File

First, let's take a look on a real one. The following INI file was used in PMSE to get list of geographic names:

```
[global]
lastfetch =
interval  = 6 months
name      = geonames

[geonames]
threads = 1
URL     = http://download.geonames.org/export/dump
match   = (?<file>(allCountries|alternateNames).zip)
get     = "%URL%/$file"
store   = "$ENV{PMSE_ROOT}/m/u/l/original/geonames/$file"
```

The INI file has two sections: *global* and *geonames*.⁷ Three parameters take place in the *global* section. *Lastfetch* may contain the date of the last download; this parameter is obligatory. *Interval* is the time period between downloads. *Name* is the name of the project, that you call from P_daf.

The section *geonames* contains several parameters:

threads specifies the number of processes

URL a web address containing a hyperlink(s) to the data

match a regular expression that match the hyperlink(s) aiming on the data

get specifies the target (matched hyperlink) to download

store specifies the place in a filesystem, where to store the data

As you can see by the *get* parameter, templating can be used. The value of a previously defined parameter is accessed as a %<name of parameter>% template.

The \$file variable is specified as a named backreference (captured group). in *match* parameter.⁸

8.6.3 Private Data and Personal INI

Some of your projects may request login. In that case we recommend you to add 3 lines to the global section of the INI:

```
login    = 1      # this section will tell P_daf
                # that login info is required
password = XYZZ  # here you will store your password
username = NN    # and here username
```

Note: if you will specify `login = 1` and won't specify password or username, P_daf will skip your INI and give you some warning.

In the case you don't want to store your personal credentials in common directory, you may create a directory containing your personal INI in your home.⁹ Even if there exists an INI of the same name in the common directory, P_daf will prefer yours. You have just to add your private directory to the `--ini` option:

```
# look for INIs in private and default directory
P_daf --fetch <private.ini> --ini <def.path> , <pers.path>
```

8.6.4 Extended INI File

The following INI file¹⁰ shows how to fetch the newest data from a web site offering multiple targets sorted by a date. The base URL is an index of files to download.

⁷The section name is always enclosed by square brackets.

⁸<http://perldoc.perl.org/perlretut.html#Named-backreferences>

⁹P_daf checks all paths to INIs and if there are duplicate INI names, it will prefer path containing your home address.

¹⁰The URL address in the BASE (currently in 2 lines) line must be placed on one line. We broke the line because of readability.

```

[global]
lastfetch =
interval   = 6 months
name       = openstreetmap

[whole_planet]
BASE = http://ftp5.gwdg.de/pub/misc/openstreetmap/planet.openstreetmap.org/
      planet/2013/
url   = %BASE%
prehook_start =<<PREHOOK_START
    our $newest = 0;

    # 1. do we already have whole planet map? _NO_? then go on
    if (!glob("$ENV{PMCORP_ROOT}/m/u/l/original/openstreetmap/planet*")){
        # here we have to find the newest date
        while ($source =~ m{>(?!<file>planet-(?!<date>\d{6})\<}xmsg){
            if ($+{date} > $newest){
                $newest = $+{date};
            }
        }
    }
PREHOOK_START
match   = \A # exactly one match
get     = $newest ? "%BASE%/planet-$newest.osm.bz2" : undef
store   = "$ENV{PMCORP_ROOT}/m/u/l/original/openstreetmap/$file"

```

The `prehook_start` parameter contains a Perl code (in the heredoc format) which checks the input (BASE) file (web site) and searches for the latest date.

Hyperlinks aiming to the target files are matched with the

```
m{>(?!<file>planet-(?!<date>\d{6})\<}xmsg
```

regexp. We have a named backreference called `$file`, which holds the name of each target. A date of origin is a part of the name. As we want to fetch only the newest file, we make a simple comparison of all dates in the *while* loop. The latest date is assigned to the `$newest` variable.

The variable called `$source` holds the content of the BASE.

The *match* section contains a `\A` anchor, that causes only one match on the given web site.

The *get* parameter contains a condition we can read as: "do we have a defined variable `$newest` ? YES - fetch the newest file, NO - do nothing".

8.6.5 Hooks

In previous section, we have mentioned `prehook_start` parameter. The concept of hook is to provide a 'raw' Perl code that will be executed in a specific place (time) of the process of acquisition of the data. There are available four hooks, what makes the DAF config file

pretty configurable. There exist 2 pre-hooks and 2 post-hooks. They are called:

```
prehook_start # process BASE file -before spec. targets
prehook       # before a specific target is downloaded

posthook      # after a specific target is downloaded
posthook_final # after all downloads are completed
```

The whole process is described on figure 8.6.5. Phases of the data acquisition are visualized as box-like nodes. Red labels stand for hooks. *Get* and *store* parameters are similar to hooks, because their content may be modified by a Perl code.

With *get* you may modify the specific download link, with *store* the path, where do you want to store the data.

8.6.6 Examples

```
$ P_daf --fetch geonames
```

8.7 P_dmf: Data Mining Framework

The *P_dmf* tool allows to convert numerous input formats to plain text. It has also some extended features like Wikimedia projects processing. *P_dmf* is designed to work with specific file structure, which is described below.

```
$ P_dmf -?
```

8.7.1 Reference

SYNOPSIS *P_dmf* [options]

OPTIONS

-base <base path>

Base path of your corpus data. Default is '/data/library'.

-conv <all|?>

This option will list available converters. The resulting list differs according to input option:

```
all  all available modules in PMLIB
?    converters installed in the system
```

-in <file|glob>+

Defines the input source file or glob. Option can be given multiple times for multiple input source files. If you define a glob, please be aware to put it in single quotes to prevent shell expansion.

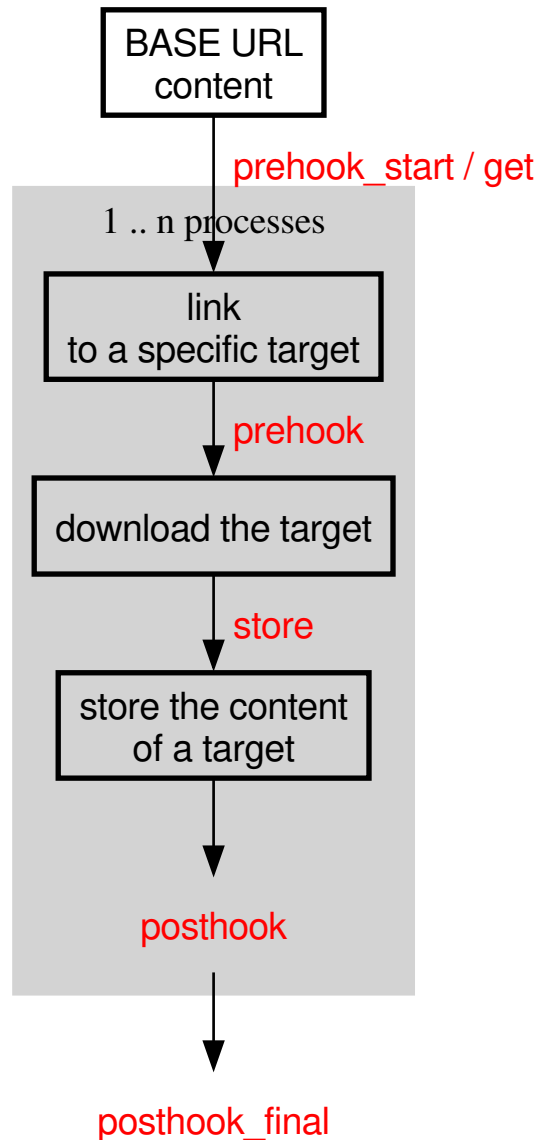


Figure 8.7: P_dmf processes overview

For this file you could also provide config file `.dmf.info`. You should put this file into one of parent dir. The closest one is applied.

-out <dir>

In case only one input dir is specified, P_dmf can store results into specified directory.

8.7.2 File Structure

Consider a situation, when the amount of the textual documents is dynamically changing. In the filesystem exist old, yet processed documents and also come new - in various formats, encoding etc. PMSE adopted a specific strategy, how to handle this.

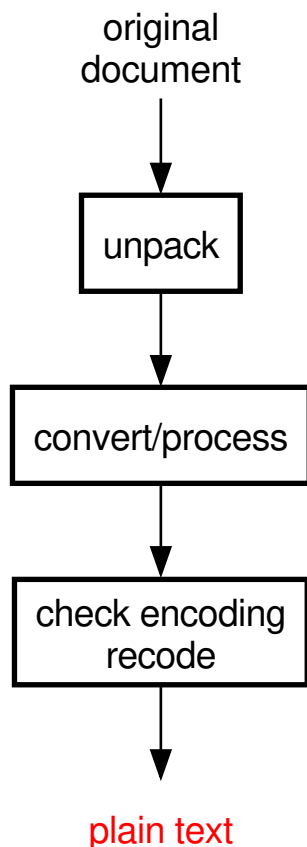


Figure 8.8: P_dmf schematic overview

PMSE uses by default the path `/data/library/` as document root to store any processed documents. This can be changed by the environment variable `PMCORP_ROOT`.

The environment of PMSE is extensively multilingual, therefore it offers a very generic way to store data for numerous languages under this document root. Each language is represented as a directory trie¹¹. The name of the trie is derived from the appropriate iso-639-3¹² code of the language.

The path for documents in English is therefore:

```
/data/library/e/n/g/
```

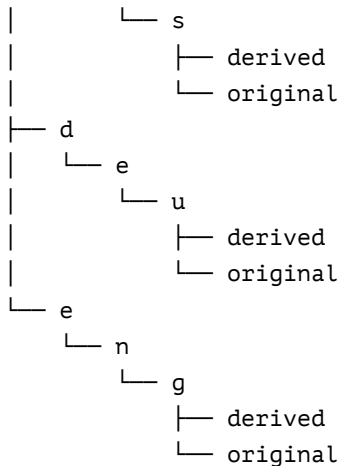
Original documents (incoming, raw data) are meant to be stored in a subdirectory called **original** and processed documents in a subdirectory **derived**. The resulting skeleton structure looks like this

```

data
├── library
│   ├── c
│   └── e
  
```

¹¹<http://en.wikipedia.org/wiki/Trie>

¹²http://en.wikipedia.org/wiki/ISO_639-3



Where we have only presented the Czech (ces), German, (deu) and English (eng) tries. In productive environments, PMSE can handle all of the 7000+ languages defined in iso639-3 this way.

If your environment has to cope with lots of languages, in order to be able to quickly visit the respective language on the command line, there is a shell alias `go` defined. This allows you to do

```
$ go ell
```

And you end up in the directory `/data/library/e/l/l/original/` (Greek original data).

In the original directories you can place your raw data files as you desire. because the derived directory has to maintain a quasi-mirror image of the original data structure, its stucture is slightly more complex, because one source file in *original* may correspond to several target files in *derived* (think e.g. about .zip archives in the original dir).

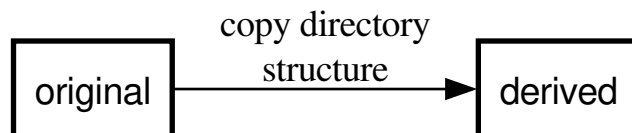


Figure 8.9: P_dmf directory structure

P_dmf will create similar structure in *derived* directory, but also will (eventually after several steps) convert the original document to a plain text encoded in UTF-8. The resulting text is placed in a directory called **lvl.last**.

Assume we have in the german original directory several yearly archives of PDF texts

```
$ ls /data/library/d/e/u/original/journal
```

```
2004.tbz2 2005.tbz2 2006.tbz2 2007.tbz2 2008.tbz2 2009.tbz2
2010.tbz2 2011.tbz2 2012.tbz2 2013.tbz2 2014.tbz2
```

where every of these archives contains several PDF files:

```
$ tar txf 2006.tbz2
```

```
2006/
2006/January.pdf
2006/February.pdf
2006/March.pdf
...
```

Then, after this script has processed these files, we will end up in the derived directory structure with something like this:

```
├─ 2004.tbz2
│  └─ lvl.1
│     └─ 2004.tbz2
│        └─ 2004.tar.bz2
│  └─ lvl.2
│     └─ 2004.tbz2
│        └─ 2004.tar.bz2
│           └─ 2004.tar.bz2
│              └─ 2004.tar
│                 └─ 2004.tar.gz
│  └─ lvl.3
│     └─ 2004.tbz2
│        └─ 2004.tar.bz2
│           └─ 2004.tar.bz2
│              └─ 2004.tar
│                 └─ 2004
│                    └─ January.pdf
│                       └─ February.pdf
│                          └─ March.pdf
│  ...
│  └─ lvl.4
│     └─ 2004.tbz2
│        └─ 2004.tar.bz2
│           └─ 2004.tar.bz2
│              └─ 2004.tar
│                 └─ 2004
│                    └─ January.pdf
│                       └─ January.txt
│                          └─ February.pdf
│                             └─ February.txt
│                                └─ March.pdf
│                                   └─ March.txt
│  ...
```

These January.txt, February.txt, March.txt files are what we were looking for.

No matter how many levels of conversion are required to end up with the desired target

text¹³, there will always be a `lvl.last` directory containing symbolic links to the final texts. Which means that if you are not interested in the intermediary data created during the conversion process, you can go to or point further processing tools to this directory.

8.7.3 Input Formats

P_dmf is able to handle several formats to convert input data into txt:

compressed files	7z ace ar arc arj bz2 cab gz lha lzma lzx pbz pbz2 pet rar rpm rz sea shar sit tar tar.bz2 tar.gz tar.rz xar xz zip zoo
doc files	doc docx fodt chm htm html htmlz odt ott rst rtf snb stw sxw troff txt uot wpd xml
pdf related	djvu dvi pdf ps tex texti
ebooks	azw epub fb2 lit lrf mobi pdb tcr txtz
other formats	cpio png

And possibly others. The real value add to P_dmf is its inferential capability to apply a chain of conversions to achieve a result even if your input data are nested archives and/or formats that need intermediary processing until a UTF-8 encoded text can be obtained.

8.7.4 Dependencies

P_dmf uses several available external converters. The script will warn you, if some of these auxiliary tools are missing on your system.

If you want to know the supported or available converters on your system, use the `--conv` option. If you want to know which are missing on your system, call

```
$ PM_CONVERTOR_WARNINGS=1 P_dmf -?
```

This will list supported, but not installed converters:

```
For support of arc2_ conversion install a tool used by PMLIB...
For support of arj2_ conversion install a tool used by PMLIB...
For support of chm2pdf For support of arc2_ conversion ...
For support of arj2_ conversion install a tool used by ...
For support of chm2pdf conversion install a tool used ...
```

Missing dependencies may be also tested (and installed) by `pm_install` script which is placed at `/opt/PetaMem/bin/active`. Just call

```
$ perl pm_install -testonly
```

and the installation process will be driven automatically.

¹³if possible at all, as you may have placed e.g. photos in the original directory

8.7.5 Input Texts, Encoding

P_dmf checks files for encoding, but it is only a heuristic guess. When the encoding is not recognized, text is removed from a processing queue. Although P_dmf is able to change encoding and to "repair" the text¹⁴, these functions are limited.

If the pre-defined conversion methods do not fit the requirements on the file processing, it is possible to specify a configuration file with a complete set of instructions - including a call of an external application. See section 8.7.6 below.

8.7.6 Wikimedia Processing

Wikimedia files are fetched by P_daf (see 8.6) in PMSE as `xml.bz2` dumps. The Wiki file must have a specific name containing a date in a format of `YYYYMMDD`. The absolute path of the file may be e.g.:

```
/data/library/e/n/g/wikimedia/wikipedia/eng-20110901.xml.bz2
```

P_dmf will load the newest file, unpack it, split it into single articles and create a trie structure in the *derived* directory according to names of the articles. This feature is quite different from the standard procedure (conversion from one format into another) and therefore it needs a special code path.

This code path is specified via `.dmf.info` configuration files. These files should be included in a directory (or a parent directory) with the source files - that means somewhere in the *original* directory.

Wikipedia Configuration File

The configuration file should be stored in a text format. See the example `.dmf.info` example below.

The whole config file is snippet of Perl code. The main frame of the config file is an array (reference - see enclosing square `[]` brackets). Each position in the array holds a hash (reference) `{}` - the hash contains information for processing of specific file. The hash may contain several key/value pairs:

order defines the sort order of sections if instructions for multiple files are defined. (The main array contains more than one hash reference.)

regex defines a regular expression which is matched against all source files in a given (sub)directory. If the name of the source file matches the regexp, the source file is recognized as a target of this configuration file and the operations specified in *tasks* key will be processed on the specified source files.

tasks defines a hash which itself may contain three keys:

process Replaces the generic conversion chain with specific instructions

prehook Allows to modify the processed file before the conversion.

posthook Allows to modify the processed file after the conversion.

¹⁴See section 8.14 for details.

Table 8.1: .dmf.info example

```

[
{
  order => 1,
  regex => qr{/original/wikimedia/\w+/\w{3}\-d{8}.xml},
  tasks => {
    process => sub {
      my $args_hr = shift;
      my $w = $args_hr->{file};

      my $newest = 1;

      # if filename contains a date
      if (my($s, $version, $e) = ($w =~ m{(.*)((?:1|2)(?:9|0)\d\d(?:0-9][10|11|12](?:[0-2]\d|30|31))([\^/]*)([^\^*])xms)) {
        my @all_file_versions = glob("$s*$e");
        my @all_dates = sort { $b <=> $a } map {s{\A$(.*)$e\z}{1}; $_;} @all_file_versions; ## no critic
        $newest = 0 if $version != shift @all_dates;; # return newest version
      }

      if ($newest) { # no version
        # parse the filename to get name of current file
        $w =~ m{^(?<path>.*?)(?<file>\w{3}-\d{8}\.xml(?:\.\bzz)?)}xms;
        $w = realpath($w);
        # the absolute path to the file /data/library/i/n/a/original/wikimedia/wikipedia/ina-21130909.xml.bz2
        my $file = $+{file}; # name of the file: ina-21130909.xml.bz2

        my $out = $w;
        $out = ~ s{original}{derived}xms;
        $file = ~ s{\.\bzz}{xms}; # strip suffix of the archive

        my $cmd = "P_dm_wiki --action txt=$file --in $w --out $out/lvl.last/$file --fullnames";
        pmse_report("Performing $cmd\n", 3);

        ` $cmd `; # exec the command

        pmse_report("The split of $file is finished\n", 3);
      }
    },
  },
}
]

```

The type of the conversion chain and the defined hooks are independent.

The *process* key in the previous example contains instructions for the conversion. It can be briefly described in a few steps: We check the date in the filenames. This is handy when multiple Wikimedia dumps are present in the same directory, because we want to process only the latest dump.¹⁵ Then is prepared the resulting path in *derived* directory. Finally, the Wikipedia dump is unpacked and segmented by `P_dm_wiki` script.

Further Processing

`P_gnp` is able to read all files from given directory and process them at once. So, if you want to obtain a word list, or a list of n-grams from the whole Wikipedia, just specify the input directory - like e.g.:

```
$ P_gnp --in $ PMCORP_ROOT/e/n/g/derived/wikimedia/wikipedia/ <other opts>
```

8.7.7 Examples

Documents processing

Process all newest documents in *wikimedia* directory. The `--in` option is specified as a glob, thus the *wikimedia* directory is searched as a shallow structure.

```
$ P_dmf --in original/wikimedia/*
```

Process all files in *original* directory. Use `P_ici` for parallel processing and recursive descent.

```
$ P_ici --recurse --parallel --cpu 8 'P_dmf --in [%f]' original
```

8.8 P_dmp: Distance Measures Processor

`P_dmp` calculates distance measures for pairs of N-grams. You can use about cca. twenty distance metrics. See section 8.8.4 that describes these in detail.

```
$ P_dmp -?
```

8.8.1 Reference

PMSE Distance Measures Processor

USAGE `P_dmp` [options]

¹⁵The unpacking and segmentation of Wikipedia requires hundred thousands of directories and gigabytes of disc space.

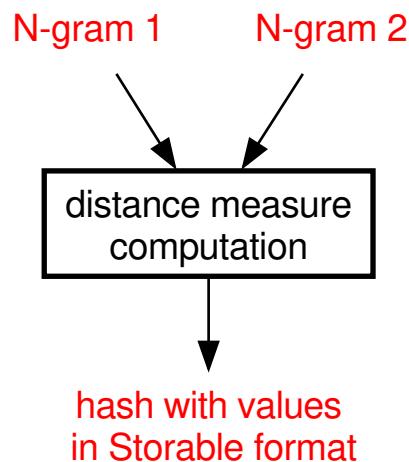


Figure 8.10: P_dmp schematic overview

OPTIONS**-bulk <filename>**

Read options `--ngrams` and `--distance` and distance parameters from given bulkfile.

-distance <distance_name=params?>

Will count values of the specified distance measure for the given N-grams.

To list available measures:

```
P_dmp -distance ?
```

Available common parameter is `normalized`. If it is true, resulting distance is rescaled and has values in interval $[0, 1]$.

There are few distances having specific optional parameters. These are:

mahalanobis: matrix

Matrix has to be regular matrix written in the form as:

```
[[1, 0], [0, 1]]           for 2-dimensional space
[[1, 0, 0], [0, 1, 0], [0, 0, 1]] for 3-dimensional space
```

etc. If unit matrix is specified, mahalanobis comes into euclid.

minkowski: p

Domain of p is interval $(0, \text{Inf}]$.

In case $p=2$ distance comes into Euclid. For $p=1$ we are receiving manhattan distance.

tversky: coef12 coef21

Both parameters are nonnegative reals and at least one is positive.

For `coef12=coef21=1` tversky becomes tanimoto. For `coef12=coef21=0.5` tversky becomes dice.

jaro_winkler: prefix_lenght_min prefix_weight

`0<prefix_weight<1/4, 0<prefix_lenght_min<=4`

-ngrams <ngram1> <ngram2>

Contains N-grams for which a mutual distance will be computed.

-out <filename>

Defines the output directory. If `--out STDOUT`, the output will be printed to STDOUT.

-separator <str>

Character(s) separating tokens in N-gram. Empty string by default.

8.8.2 Examples

Example of use

```
$ P_dmp --distance 'levenshtein=normalized=1' --ngrams 'black dog barks' \
> 'white dog barks' --out STDOUT --separator ' '
```

This will calculate normalized Levenshtein distance between trigrams `black dog barks` and `white dog barks`

The output looks like:

```
'levenshtein' => {'params' => {'normalized' => 1, 'l1' =>
['black','dog','barks'], 'l2' => ['white','dog','barks']},
'value' => '0.3333333333333333'}
```

Similarly, we could move options to bulk file. Let's have following file `bulk`:

```
[levenshtein]
l1=black dogs barks
l2=white dogs barks
normalized=1
```

We could call script like this:

```
$ P_dmp --bulk bulk --out STDOUT --separator ' '
```

8.8.3 Q&A

How to display the output?

The option `--out` defines the name of the outfile, which is stored in Storable format. (Use `--out STDOUT` to print the result on STDOUT.) To convert the output to human-readable text use the `P_dvf` or the Perl `Data::Dumper`¹⁶ module.

¹⁶<http://search.cpan.org/~smueller/Data-Dumper-2.131/Dumper.pm>

8.8.4 Distance Functions Characteristics

The following distances are supported by P_dmp. The most of them are metrics (i.e. they are following axioms of identity, symmetry and triangle inequality) when the `normalized` option is used. Several distances are not metrics (`renkonen` distance breaks axiom of identity, `tversky` distance breaks axiom of symmetry, `jaro_winkler` breaks axiom of triangle inequality etc.).

For all the following paragraphs let's suppose we have N-grams (i. e. vectors) p and q . Let's denote p_i as i 'th token of N-gram p , and q_i as i 'th token of N-gram q .

Futhermore let $\#p$ denote the number of tokens in N-gram p (i. e. $\#p = n$) and let $\hat{\#}p$ denote the number of unique tokens in N-gram p (i. e. $\#p \geq \hat{\#}p$). Similarly for the others vectors/sets.

We will also need unions and intersections. Hence let's denote I as the set of tokens at the intersection of p and q and U as the set of tokens at the union of p and q .

Another terminology is need for the distances between frequency-ordered N-grams. In fact, a frequency-ordered N-gram is a histogram. Let $f(x)$ denote the position of the token x in the vector/histogram p . Similarly, let $g(x)$ denote the position of the token x in the vector/histogram q if it exists. Position of x needn't to exists in p . In this case let's $g(x)$ denote $\#q$.

Block distance

Let's denote P as the number of elements of p in i and, similarly, let Q denote the number of elements of q in i . Then we can define the Block distance as follows:

$$D_{block}(p, q) := \frac{\#p + \#q - P - Q}{\#p + \#q}.$$

BrayCurtis distance See Dice distance.

Canberra distance

$$D_{canberra}(p, q) = \sqrt{\sum_{i=1}^n \frac{|f(p_i) - g(p_i)|}{|f(p_i)| + |g(p_i)|}}.$$

City block distance See Manhattan distance.

Chebyshev distance

Let p and q are frequency-ordered histograms of the same dimension. Then we define:

$$D_{chebyshev}(p, q) := \max_i(|f(p_i) - g(p_i)|).$$

Chess-board distance See Chebyshev distance.

Correlation distance

In fact this distance equals sample correlation coefficient. See PMLIB::Stochastic::Sample::sample_correlation.

Cosine similarity

Both vectors must be of the same length $\#p = \#q = n$.

$$D_{\text{cosine}}(p, q) := 1 - \frac{\sum_{i=1}^n p_i q_i}{\sqrt{\sum_{i=1}^n p_i^2} \sqrt{\sum_{i=1}^n q_i^2}}$$

Czekanowski distance

$$D_{\text{czekanowski}}(p, q) := \frac{2\#I}{\hat{\#p} + \hat{\#q}}$$

Damerau-Levenshtein distance

The Damerau-Levenshtein distance is calculated by counting the minimum number of operations needed to transform one string into the other. An operation is defined as an insertion, deletion, or substitution of a single character, or a transposition of two adjacent characters.

See also Levenshtein distance, which works in a similar process, except that it doesn't include transpositions.

Dice distance

$$D_{\text{dice}}(p, q) := \frac{2\#I}{\#p + \#q}.$$

It is special case of Tversky distance for $\alpha = \beta = \frac{1}{2}$.

Discrete Distance

$D_{\text{discrete}}(p, q)$ is 0 if p and q are not identical, 1 otherwise.

Edit distance See Levenshtein distance.

Euclidean distance

The Euclidean metric is probably the best known distance. The Euclidean distance between points p and q is the length of the line segment connecting them.

Here is the form for frequency-ordered histograms of the same dimension:

$$D_{euclidean}(p, q) = \sqrt{\sum_{i=1}^n (f(p_i) - g(p_i))^2}.$$

It is a special case of the Minkowski and Mahalanobis distance.

Hamming distance

the Hamming distance, between two ngrams of the same dimension, is the number of positions at which the corresponding symbols are different.

Jaccard index

The Jaccard index is used for comparing the similarity and diversity of sample sets. The index is given by the formula:

$$D_{jaccard}(p, q) = \frac{\#I}{\#U}.$$

Jaro distance

$$D_{jaro}(p, q) = \frac{1}{3} \left(\frac{m}{\#p} + \frac{m}{\#q} + \frac{m-t}{m} \right),$$

where

- m is count of tokens occurring on similar positions in both n-grams, i. e. $m = \#\left\{x : |f(x) - g(x)| < \lfloor \frac{\max(\#p, \#q)}{2} \rfloor - 1\right\}$. Let M denote set of these tokens.
- t is half the number of transpositions for tokens in M .

Jaro-Winkler distance

It is a variant of the Jaro distance metric with 2 extra parameters: we have k as a `prefix_length_min` parameter and l as a `prefix_weight` parameter.

$$D_{jaro_winkler}(p, q) = D_{jaro}(p, q) + kl(1 - D_{jaro}(p, q))$$

Kulezinski distance

$$D_{kulezinski}(p, q) = 1 - \frac{\#I}{\#U}$$

Levenshtein distance

The Levenshtein distance, sometimes called the *edit distance*, is a string metric for calculating the difference between two sequences. It is defined as the minimum number of edits needed to transform one ngram into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character.

Mahalanobis Distance

The Mahalanobis distance is based on correlations between variables by which different patterns can be identified and analyzed. It calculates the similarity of an unknown sample set against a known one.

If we let p and q be frequency-histograms of the same dimension. And we also let v denote a vector $(f(p_i) - g(p_i))_{i=1}^n$. Given a regular matrix S we can obtain the distance by using the following formula:

$$D_{mahalanobis}(p, q) = \sqrt{v^T S^{-1} v}$$

We can observe that Mahalanobis distance comes to Euclidean if S is unit matrix.

Manhattan Distance

$$D_{manhattan}(p, q) = \sqrt{\sum_{i=1}^n |f(p_i) - g(p_i)|}$$

This is a special case of a Minkowski distance.

Minkowski Distance

The Minkowski distance is a form of geometry in which the usual distance function, or metric of Euclidean geometry, is replaced by a new metric in which the distance between two points is the sum of the absolute differences of their coordinates.

$$D_{minkowski}(p, q) = \sqrt[p]{\sum_{i=1}^n (f(p_i) - g(p_i))^p}$$

where p is given as a parameter.

Needleman-Wunsch similarity Returns number of one-element matches from optimal local alignment between two vectors according do Needleman-Wunsch algorithm.

Overlap distance

$$D_{overlap}(p, q) = \frac{\#I}{\min(\#p, \#q)}$$

Renkonen distance

Let $r_v(x)$ be the relative occurrence of the token x in the ngram v . If we suppose $\#p = \#q$, then the Renkonen distance is given by the following formula:

$$D_{renkonen}(p, q) = \sum_{x \in U} \min(r_p(x), r_q(x)).$$

Russel-Rao dissimilarity

Both boolean vectors must be of the same length.

Let's denote n_{ij} as a number of corresponding pairs of elements in p and q respectively equal to i and j .

$$D_{russel_rao}(p, q) = \frac{n_{10} + n_{01} + n_{00}}{\#p}$$

Smith-Waterman similarity Returns number of one-element matches from optimal local alignment between two vectors according do Smith-Waterman algorithm.

Sokal-Sneath dissimilarity Both boolean vectors must be of the same length.

Let's denote n_{ij} as a number of corresponding pairs of elements in p and q respectively equal to i and j .

$$D_{sokal_sneath}(p, q) = \frac{2(n_{10} + n_{01})}{n_{11} + 2(n_{10} + n_{01})}$$

Sørensen distance See Dice distance.

Tanimoto distance

Tanimoto is a special case of a Tversky distance for $\alpha = \beta = 1$.

Tversky distance

Let $s(a, b)$ denote those tokens from the N-gram a which are not in the N-gram b . The Tversky distance is an asymmetric similarity measure which compares a variant to a prototype.

$$D_{tversky}(p, q) = \frac{\#I}{\#I + \alpha\#s(p, q) + \beta\#s(q, p)}$$

See both the Tanimoto and the Dice distances for important special cases.

Typo distance

Typo (typographical) distance is a combination of levenshtein and euclid distance measures. Given two strings, we use levenshtein distance to detect the place in the sequence where edit operation will happen, then we count euclidean distance of the characters participating on the operation.

Input parameter is the keyboard model and the key-map for given language. Various models are available. See PMLIB::Metric::Typo for more information.

Yule dissimilarity Both boolean vectors must be of the same length.

Let's denote n_{ij} as a number of corresponding pairs of elements in p and q respectively equal to i and j .

$$D_{sokal_sneath}(p, q) = \frac{2(n_{10} + n_{01})}{n_{11}n_{00} + n_{10}n_{01}}$$

8.9 P_dvf: Data Visualization Framework

The P_dvf script enables you to read and convert output of PMSE scripts into a human-readable format. This is necessary because the PM scripts outputs Perl native data structures which can be difficult to read easily. You can convert the data to various data-types. P_dvf also supports formatting, sorting and filtering of the output data.

You can load in data stored as Storable, text (even data printed with Dumper) and data serialized with YAML. Data generated in PMSE is stored as a PMSE object in Storable format. Each object has specific methods for visualization and post-processing depending on the nature of the data.

Data imported outside of PMSE will be handled with the most basic class: PMSE::Visualize. This class allows merely basic visualization methods and storage. But if you know your data correspond with one of the visualization class of PMSE::Visualize, you may specify type of object (see reference: [8.9.1](#)).

\$ P_dvf -?

8.9.1 Reference**PMSE Data Visualization Framework**

USAGE P_dvf [options]

P_dvf is a script intended for displaying results of low-level PMSE data-mining processes.

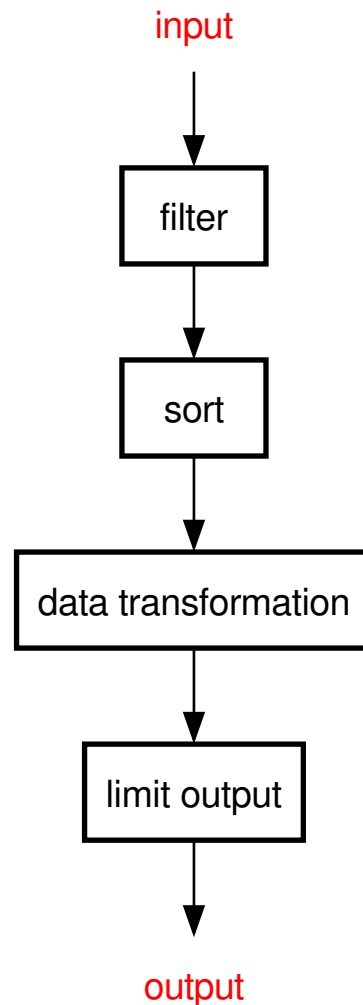


Figure 8.11: P_dvf schematic overview

OPTIONS Unlike to other PMSE scripts, options may differ depending on the type of input data. Some objects support some options and some don't. Each object is aware about methods, that may be used publicly, so when some P_dvf option is used in a wrong way, the script will throw out a warning. The important thing to know is that all PMSE objects have built-in documentation. When invoked, the self-documentation will provide info, how to handle given object in P_dvf.

Below is a complete list of options that may be used with P_dvf.

-doc all|<option>

Will list info about loaded object. If all parameter was chosen, script will provide complete overview about options that may be used with given object. You may also specify one single option.

-filter <code>

The filtering is done via a perl code:

```
'$key =~ m{\d+}xms'
```

Will remove all keys (tokens) matching given regex.

```
'$val == 60'
```

Will remove all occurrences of given value. Note: filtering of Contingency object will not cause the re-calculation of the values in the table.

-in <filename>

Defines an input file. Option `--itype` defines various available input data types.

-itype <type>

Format (type) of input data. Valid types are:

```
sbl      Perl data structure
          (default type)

dumper   Data::Dumper style printed data
text     plain text format
yaml     YAML-style data
```

-istruct <object type>

This option may be applied when you import external data structure into `P_dvf`. Let's say you have a histogram in textual format, like this:

```
AA      3.5
BB      0.22
XX      1
...
```

By default, this structure will be visualized (and dumped) by the `PMSE::Visualize` module, which provides basic methods only. When the input structure is a histogram, it may be specified with this option. Data will be treated then with appropriate module (e.g.: with `PMSE::Visualize::Histogram`).

-otype <out-type>

Type of output data. `P_dvf` adopted the object oriented approach to visualization, therefore exist a range of methods for given object. Some of them are generic and some are specific

Generic methods are

```
graphic   SVG
pdump     Data::Dumper output
storable  Storable (binary) file
text      plain text
yaml      YAML language
```

The SVG picture is created according to pre-defined options which are different for various objects.

Specific methods for object:

```
Binary tree (categorization output)
  newick
  spreadsheet
```

-out <filename>

Defines the filename for the output to be written to. If not defined, the output is printed to STDOUT.

-sort <string>

The sorting string may be:

```
+val ascending
-val descending
+key alph. ascending
-key alph. descending
```

8.9.2 Examples

Filter usage

Keys passed through positive filter:

```
_sort=$key !~ m{\d+}xmsg
sort_=$key !~ m{\w+}xmsg
```

1. Pre-sort filtering: data will be filtered before sorting. Unless the key contains a number, it is deleted. In other words: Only keys with a number will pass through.
2. Post-sort filtering: set on sorted data, only keys comprised from alphanumeric characters will pass through.

Negative filter on keys:

```
_sort=$key =~ m{\pP}xmsg
sort_=$key =~ m{\d+}xmsg
```

1. Pre-sort filtering: keys which contain a punctuation mark will be deleted.
2. Post-sort filtering: keys which contain a number will be deleted.

Filtering values:

```
_sort=$value > 15
sort_=$value < 5 and $value > 15
```

1. Pre-sort filtering: values greater than 15 will be deleted.
2. Post-sort filtering: values lower than 5 and greater than 15 will be deleted.

When we say that keys or values will be deleted we mean that the entire hash entry will be deleted.

Convert Text to Perl Data Structure

Consider the case where you have a printed data structure stored in a text file:

```
$VAR1 = { ' ' => 1078,
          ' A' => 72,
          ' Ach' => 6,
          ' Aha' => 1,
          ' Aj' => 1,
          ' Ale' => 39,
          ' Ani' => 1,
          ' Atheneum' => 1,
        };
```

Now you want convert it back to an internal Perl data structure:

```
$ P_dvf --itype dumper --in printed\_text.txt --otype storable \
> --out storable_file
```

If you need the text in human readable form, just omit the `--otype` option, and the output will be automatically converted to text. For more information about converting to supported formats with see figure table below.

FORMAT	txt	pdump	YAML	Storable	dot	JSON	XML
txt	+	+	+	+	-	?	?
pdump	+	+	+	+	-	?	?
YAML	+	+	+	+	-	?	?
Storable	+	+	+	+	+	?	?
dot	-	-	-	+	-	?	?
XML	?	?	?	?	?	?	?
JSON	?	?	?	?	?	?	?

8.10 P_gnp: Generic N-grams Processor

P_gnp is a powerful tool that covers a wide range of functions. It calculates and processes N-grams, creates contingency tables, as well as calculates MI-score and t-score. Also, it is capable to create ranks and compute key-words.

See the schema [8.10](#) for an overview of the process flow.

```
$ P_gnp -?
```

8.10.1 Reference

PMSE Generic N-grams Processor

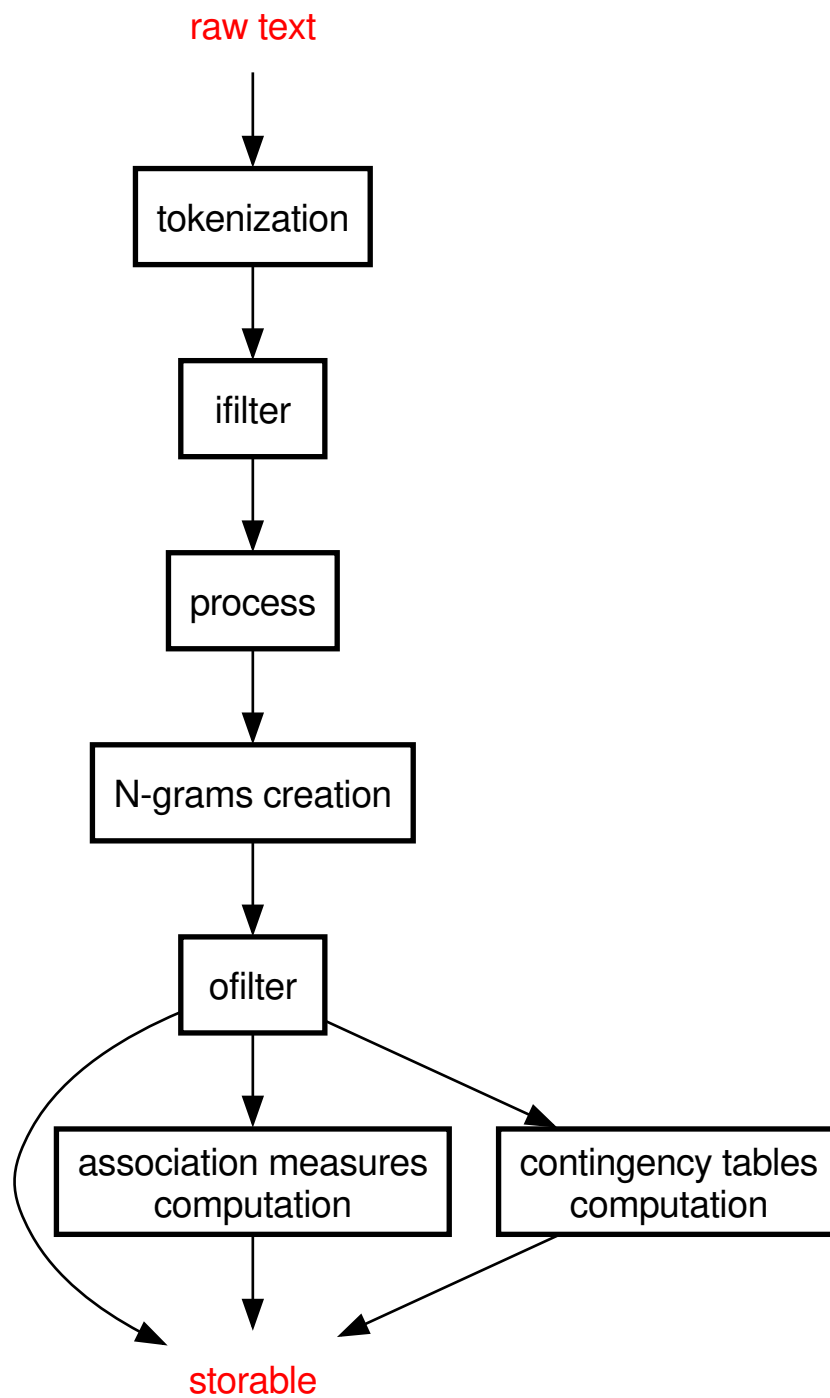


Figure 8.12: P_gnp - overview of the processes

USAGE P_gnp [options]

P_gnp is the core N-grams processing script in PMSE. It provides a wide range of N-grams evaluations.

The process workflow of P_gnp is as follows:

```
text(s) -> tokenization -> N-grams frequencies ->
contingency tables -> association measure score -> output
```

Throughout this processing chain, various hooks are deployed and support extensive data mangling (filters, transformations etc.).

OPTIONS

-bulk <file>

With the `--bulk` option you can define an INI file, which can be used for histogram / hash filtering and tokens processing. The INI file has a structure of '[section]' and 'name=value', the heredoc style can be used. 'Section' is the name of a procedure (process / ofilter), 'name' is name of a hook (the same as in `--process` and `--ofilter`). 'Value' stores the code, which will be executed on the result of an action.

-cluster [action1, action2 ... | ?]

Available actions are:

```
count - results are n-grams counts
freq  - results are n-grams frequencies
prob  - results are n-grams probabilities
```

If no action is specified, the default action is output.

It is possible to specify several actions in one call.

-contingency

Computes contingency tables for N-grams.

Note: In case of a lot of N-grams it can take some time.

-delimiter <regex>

Delimiter has the form of a Perl regular expression. It enables the tokenizer to dissect the text into discrete tokens. If the user doesn't set his own value, the default delimiter from the PMLIB tokenizer is taken.

-histogram [<action>=<filename> | ?]

We can count the distribution (histogram) for the result of each 'action'. The histogram is stored in file 'file'. Actions could be:

```
count      histogram of n-gram(s) occurrences
prob       histogram of n-gram(s) probability
freq       histogram of n-gram(s) frequency
measure    histogram of n-gram(s) measure(s)
```

Output file is stored in Storable format. When the hook is 'measures', a code referring to appropriate measure and dependence is prepended to the filename, because action 'measures' may contain multiple values.

-ifilter [`<type>=<regex> | ?`]*

This option may be provided multiple times (with different content for `type` of course) to define various filters, these are inserted at specific places during data stream processing. Valid values for `ifilter <type>` are:

```
+token    tokenizing step: matching tokens will pass
-token    tokenizing step: matching tokens will be blocked
+ngrams   n-gram processing step: matching tokens will pass
-ngrams   n-gram processing step: matching tokens will be blocked
```

`<regex>` may be any regular expression. Please take care to quote the whole expression, like '`<filter>=<regex>`', to prevent the shell modifying it.

-in `<filename|path>`

You may specify a file or a path to directory. If a filename is specified, only this concrete file will be processed.

If a path to directory was specified, the `P_gnp` will find recursively all plain text files with `*.txt` suffix and these will be tokenized in a parallel loop.

Note: in such case cross-text n-grams will be created unpredictably, because the order of input files will be random - that is a consequence of parallel processing.

-keywords `<filename>`

Outputs keyword weights. `<filename>` is hash in storable format representing frequencies/probabilities of reference corpus.

Interpretation: higher value for keyword means higher representativeness in corpus regarding to reference corpus.

-measure [`<measure_name>=<args> | ?`]

Computes requested measure for all N-grams.

The following measures are currently implemented:

```
mi    miscore
t     tscore
```

Both MI-score and t-score accepts dependencies as a parameter. Dependencies are of the form

```
all
(nothing)
dep11|dep12|...|dep1N,dep21|dep22|...|dep2N,...,depM1|depM2|...|depMN
```

where '`depkl`' is of the form '`number1 number2 ... numberp`', where `number` is the `i`-th token of an N-gram.

If "all" is specified, all dependencies will be printed.

If no param is specified, all tokens are considered as independent. It is the same as if we would specify `1|2|3|...|n`, i. e. `1|2` for bigrams.

Let's look at the 4grams example. If we specify: `--measure 'mi=1 2|3 4,1|2|3|4'`

P_gnp outputs for each N-gram:

- 1) MI-score for dependence between 1st and 2nd token
- 2) MI-score for no dependencies

Note: in dependencies, independent tokens could be omitted. This means it is sufficient to put '1 2' instead of '1 2|3|4'.

-ngram [`<n-size>` `<window>` `<separator_character(s)>`]

`<n-size>` must be a positive integer, `ngram` option defines the number of tokens from which an N-gram is comprised.

`<window>` is also a positive integer and defines the number of contextual tokens from which an N-gram is generated.

`<separator>` defines string to be used for separating tokens in an N-gram.

Default value is `2 2 ' '`. If you specify one of them, you have to specify all the others.

-ofilter [`<hook>=<code>` | `?`]

Ofilter can be used on the result of each action, which is always a hash, thus with `<code>` we can affect keys and values. Hook denotes the specific part of the process where the `ofilter` is applied. Hook can be:

```

ngrams      filter hash with N-grams and their frequencies
_hist       filter result of action before histogram is created
histogram   filter hash with histogram values (note: keys and values
            are both numeric)

```

Code could have form of `$key =~ m{.*}xmsg` OR `$value >=< number`. You can also insert the name of an INI file via the `-bulk` option.

-out `<directoryname | STDOUT>`

Defines the output directory. Results of all actions will be printed into correspondingly named files in this directory in the Storable format.

If `STDOUT` is specified instead of file, the Perl data structure will be printed directly on `STDOUT` in `Data::Dumper` format.

-process [`<hook>=<subst>` | `?`]*

`<subst>` is a Perl substitution operator: `"s{pattern}{replacement}flags"`, whereas `"pattern"` is a regular expression, `"replacement"` may be a string or even Perl code if the `"e"` flag is given. For further info see e.g. <http://perldoc.perl.org>

`<hook>` may be one of the following:

```

_ngrams     first tokens processing before set is made
ngrams_     second tokens processing before set is made

```

A filter can be applied between both hooks. After the second processing of tokens is finished, the `tokens-list` is available, from which are generated N-grams.

-rank `<measure_name | ?>`

Outputs rank of specified measure.

Measures have their own parameters (like dependencies in MI-score or t-score). If both **-rank** and **-measure** are specified, rank is computed for all satisfying measures. I. e. if **-measure 'miscore=1|2|3,1|2 3'** **-rank=miscore** is specified, two ranks for MI-score are computed.

If **-rank** is specified and **-measure** is not, the default parameter for that measure (which is specified as parameter of **-rank**) is used.

8.10.2 Examples

Get basic N-grams list

```
$ P_gnp --cluster count --ngram 3 4 ' ' --out outdir --in corpus.txt
```

Will get 3-grams from the window of size 4 from "corpus.txt" and store them as Perl data structures into the "outdir" directory in the Perl Storable format. Tokens in each N-gram are delimited by whitespace.

Note: The number *N* must be a positive integer number, thus '**--ngram -2.5'** is not accepted.

Define more options for N-grams list

```
$ P_gnp --cluster count --ngram 2 3 ' ' --out outdir --delimiter '\s+' \  
> --process '_ngrams=s{A}{B}xmsg' --bulk INI_file \  
> --histogram 'ngrams=histogram' --ofilter 'histogram=$value < 5' --in corpus.txt
```

Computes bigram frequencies from a 'text' window of size 3. The tokens in the text are split by white-space. The tokens in each N-gram are also delimited by whitespace. Before we get the hash with the N-grams, we change each token 'A' to token 'B'. Where we originally had 'A D', we now have 'B D'. With **--bulk** we will load the INI_file with the next commands for processing. When the N-grams are finished, we create a histogram, which stores distributions of the N-gram frequencies. Because we don't want any values which occur less than 5 times, we use the **--ofilter** option to filter out these values.

Process multiple source texts

```
$ P_gnp --cluster count --ngram 2 4 ' ' --in /data/library/e/n/g/derived
```

Will find recursively all **txt* files in /data/library/e/n/g/derived and process them at once.

Get contingency table for N-grams

```
$ P_gnp --contingency --ngram 2 2 ' ' --out outdir --in corpus.txt
```

Computes bigram contingency tables. For the options 'contingency' and 'measure', the separator must be given.

Get N-grams and MI-score

```
$ P_gnp --measure 'mi=all' --ngram 2 3 '<>' --out outdir --in corpus.txt
```

Computes MI-score for bigrams in a window of size 3, the tokens in each bigram are delimited like this: word_A<>word_B

```
$ P_gnp --measure 'mi=all' --ngram 3 4 '*' --out outdir --in corpus.txt
```

Computes MI-score for trigrams in window of size 4, resulting structure will contain all types of MI-score.

```
$ P_gnp --measure 'mi=1 2|3 4' --ngram 4 4 ' ' --out outdir --in corpus.txt
```

Computes MI-score for trigrams in window of size 4, resulting structure will contain the MI-score with the dependencies in 3rd and 4th tokens and in 1st and 2nd tokens.

8.10.3 Q&A

What is a 'window' in practice?

There is a sentence: Fred is going to Wilma, let's assume the user wants trigrams from window = 3. Then the first trigram will be: Fred is going, the second: is going to and the third: going to Wilma. If window = 4, we will have 4 tokens from which the trigram will be generated, in which case the total counts of trigrams will increase:

window = Fred is going to. Possible trigrams are: Fred is going, is going to, Fred going to and Fred is to.

What type of separator may be defined?

Everything you can represent with a regular expression.

```
'Fred is going' (--separator = ' ' [whitespace])
'Fred*is*going' (--separator = *)
'Fred<>is<>going' (--separator = <>)
```

N-grams in the output file are strange. They have different sizes.

Did you use the 'ifilter' option? It's possible that "trash" might be found within the tokens. For example: white spaces, punctuation, ends of lines etc. You could set

```
--ifilter '+token=\A\p{Alpha}+\p{Digit}*\z'
```

to get N-grams comprised from alpha-numeric characters only.

How do the dependencies work? What is MI-score and t-score for?

The MI-score for bigrams is computed as given by this expression:

$$MI = \log_2 \frac{P(\text{token1}, \text{token2})}{P(\text{token1})P(\text{token2})}$$

We can generalize this into a trigrams formula in several ways. If all the tokens are independent, we could use the following formula:

$$MI = \log_2 \frac{P(\text{token1}, \text{token2}, \text{token3})}{P(\text{token1})P(\text{token2})P(\text{token3})}$$

In that case we don't use dependencies.

If we wanted to set a dependency between `token2` and `token3`, the formula would look like this:

$$MI = \log_2 \frac{P(\text{token1}, \text{token2}, \text{token3})}{P(\text{token1})P(\text{token2}, \text{token3})}$$

We add `--measure 'mi=2 3'`.

If we want to know about all kinds of dependencies, we have to add the option `--measure mi=all`.

Here is a formula for bigrams for the t-score:

$$t = \frac{NP(\text{token1}, \text{token2}) - P(\text{token1})P(\text{token2})}{\sqrt{NP(\text{token1}, \text{token2})}}$$

Dependencies behave in a similar fashion in both t-score and MI-score.

However, the MI-score is not quite so accurate for low frequency N-grams. Some linguists recommend to filter the N-grams before computing the MI-score.

In the case of the t-score, high values are reached by synsemantic words (like *in*, *on*, *with*, *or* and so on) or by punctuation marks.

How to interpret the dependencies?

If no dependencies are set, we simply do a basic test for collocations.

In case we want to know more detailed information about current collocation, dependencies come in handy.

Let's say, we've found the trigram 'a b c' with high MI-score without dependencies. This means that some tokens of this 3gram are probably 'somehow' related. How related? That's where we should ask for dependencies.

If we switch the `--measure mi=all` option, we can see if the high MI-score in the independent case is caused by the bigram 'a b' (dependence '1 2'), or by the bigram 'b c' (dependence

'2 3'), or even by 'a * c' (dependence '1 3'). Where all kinds of MI-score are high, this can signify that the entire trigram 'a b c' is a collocation.

If we have the 5gram 'a b c d e' we can also set more than 1 dependency. For example the dependencies 'a b' and 'b c d' could be specified altogether as '1 2' '3 4 5' (i. e. high MI-score in this 5gram could be caused by the bigram 'a b' and the trigram 'c d e' being collocations). We should note that with any outcome where 'n' is bigger than 3 or 4, we might well look for other kinds of MI-score.

The same techniques can be used for t-score.

8.11 P_help: PMSE Helper

P_help will give you a basic reference about PMSE environment. Current version supports also listing of environmental variables.

```
$ P_help -?
```

8.11.1 Reference

PMSE Helper

USAGE P_help [options]

P_help provides generic help for PMSE environment.

OPTIONS

-pmroot <path>

You can specify PMSE root manually. Otherwise will be used a value of PMSE_ROOT variable.

-topic <type>|?

Choose topic of help. Available are:

```
list will list all PMSE scripts, their function + version
env  will check and list environmental variables
```

8.11.2 Examples

Basic usage

```
$ P_help --topic list
```

Will list all PMSE scripts and will print brief information about their functions.

8.12 P_ici: Intelligent Command Iterator

The purpose of this script is to provide a powerful, but easy to use, command iterator. While the shell provides several methods for using loops, it can be quite difficult to get simple repetitive commands done quickly as so called "one-liners". P_ici is meant to facilitate precisely this: Execute repetitive commands that change only a little in their input parameters.

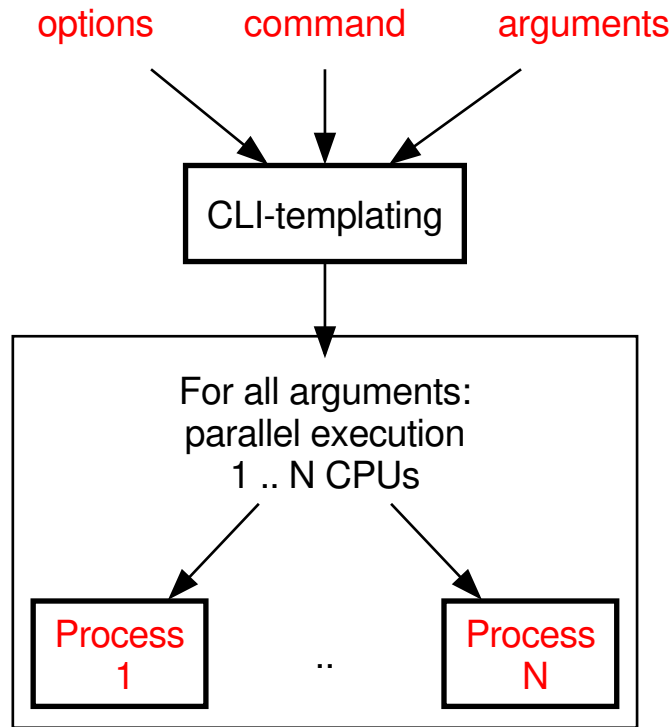


Figure 8.13: P_ici schematic overview

While the command may be used most often directly with files, it should also be considered a more generic tool. The arguments could be several tokens that have to be fed as input parameters to the input commands to be executed. Finally, if file searching, defining and filtering capabilities of the P_ici are not sufficient for your needs, you might consider combining this tool with the UNIX `find` utility.

```
$ P_ici 'echo [%f]' `find . <some exotic search params>`
```

There are some extensions to this idea, like parallelization, which go beyond the possibilities of shell iteration and allow you to make better use of your system resources (see a few examples in subsection 8.12.2).

P_ici allows to load two types of arguments: "pure" arguments (where no semantic is assumed) and arguments related to path semantic (files or directories). The second type of arguments is loaded via `--in` option. The 'pure' arguments are interpreted in the frame of the command *as they are*.

On the other hand, files (specified with a glob or a recursive search) are transformed into their absolute paths before they are used as an argument for the command. Here is an example:

```
$ P_ici --in '*' 'echo [%f]'
```

```
/home/foo/file_a.txt
/home/foo/Documents
/home/foo/file_b.pdf
```

Will list all items (files, directories, links, etc.) that are matched by the '*' glob in the current working directory, with their absolute paths.

```
$ P_ici 'echo [%f]' *
```

```
file_a.txt
Documents
file_b.pdf
```

The shell will first find all visible files and directories in cwd, then it will send this list to P_ici as arguments. The result is a list of files with their paths relative to cwd.

It is very important to understand the difference between these two concepts and the need to escape shell expansion when you want P_ici to generate the list of files. Take for example the command

```
$ P_ici 'echo [%f]' --in *
```

which gives the output:

```
/home/foo/file_a.txt
Documents
file_b.pdf
```

Why is that? Because the shell expanded the unescaped glob '*' and send the arguments to P_ici, which will take these arguments and only the 1st one of them will be bound as parameter to --in and gets its file semantics, while the others will be passed as regular arguments as if you had done

```
$ P_ici 'echo [%f]' *
```

A Word About Parallelization

The available `--parallel` option allows you to spread the tasks to the available CPUs. In combination with the `--cpu` option you may even over- or undercommit the number of available CPUs (and thus the level of parallelization) overwriting the autodetection.

You may want to undercommit if you have IO-intensive tasks and your IO subsystem could not cope with a number of parallel tasks equivalent to the number of CPUs. On the other hand you may want to overcommit if you are developing and testing CPU-intensive parallelised tasks for a bigger system on a smaller system.

It is also noteworthy to say, that an automatic load balancing happens. If you have 500 tasks to process and 4 CPUs available, then each of the CPUs will get approximately 125 tasks if these are about the same runtime each.

If on the contrary all tasks have various runtimes, the CPUs will be given tasks to process as they become free. While `P_ici` itself cannot know the runtime of the task in advance, if you do, you can send longest-running tasks to be processed 1st and thus get an optimum overall runtime.

For many applications we can work under the assumption, that the longest files will also be those who take the longest time to process. Then, you can ensure by giving `--insort -size` that the biggest files are processed first.

```
$ P_ici -?
```

8.12.1 Reference

PMSE Intelligent Command Iterator

USAGE `P_ici [options] cmd [argument(s)]`

Where `cmd` defines the command that shall be applied to all arguments. `cmd` contains one or more unix/shell commands and one or more of these special placeholders:

```
[%\d] = the argument at index \d (given as positive integer)
[#]   = the index of the current file (starting at 1)
[b]   = basename of the current file (without last suffix)
[d]   = directory portion of the current argument/file
[f]   = the current file/argument quotedmeta'd
[F]   = the current file/argument raw
[m]   = modified filename quotedmeta'd
[mb]  = modified basename
[md]  = modified directory portion
[ms]  = modified last suffix
[M]   = full modified filename unquoted
[t2]  = the current file/argument-trie as string
[2t]  = the current file/argument-string as trie
[p]   = the current P_ici process-id
[s]   = suffix of the current file/argument
[t]   = current unix time in seconds since the epoch
[u]   = current time in microseconds after [%t]
[x]   = size of the current file/argument
```

This substitution does not happen if `--raw` option is given.

OPTIONS

-fatal

If this flag is set, execution of the issued command will stop on error. Else execution will continue (default).

-filter_name mod=<repl>|neg=<regex>|pos=<regex>

Define positive (everything that matches will pass), negative (everything that matches will not pass) and modification (everything will go through a replacement regex) filters for input files. Please be aware, that only files defined by `--in` parameter are actually considered for filtering. Arguments are unfiltered.

You can define all filters at once anywhere on CLI, where the 'mod' filter is applied after the positive and negative filters.

-filter_type [<string>]

Define a string with file test operators. By default this string is empty and all types of files are allowed as arguments. All Perl file test operators without the dash are allowed, e.g. 'f r s' means "only readable files with nonzero size". Please be aware, that only files defined by `--in` parameter are actually considered for filtering. Arguments are unfiltered.

-grace

If this flag is set and no arguments are given, the script will end gracefully.

-in <filename|glob>

Defines an input file or a glob, in which case all filenames that match will be considered for input. The glob must be escaped/quoted to prevent the shell from expanding it.

Arguments specified via `--in` option will be treated as files / or directories, i.e. the path semantic will be expected.

If you want the input arguments to be 'pure', specify them on the end of the command as is suggested above.

-insort <type>|?

Defines the type of sorting for input files. Possible values for <type> are:

```
orig      sequence of input files as they come from shell (default)
shuffle   apply Fisher-Yates algorithm (unsort/shuffle file list)
+alpha    sort in ascending alphabetical order
-alpha    sort in descending alphabetical order
+size     sort in ascending file size (from smallest to biggest)
-size     sort in descending file size (from biggest to smallest)
+time     sort by file timestamp (from oldest to youngest)
-time     sort by file timestamp (from youngest to oldest)
```

-max <n>

Perform a maximum of *n* iterations.

-parallel

Execute commands in parallel, depending on number of cpus.

-raw

Do not interpret the command argument (ignore all [%x] sequences - if present - and treat them verbatim)

-recurse

Recursive descent when iterating arguments (files/directories/...). This option should be called together with `--in`.

-sleep <n>

Sleep *n* milliseconds between `cmd` calls in the main loop.

8.12.2 Examples**Find PDF Files**

Find all PDF-Files (in the current directory and subdirectories) and convert them to ASCII-text (in parallel).¹⁷

```
$ P_ici --in '*' --filter_name pos='\.(?(i)pdf)\z' --parallel --recurse 'pdftotext [%f]'
```

The PDF→TXT conversion will be performed in parallel, depending on the number of CPUs available - or given (`--cpu`).

In the above example, we are using a positive filter `--filter_name pos` to grab only pdf files from all files that were matched by `'*'`. We do this, because the filter we apply is case insensitive and thus we will get all files with endings like `.pdf`, `.PDF`, `.Pdf` and so on.

We could have achieved a similar effect more efficiently by directly globbing for PDF-files only like

```
$ P_ici --in '{pdf,PDF}' --parallel --recurse 'pdftotext [%f]'
```

but this would omit weird cases like `.Pdf`, `.pDF`, `.pDf` etc. If you do know, that these do not occur in your data, the second form is more efficient. If you do not know the data or do not want to make any assumptions, the first form is preferred.

Identical Filenames in Subdirs

Next we will make the above example more specific. Let's consider we have files of identical names in subdirectories:

```
./web/dir1/listA.pdf
./web/dir1/listB.pdf
./web/dir2/listA.pdf
./web/dir2/bookxyz.pdf
```

If we use example 1, the second file called 'listA' will overwrite the first file of the same name when the text is extracted into the target directory, because we effectively flatten the

¹⁷`pdftotext` can be found as part of the `poppler` (see <http://poppler.freedesktop.org/>) package on most Linux distributions.

directory structure. Thus we need to disambiguate 'dir1/listA' and 'dir2/listA' in the output. We can do that, for instance, by adding a timestamp, [%u], to the output file name:

```
$ P_ici --in '*' --filter_name pos='\.(?i)pdf)\z' \
> --parallel --recurse 'pdftotext [%f] target_dir/[%u]-[%b].txt'
```

We recommend that you do not apply the template [%#] (an index of the current file being processed) when the option `--parallel` is in use. If you want to distinguish the identical filenames, this template will not help you, because each process evoked by `P_ici` obtains only one file from the source directory, thus the index is '1' for all files. Instead of [%#] you can use a timestamp [%t][%u].

Advanced Filtering

You can apply all filters at once, where the available name filters (`--filter_name`) 'pos' and 'neg' will filter the given file names by applying the regular expressions given in a pass and block semantics respectively. The 'mod' filter is applied after these and will change the file names according to the replacement regular expression given.

Let's say you need to perform this complex filtering:

“Get all .pdf files which start with a lowercase character and have an uppercase character as the second character in their file name. From these filter out all those files that contain a 'x' (or 'X') anywhere in their filename. Finally, modify the resulting list to lowercase.”

Your filtering arguments could look like this:

```
$ P_ici 'echo [%f]' --in '*.pdf' --filter_name pos='^A[[:lower:]][[:upper:]]' \
> --filter_name neg='[xX]' --filter_name mod='s{(.+)}{lc($1)}ge'
```

The reason we use the regex filtering instead of trying to use shell globbing, is because if our file names should be utf8, we would not catch all uppercase characters by a mere [A-Z] - thus omitting greek or cyrillic or other uppercase characters.

You can additionally filter by file type, where all Perl file test operators¹⁸ are allowed. Please be aware, that the file test operators must be given without the preceding dash. Also, `P_ici` will ensure each operator is evaluated only once, even if given multiple times.

Modification Filtering

```
$ P_ici 'mkdir -p [%M]; mv [%f] [%M]' -filter_name \
> mod='s{.+\/(.)+\z}{uc($1)}e' -in 'alfons beta cecil'
```

8.13 P_rer: Regular Expression Replacer

The `P_rer` script provides support for performing arbitrary and highly complex replacements on the contents of a set of files.

¹⁸see <http://perldoc.perl.org/functions/-X.html>

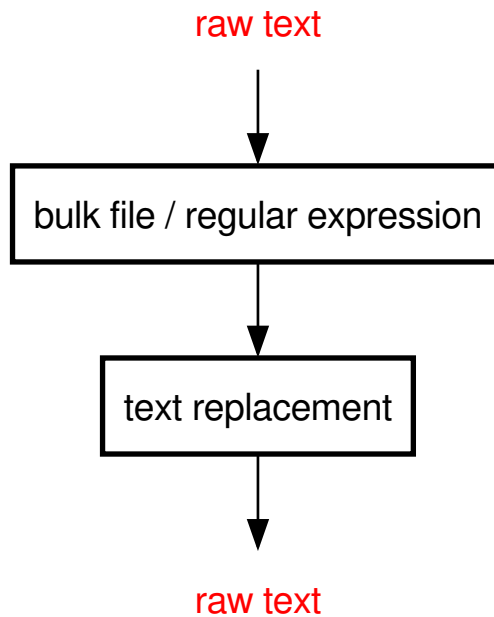


Figure 8.14: P_rer schematic overview

```
$ P_rer -?
```

8.13.1 Reference

PMSE Regular Expression Replacer

USAGE P_rer [options] regexp(s) file

At least one regular expression in the form `s{}` must be given. On the end of the command, exactly one file argument must be given, but you can enter a glob matching several files. Input and output may be specified also with options. You have to quote anything that contains white-space characters. Use 'P_ici' for more complex file specifications (albeit this imposes a performance penalty).

OPTIONS

-bulk <file>

If regular expressions are too cumbersome to enter with bash escapes and/or it is expected that you will need to recycle the replacement commands, you can specify an INI-style bulk file that will be used to define the replacements.

You can name the sections in any way you like, but the expected keys per section are `<i>` (mandatory and must not be empty), `<o>` (optional - default: empty string) and `<m>` (optional - default: 'xmsg').

You can use heredoc-style for the value definitions in all keys and you may define these keys multiple times per section, in which case they are concatenated. All other key names are ignored.

The resulting replacement is `s{i}{o}m`.

```
[section name]
i = \N{INFORMATION SEPARATOR ONE}(.)
o = foo$1
m = ms
```

will perform a `s{\N{INFORMATION SEPARATOR ONE}(.)}{foo$1}ms` replacement on the text in question.

```
[multiline]
i = a
i = b
o = x
o = y
```

will perform a `s{ab}{xy}xmsg` replacement.

If several sections are given, the replacement order is defined by the alphabetical order of the sections.

-in <file>

Input file. When you use `--in`, you have specify also `--out` and vice versa. Prepending of file specification to the end of the command is not allowed.

You may use

```
P_rer <in> <out> <regexp>
```

or

```
P_rer <regexp> <file>
```

Combination is not possible.

-out <file>

Outfile. When you use `-out`, you should use also `-in` option. See `-in` for details.

-sect <match_rx>

You may optionally give a regular expression to define which sections are to be processed. By default, this are all sections that do not start with a '!'. (default is `qr{(?<!\A!)}`)

8.13.2 Examples

Replace occurrences

Replace all occurrences of 'computer' with 'notebook' in all text files in the current directory.

```
$ P_rer s{computer}{notebook}g '*.txt'
```

Remove trailing space

Remove all trailing space in the file `spatial.text`

```
$ P_rer 's{\s+\z}{}g' spatial.text
```

As the regular expressions make use of the Perl Regular Expression engine, you can also make full use of embedded Perl code within your regular expressions:

Edit timestamps

Replace all occurrences of `%d%` in all `*.time-stamp` files with the current date.¹⁹

```
$ P_rer 's{%d%}{scalar localtime()}ge' '*.timestamp'
```

8.14 P_trt: Text Repair Tool

Similar to `P_rer`, `P_trt` is a tool for text manipulation. Unlike to `P_rer`, `P_trt` is working on more abstract level. Basic function of `P_trt` is a deformatting of text. The most general function is `--action deformat`.

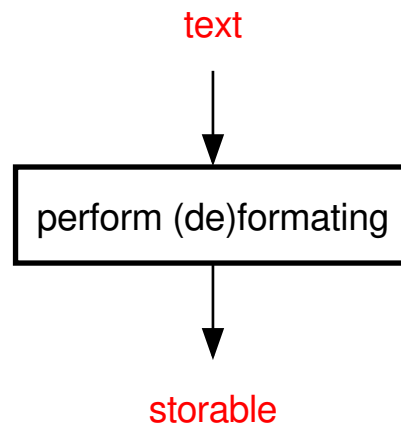


Figure 8.15: `P_trt` schematic overview

```
$ P_trt -?
```

8.14.1 Reference

PMSE Text Repair Tool

USAGE `P_trt` [options]

`P_trt` manipulates with text on more abstract level.

¹⁹quoting the regular expression may be necessary if it contains spaces

OPTIONS**-action <name>|?**

Available actions are:

```
dehyphen
compress_whitespaces
deformat
remove_headline
repair_interpunction
trim_whitespaces
```

-in <filename>Defines the input file name. If `--in STDIN`, the input will be read from STDIN.To end the the input use `*END*` sequence.**-out <dir>**Defines the output directory. If `--out STDOUT`, the output Will be printed to STDOUT.**8.14.2 Examples****Repair short text**

Consider two lines of text followed by 2 newlines:

```
Perl is      cool.
( I think .  )
```

Now replace multiple inline white space characters to one single space and trim leading and trailing white space of a string:

```
$ P_trt --out STDOUT --in text.txt --action compress_whitespaces
```

```
Perl is cool. ( I think . )
```

Action `deformat` has in this case the same effect, because it replaces all newlines with spaces. Then the function performs compression of white space characters. (The end white space is trimmed, because nothig follows.)Action `repair_interpunction` will try to make the text more coherent. It will group together the text, punctuation and brackets. Only one line will be affected in this example.

```
$ P_trt --out STDOUT --in text.txt --action repair_interpunction
> \end{verbatim}
>
> \begin{verbatim}
> (I think.)
> \end{verbatim}
>
> Here is a more complex example of interpunction repair:
>
```

```

> \begin{verbatim}
> Bla bla , bla.( Ha ,ha. ) # false
> Bla bla, bla. (Ha, ha.) # correct
> \end{verbatim}
>
> Action \verb;trim whitespaces; will trim white space characters on the beginning and end
> of the string.\\
>
> \begin{Verbatim}[frame=single]
> P_trt --out STDOUT --in text.txt --action trim whitespaces

```

```

Perl is cool.
( I think . )

```

More complex reparation of text may be performed by multiple use of P_trt.

8.15 P_vls: Variable Length Splitter

P_vls is one of the helpers PMSE script. It is intended to cut specified amount of word types from a text file. P_vls converts the text file to a frequency list, sorts the list in descending order and then splits the list according to specified options.

Tokens delimiter can be specified.

It is possible to specify both absolute or relative number of the carved word types. Also, it is possible to cut a range. The output file is stored in Perl Storable format.

```
$ P_vls -?
```

8.15.1 Reference

PMSE Variable Length Splitter

USAGE P_vls [options]

P_vls extracts words from given files and gives a cut its order.

OPTIONS

-cut <str>

str is composed of bounds delimited by whitespaces. Each bound specify where to split the hash.

We returns a list of cuts from one bound to forthcoming one.

These boundaries could be specified as

```

absolute number (e. g. 40)
percent value (e. g. 40%)
increment (e. g. +10%, +50)

```

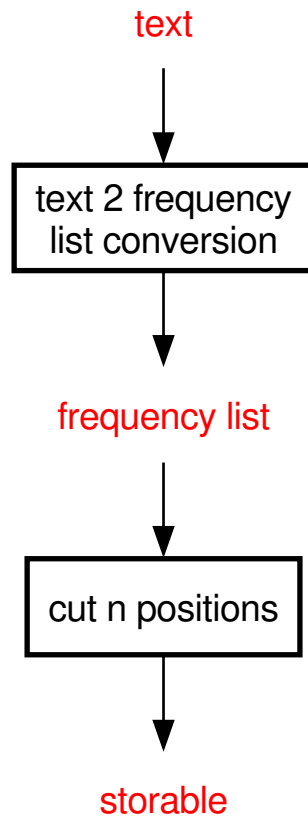


Figure 8.16: P_vls schematic overview

For example string '0 0 1 2 +1% 100%' for text of 1000 token types will be interpreted as following:

```

1st cut: first word
2nd cut: second word
3rd cut: third word
4th cut: from fourth to fourteenth word
5th cut: from fifteenth to the end (1000th)
  
```

-delimiter <regex>

String that will be converted to regex which delimits words.

-in <filename>

Input files.

If you specify glob, be sure to use apostrophes because of bash expansion.

-out <directoryname> or <STDOUT>

Defines the output directory. Results of all actions will be printed into correspondingly named files in this directory in the Storable format.

If STDOUT is specified instead of file, the Perl data structure will be printed directly on STDOUT in Data::Dumper format.

8.15.2 Examples

Basic information retrieval

```
$ P_vls --out STDOUT --in frequency_list.txt --cut '1000'
```

Will cut first 1000 words from frequency_list.txt and print them on STDOUT.

```
$ P_vls --out STDOUT --in frequency_list.txt --cut '10' '20'
```

Will cut word types from 10th to 20th position of the frequency list and print them on STDOUT.

```
$ P_vls --out STDOUT --in '*' --cut "+1% 0%" --delimiter '\s+'
```

Will cut e.g. first 10 tokens from a wordlist of 1000 tokens and will print the modified list in reverse order on STDOUT. The input is specified as a glob. We delimit tokens just by a white-space.

8.16 PMSE: Tutorial

8.16.1 Learning by Example

The more functions PMSE offers for text processing the more difficult it becomes to learn how to use PMSE effectively. To get acquainted with the PMSE suite quickly, we have prepared tutorials with examples based on experience, of step-by-step PMSE script usage. You can find these on the following pages.

8.16.2 Corpora

Here are a few corpora we are using in examples.

C1

```
A natural nuclear fission reactor is a uranium deposit where
analysis of isotope ratios has shown that self-sustaining nuclear chain
reactions have occurred.
```

C2

```
Klasterec above Ohre is a nice city.
```

C3

```
192.168.0.1, 192.168.0.2, 192.168.0.1 192.168.0.3, 192.168.0.1, 192.168.0.3
```

8.16.3 P_csp Interactive

Basic Usage of `-iact`

Now we will show the basic usage of the `P_csp` command, step by step. This tutorial uses 'interactive mode' and should help you to get started quickly. There is also the 'CLI mode'²⁰, if you prefer. Interactive mode is easier to use than the CLI mode, but it is less powerful. Complete functionality is achieved with CLI mode, although it is more complicated to learn. So while this is good place to start, by the time you have completed these exercises, you are encouraged to explore the power of the CLI.

We will use the *C2* corpus in this tutorial:

```
Klasterec above Ohre is a nice city.
```

```
$ P_csp --iact
```

First we have to insert the path to our corpus.

Insert path to the corpus :

```
pmse> c2
```

```
Wrong format (file is unreadable, empty or nonexistent)
```

This error message is generated because we made a mistake - the system is case sensitive. We have to type the path again, using the correct case:

```
pmse> C2
```

Insert directory or 'STDOUT' (default value: >P_csp.1569<):

```
pmse>
```

Now you can specify the path where the output directory is located. If you want to just display the output on the screen, type `STDOUT`. Note that entering no input (just pressing ENTER) will cause `P_csp` to create a local directory with the ID of the current run. This 'default' directory will be placed in the CWD²¹.

Insert regex delimiting tokens (or ENTER to skip):

```
pmse>
```

This option is called the `delimiter`. It affects how the text is split. There is a default tokenizer in PMLIB, but you can define your own, let's take a look at an example (corpus *C2*):

Insert regex delimiting tokens (or ENTER to skip):

```
pmse> \s+
```

result:

```
120424-09:42:18.098449/1 - Output directory: STDOUT
```

```
'utcount' => {'a' => 1, 'above' => 1, 'Klasterec' => 1,
              'city.' => 1, 'is' => 1, 'Ohre' => 1, 'nice' => 1},
```

We can see the counts of tokens in the sentence. We used `\s+` - one or more spaces - to separate tokens in the sentence. The output looks almost OK, except for the sequence 'city.'

²⁰CLI = command line interface

²¹Current working directory.

To clean up this token, and others like it, we must add a regex to define punctuation as the delimiter:

Insert regex delimiting tokens (or ENTER to skip):

```
pmse> \s+|\p{P}+
```

result:

```
120424-09:49:18.551789/1 - Output directory: STDOUT
'utcount' => {'city'      => 1, 'a'      => 1, 'above' => 1,
              'Klasterec' => 1, 'is'     => 1, 'Ohre'  => 1, 'nice'  => 1},
```

Choice of the delimiter determines the concept of your tokens. Because of our choice of delimiter, you don't have punctuation marks and spaces among your tokens now. Also consider 'words' like "Mike's". With the current definition of delimiter you would get 'Mike' and 's' as separate tokens. If you want to have spaces and punctuation in your token list, define the delimiter like this:

Insert regex delimiting tokens (or ENTER to skip):

```
pmse> \b
```

Here we use a word boundary as the delimiting element. Now we will get:

```
120424-09:51:02.061125/1 - Output directory: STDOUT
'utcount' => {'city'      => 1, 'a'      => 1, ' '      => 6, 'above' => 1, '.' => 1,
              'Klasterec' => 1, 'is'     => 1, 'Ohre'  => 1, 'nice'  => 1},
```

The default PMLIB tokenizer is a little bit complicated to be described, but when we look at the output, it is the same as in the previous example. This is caused by the length of the text. The output would differ if we had longer texts with various characters.

You can also specify your tokens concept with other options, for instance: `ifilter` and `process`. `Process` is not implemented in the interactive mode, but you can specify it in CLI mode.

The list of tokens can differ in relation to the desired output, in other words in relation to the action you want `P_csp` to do:

Insert `utcount`, `utprob` or `utfreq` (you can specify more values separated by whitespace):

```
pmse> utcount utprob
```

For information about each action take a look at the subsection [8.5.1](#). If you specify `utcount` and `utprob`, the output will be stored in the directory specified via the `out` option. The 'utcount; and 'utprob' files will be placed in this output directory - both in Perl Storable format. You can read them by running the `P_dvf` command.

Insert filters (the format of the pairs is `key=value`, pairs are separated by ENTER):

```
pmse>
```

This option is called `ifilter`, because it filters tokens going into the process of statistical computation. If you found punctuation and spaces in your tokens list and you want to remove them, you should 'catch' them beforehand with a regexp in the `ifilter`:

```
pmse> -token=(\p{P}+|\s+)
```

The `key` part of the `ifilter` denotes the place where the filter is applied. Take a look at the figure 8.5.2. This figure represents the whole process of statistical computation. You might have one set of tokens, but you can run different operations within the set for each action.

You can also specify multiple filters. In interactive mode you are asked twice or more times for the values. If you just want one single filter, type ENTER and do not enter any further values.

There is also an 'ofilter' option available for `P_csp` in CLI mode only. The 'ofilter' acts on the output, which in our case is a hash with pairs `token => value`. If you do not want to filter the output of `P_csp` directly, you can also filter it using `P_dvf`. This can be useful when you need to display modified output, but do not wish to change the `P_csp`'s output at all.

After you complete the `ifilter` options, an overview of your task is displayed:

we have following values:

```
in => 'English',
out => 'STDOUT',
delim => qr/(?^\b)/,
action => ['utcount', 'utprob'],
ifilter => {'-token' => qr/(?^u:(\p{P}+|\s+))/?},
Is it correct? (default value: >yes<):
pmse>
```

Interactive mode then asks if the given values are correct. If you are not satisfied, type 'no' and you will be asked which values you want to change. Then you can re-enter each option again. If all options are OK, just type ENTER and your task will be computed.

8.16.4 P_gnp Interactive

This tutorial for `P_gnp` is designed to follow on from the tutorial for `P_csp`'s interactive mode. Although these two scripts share some common elements of the architecture, `P_gnp` provides more functionality, and supports a wider range of input options.

Basic options, such as `in`, `out`, `ifilter`, `process` and `delimiter` are common for both scripts.²² `P_gnp` also has an option called `cluster` which is identical to `P_csp`'s `action` - in `P_gnp` determines `cluster` what should be counted as N-grams (their basic count, frequency and probability).

²²The delimiter option **always** defines how to split tokens in the text. Tokenization is the first process in `P_gnp`, and is followed by the creation of N-grams. For a graphical explanation of the flow of available actions, which might be easier to visualize, see figure 8.10.

Taking a look at the `P_csp` interactive tutorial in subsection [8.16.3](#) is a good way to learn how to work with the basic options, and it is assumed the `P_csp` tutorial has been successfully understood, before proceeding with this tutorial.

Options specific for `P_gnp` can be divided into two groups:

- options related to N-grams creation
 1. **size** – size of the N-gram
 2. **window** – size of context from which are ripped tokens creating the N-gram
 3. **separator** – element delimiting tokens in the N-gram
- options related to computation of (lexical) association measurements
 1. **measure** – name of association measure
 2. **rank** – N-grams get ranked according to specified association measure
 3. **contingency** – compute contingency tables with probabilities of tokens and N-grams

To get detailed information about each of these options, see the section [8.10](#). Now let's take a closer look at the interactive mode. We will use the `C2` corpus again. We are looking to create a list of bigrams and we also want their count, MI-scores, and a list with MI-score ranked bigrams.

Insert the correct path to the corpus:

```
pmse> C2
VALUE: >C2<
```

You have specified the correct, case-sensitive, path to your file, right?

Insert directory or 'STDOUT' (default value: `>P_gnp.2243<`):

```
pmse> english_ngrams
VALUE: >english_ngrams<
```

Output will be stored in the 'english_ngrams' directory.

Insert regex delimiting tokens (or ENTER to skip):

```
pmse> \b
VALUE: >\b<
```

You want to delimit tokens in your text by word boundary.

Insert ngram size (default value: `>2<`):

```
pmse>
VALUE: >2<
```

If you want just bigrams, type ENTER - bigram is the default value.

Insert window size (default value: `>2<`):

```
pmse>
VALUE: >2<
```

Let's specify the contextual tokens that creates the N-gram.

Insert str separating tokens in ngram (default value: > <):

```
pmse>
VALUE: > <
```

The default value is a single whitespace. If you want something more specific, e.g.: γX^*XY , type * and press ENTER.

Do you want to output contingency tables? (default value: >no<):

```
pmse>
VALUE: >no<
```

Counting of contingency tables is not particularly fast. Especially when you have a large corpus, which is why the default is 'no' here. (Just press ENTER.)

Insert count, prob or freq (you can specify more values separated by space):

```
pmse> count
VALUE: >['count']<
```

You chose a simple count of the N-grams. The other options are `prob` (probability) and `freq` (frequency).

Insert measurements and their parameters

(format of pairs is key=value, pairs are separated by ENTER):

```
pmse> miscore
pmse>
VALUE: >{'miscore' => undef}<
```

Parameter is mandatory, when you want to count the MI-score (or other association measure) for N-grams bigger than 2. If you want trigrams, you have to specify dependencies among the tokens creating the trigram. So - if you do not need to specify the dependencies, just type `miscore`.

Insert measure names for ranks (`miscore` or `tscore`)

(you can specify more values separated by space):

```
pmse> miscore
VALUE: >['miscore']<
```

If you want the list of bigrams ranked by MI-score, just type `miscore`.

Insert `ifilter(s)`

(format of pairs is key=value, pairs are separated by ENTER):

```
pmse> +token=\w+
pmse>
VALUE: >{'+token' => '\\w+'}<
```

With this token specification, the program will be searching for tokens comprised only from alpha-numeric characters. You can give multiple filters here. When you are satisfied with your choice, just type ENTER and the next option will be launched.

Insert substitution hooks (possible keys are `_ngrams` and `ngrams_`)

(format of pairs is key=value, pairs are separated by ENTER):

```
pmse>
VALUE: >{}<
```

This option in CLI mode is called `process`. You can modify your tokens here; for instance, you can transform all tokens to lowercase. Your current corpus contains just one sentence, so you probably do not need to change the tokens. However, if you wished to change all bigrams (tokens in bigrams) to lowercase, you could do something like:

```
_ngrams={\A(.+)\s}{lc($1)}xmse
```

we have following values:

```
in => 'english',
out => 'english_ngrams',
delim => qr/(?^\b)/,
size => '2',
window => '2',
separator => ' ',
contingency => '0',
cluster => ['count'],
measure => {'miscore' => undef},
rank => ['miscore'],
ifilter => {'+token' => qr/(?^\w+)/},
process => {},
```

Is it correct? (default value: >yes<):

pmse>

VALUE: >yes<

An overview of your task is now displayed. Is everything OK? If not, just type the option that need to be changed and adjust it accordingly. If everything went well, you should see a directory called `english_ngrams` with 4 files: `count`, `miscore_rank_1|2`, `miscore_1|2` and `pmse-env`. The `pmse-env` file is a log, where you can see a record of your task. 'Count' is list of N-grams and their counts, `miscore_rank_1|2` are ranked N-grams and `miscore_1|2` are unranked N-grams with their associated MI-score values.

If you do not find it particularly user friendly to browse each file separately, you can use `$PMSE_BIN/samples/create_table.pl` to make a table:

```
$ perl $PMSE_BIN/samples/create_table.pl --in 'english_ngrams' --out STDOUT
```

This stanza will produce a CSV file:

```
data,count,miscore_1|2,miscore_rank_1|2
Klasterec above,1,2.58496250072116,1
nice city,1,2.58496250072116,3
is a,1,2.58496250072116,2
Ohre is,1,2.58496250072116,4
a nice,1,2.58496250072116,5
above Ohre,1,2.58496250072116,6
```

Which you can load into R or your favourite spreadsheet editor.

8.16.5 Categorization of EMA Texts

EMA is an abbreviation for *European Medicines Agency*²³. EMA organization provides identical documents in parallel languages. All the documents are distributed in PDF format. In this example, we will describe how to

- download the documents
- convert them
- perform text categorization for each language

Fetch the Docs

Getting the docs is easy, because the P_daf script has already predefined hook called `ema`:

```
P_daf --fetch ema
```

P_daf will store the documents in `\$PMCORP_ROOT/<iso>/original/ema` where `<iso>` is the iso-639-3 code transformed into a trie structure (see section 8.7). The default target is `\$PMCORP_ROOT` which you may change in the INI file (it's a line beginning with `store =`). E.g.: in the current EMA INI file²⁴ you will find line like this:

```
store = "$ENV{PMCORP_ROOT}/" .
        x2f_any2path({input => _1or3_639( $lang )}) .
        '/original/' . $pdf
```

To change the root, substitute `\$ENV{PMCORP_ROOT}`; with the desired path - e.g.: `/home/john/documents/ema`. The resulting path for i.e. English documents then will be: `/home/john/documents/ema/e/n/g/original/ema/`. (We will use this separate directory in order to keep this example easy.) For further info about how P_daf works read please the section 8.6. Let's presume we have fetched all the multilingual EMA documents successfully into a root directory `/home/john/documents/ema/`.

To see how many languages do we have, `cd` to that mentioned directory and type:

```
tree -L 2 -i -f * | grep '././.'
```

You should get a list of directories, which contain the ema docs:

```
b/u/l
c/e/s
d/a/n
d/e/u
e/l/l
...
```

Good. Now - all the EMA docs are stored in PDF format. We will call P_dmf and convert them into plain text files. We just need to find the 'original' directory for each language. While the P_dmf needs to get an absolute path of the input argument, we will do:

²³<http://www.ema.europa.eu/ema/>

²⁴Which is placed in `\$PMSE_ROOT/cfg/daf.d/`

```
find `pwd` -name 'original' -exec P_dmf --base \
/home/john/documents/ema --in '{} ' \;
```

This will create corresponding *derived* directories. Now it is easy to create simply shell script to execute the categorization:

```
#!/bin/bash

dirs=(f/i/n f/r/a n/l/d l/i/t
      h/u/n s/l/k s/l/v s/w/e
      s/p/a d/a/n d/e/u m/l/t
      i/t/a e/n/g e/l/l e/s/t
      p/o/l p/o/r )

for i in "${dirs[@]}"
do
    $PMSE_BIN/samples/categorization/categorization.pl\
    --root /home/john/documents/ema/$i --report 3 --cpus 8 \
    --vector frequent='count=200,distance=tanimoto,weight=1,\
    preprocess=1'
done
```

After execution of the script in all `/home/john/documents/ema/**/*` directories should appear a folder called `runs` which holds the results of the categorization run.

8.17 PMSE: Cookbook

8.17.1 Recipes for PMSE

Some recipes for several practical tasks take place in this section. We use the functionality of PMSE.

8.17.2 PMSE Crash Course

This section consists of a list of commands related to PMSE environment. Purpose of the list is to provide a quick reference on the most relevant topics / functions. You will find a detailed explanation of the commands further in the documentation.

File modifications

```
P_rer 's{replace}{with}g' file
```

```
P_trt --action 'repair_interpunction' --in filename --out STDOUT
```

```
P_vls --in textfie --out STDOUT --cut '1 +10% 10% 90%'
```

Toolchain: decompression of an archive, extraction of n-grams

```
P_dmf --in $PMCORP_ROOT/e/n/g/original/archive.tgz
```

```
P_gnp --in $PMCORP_ROOT/e/n/g/derived/archive.tgz/lvl.last/\
archive.tgz/archive.tar.gz/archive.tar.gz/archive.tar/a.txt\
--ngram 3 3 ' ' --measure 'mi=all' --measure\
'tscore=1|2|3,1 2|3' --rank miscore --cluster count --cluster prob\
--keywords reference_corpus.sbl
```

```
$PMSE_BIN/bin/samples/create_table.pl --in P_gnp.out\
--out table.csv
```

Extracting a wordlist

```
P_csp --in $PMCORP_ROOT/e/n/g/derived/archive.tgz/lvl.last/\
archive.tgz/archive.tar.gz/archive.tar.gz/archive.tar/a.txt\
--action utprob --out STDOUT
```

Categorization toolchain

```
P_dmf --in $PMCORP_ROOT/c/e/d/
```

```
nohup $PMSE_BIN/samples/categorization/categorization.pl --root\
$PMCORP_ROOT/c/e/d/ --report 3 --cpus 2 --vector\
'frequent=count=200,distance=tanimoto,weight=1,preprocess=1'` &
```

```
P_dvf --in $PMCORP_ROOT/c/e/d/runs/0/categorized.sbl --otype graphic\
--out g.svg
```

Various cmds

```
P_dmp --distance euclid --ngrams may my --out STDOUT
```

```
P_cqt --in file --action concordance --query 'e' --out STDOUT
```

```
P_dvf --in storable --out STDOUT --sort val
```

```
P_ici 'P_rer "s{use encoding (qw)?.utf-?8.}{use utf8}g" [%f]'\
`find -L . -name '*.pm'`
```

8.17.3 Sentence Segmentation

Sentence segmentation is a standard NLP problem. It is a special case of text segmentation, "sentence segmentation is the problem of dividing a string of written language into its component sentences."²⁵

The goal of the segmentation is to provide reliable detection of sentence boundaries, which can be a non-trivial task in some languages as these boundaries are denoted by characters with ambiguous meaning. PMSE uses Perls extended regular expression(s) to find these sentence boundaries. By default a newline is used as quickly recognizable/parseable sentence delimiter (one sentence per line). As the whole process is parametrizable, other forms of reorganization are possible.

Before any segmentation can occur, the text should be well prepared and UTF-8 encoded. Wrong punctuation and letter casing may lead to confusing results. `P_trt` (8.14) can be used to fix some of these problems beforehand.

Basic Segmenter

Here is an example of a short Finnish text:

```
1 - 5- vuotiaat lapset: 2,5 ml oraaliliuosta (puolet 5 ml:n
lusikallisesta) kerran päivässä. 6 - 11- vuotiaat lapset: 5 ml
oraaliliuosta (yksi 5 ml:n lusikallinen) kerran päivässä.
Aikuiset ja yli 12-vuotiaat: 10 ml oraaliliuosta (kaksi
5 ml:n lusikallista) kerran päivässä.
```

The text is formatted in some random way and our goal is to get one sentence per line:

```
1 - 5- vuotiaat lapset: ... kerran päivässä.
6 - 11- vuotiaat lapset: ... kerran päivässä.
- Aikuiset ja yli 12-vuotiaat: ... kerran päivässä.
```

We use `P_trt` to deformat the original text first, which means, the text will lose all formatting information like paragraphs, line breaks etc.

```
$ P_trt --in file.txt --action deformat --out deformatted_text
```

Now the text will have a form of one long line. `P_rer` may be used to break it at the end of sentences. We need to pass a Perl regular expression with the `s` operator, because we need to insert a specific place of the line with line break.

The base of our regexp consists of a terminal punctuation mark: `\p{STerm}`. *STerm* is a Sentence Terminal punctuation. Now follows white-space character and upper case letter:

```
qr{\p{STerm}\s+\p{Upper}}
```

The beginning of the second and third sentence would not be matched properly. Second sentence starts with a number, third with a dash. Thus we have to add:

```
qr{\p{STerm}\s+\p{P}?\s*(\p{Upper}|\d)}
```

²⁵http://en.wikipedia.org/wiki/Text_segmentation

Now we have to integrate the regexp with `s{ }{ }` construction. We will use a look ahead `(?=)` construction to tell the search engine exactly when to match the STerm character. The white-space character will be replaced with a new-line character.

```
s{(\p{STerm})\s+?(?=\p{P}?\s*(\p{Upper}|\d))}{$1\n}xmsg
```

Most scripts of the world don't have cased letters, e.g. Tamil. In such case you may replace `\p{Upper}` with `\w`.

```
s{(\p{STerm})\s+?(?=\p{P}?\s*(\w|\d))}{$1\n}xmsg
```

In some scripts (e.g. in Chinese or Japanese), whitespace does not occur between the terminal punctuation mark and the new sentence. Also, the full stop/period is a special character, namely U+3002. We can take advantage of this and apply a special rule.

```
s{(\N{U+3002})}{$1\n}xmsg # very basic
```

Complex Segmentator

We can have a text with combination of punctuation marks. Or we just want a more robust segmentator:

```
s{(\p{STerm})(?(?<=\N{U+3002})(.))}{$1\n}xmsg
```

We use condition combined with "look behind" construction. The meaning is "match any character, if the STerm char is U+3002". This regexp can be extended:

```
qr{(\p{STerm})(?(?<=\N{U+3002})(.))|\s+?(?=\p{P}?\s*(\p{Upper}|\d))}
```

It means: "match any STerm character; if it is the U+3002 character, match any following character, else the following character(s) must match a combination of white-space, punctuation and upper case letter or digit".

Now we need to integrate the regexp into `s{ }{ }` construction. We will use named backreferences to identify captured sequences:

```
s{
  (?<TERM>\p{STerm})
  (?(?<=\N{U+3002})(?<SENT>.)|\s+?(?<SENT>\p{P}?\s*(\p{Upper}|\d)))
}{
  ${TERM}\n${SENT}
}xmsg
```

The environment before the STerm character also affects the segmentation. The full stop character in latin scripts may be combined with digits (to express ordinality)²⁶ or it is a part of abbreviations. We don't want to match the STerm character in this context, thus we will use "negative lookbehind" condition in the regexp.

```
qr{(?<!\d)\p{STerm}}
```

²⁶This occurs in several European languages: in Croatian, Czech, Danish, Estonian, Faroese, German, Hungarian, Icelandic, Latvian, Norwegian, Polish, Slovak, Slovene, Serbian, Turkish. http://en.wikipedia.org/wiki/Ordinal_indicator

Will exclude the context of the STerm character standing after a digit. The negative lookbehind construction can be extended with other non-wanted contexts:

```
qr{(?<!\d)(?<![jJmM][rs]s?)(?<![aA]bbr)\p{STerm}}
```

You can specify as much abbreviations as you want. The complexity of the `s{}` construction will grow:

```
s{
  (?<!\d)(?<![jJmM][rs]s?)(?<![aA]bbr)
  (?<TERM>\p{STerm})
  (?(?<=\N{U+3002})(?<SENT>.)|\s+?(?<SENT>\p{P}?s*(\p{Upper}|\d)))
}{
  ${TERM}\n${SENT}
}xmsg
```

Advanced Segmenter for Czech

The regular expression grows with the list of abbreviations you want to apply. In such a situation, it is handy to use an INI file for `P_rer`, because you will avoid troubles with the CLI.

You can find an example of such an INI file below. Please note that the `i` section is formatted by newlines to improve readability of this document. The `i` section must be formatted as one line in the real INI file.

```
[sentence]
i = (?<!\s(č|f|m|n|p|r|s))
i = (?<!\s(aj|ak|ap|Bc|bl|br|čl|dl|ev|fr|hl|ie|it|kr|kř|
  mj|ml|ms|ob|pf|pl|sg|sl|tj))
i = (?<!\s(abl|adj|adm|adv|akt|arg|atd|atp|att|bás|BcA|boh|
  bot|býv|CSc|csl|dán|dat|děj|
  dep|des|dět|DiS|doc|dol|dop|dór|fam|fem|fil|fin|
  fot|fut|fyz|gen|hod|hor|hud|hut|imp|ind|inf|Ing|
  ión|jap|kpt|lat|lék|les|lid|lit|log|lok|mat|MgA|
  Mgr|mjr|mld|mod|náb|nám|něm|než|niz|nom|nor|odd|
  odp|opt|pas|plk|pol|por|rak|reg|rkp|rtm|rtn|rum|
  rus))
i = (?<!\s(alch|amer|anat|angl|apod|arab|arch|astr|belg|bibl|biol|
  brit|bulh|círk|dial|dopr|dosl|ekon|epic|film|form|geol|
  geom|germ|gram|hebr|hist|horn|chem|chil|impf|iron|JUDr|
  klas|kniž|komp|konj|kuch|metr|MUDr|MVDr|mysl|např|npor|
  nrtm|obch|obyč|odst|ojed|part|pers|PhDr|plpf|pomn|popř|
  pplk|ppor|prap|práv|prep|prof|rcsl|refl|resp|RNDr|RSDr|
  slov))
i = (?<!\s(absol|eufem|event|geogr|hovor|chcsl|instr|kanad|konkr|
  námoř|nprap|pejor|pprap|předl|přivl|slang|s\.r\.o))
i = (?<!\s(archit|astrol|genmjr|genplk|genpor|herald|interj|liturg|
  meteor|neklas|nstržm|samohl))
```

```
i = (?<!\s(etnonym|indoevr|katalán|nesklon|PharmDr))
i = (?<!\s(anglosas))
i = (?<sep>\p{STerm}\p{QMark}?\s?)\s*(?=\p{QMark}?\s?\p{Upper})
o = ${sep}\n
m = xmsg
```

Resulting regexp takes advantage of multiple negative look-behind conditions. Each of these concatenates abbreviations (in 'OR' relation) of the same length.

Abbreviations of titles²⁷ and abbreviations of textual units e.g.: *par. 6* or *par. F* would cause false boundary match.

The INI file may be applied with `P_rer` like this:

```
$ P_rer --bulk ssegmentation.ini text.txt
```

The INI file mentioned above was created by `builddrx_neglookbehind_from_abbrevs`²⁸ - a help script that builds complex INI files from list of abbreviations. Please note that you may use multiple `i` and also `o` sub-sections for `P_rer`'s bulk file. Also, you may use multiple sections. `P_rer` has option `--sect` for specification of particular section.

Abbreviations are placed in `$PMSE_ROOT/cfg/P_rer/abbreviations`. You will find there abbreviations for all major languages in EU. Resulting INI files are stored in `$PMSE_ROOT/cfg/P_rer/segmenter`.

8.17.4 Sub Word N-grams Extraction

In this manual, a *sub word n-gram* is a string of characters smaller than a word. subword N-grams are capable to describe repetitive patterns of graphemes. Formally, a subword n-gram is just a substring of the input text.

In other words: it is an n-gram, but the units, from which the subword n-gram consists, are not word-like tokens, but graphemes. E.g.: the word *grapheme* consists of these subword n-grams of size 2:

```
gr ra ap ph  # inside a word
he em me

_g          # over the word boundary
e_
```

The subword n-gram may occur in several positions: inside the word, on the word boundary and over the word boundary. If the length of the subword n-gram is large, it may consists of several words. We will mark the white-spaces, in order to identify (cross) boundary subword n-gram.

First of all, deformat the text to remove indention and multiple white-spaces.

```
$ P_trt --in text.txt --action deformat --out deformatted
```

²⁷E.g.: titles stand often before the names in the Czech language, e.g.: PhDr. Jan Novák.

²⁸This script is placed in `$PMSE_DEVBIND` directory.

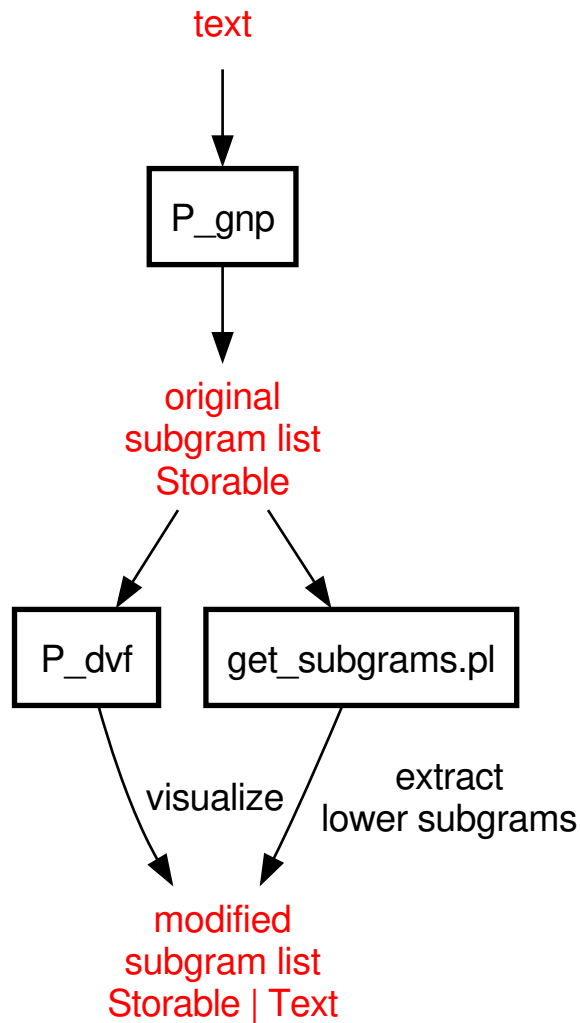


Figure 8.17: Schematic overview of subgrams extraction

Now apply `P_rer` on the deformatted text. We want to mark white-spaces

```
$ P_rer 's{\s}{_}xmsg' deformatted/repaird
```

and remove the punctuation:

```
$ P_rer 's{\p{P}}{}xmsg' deformatted/repaird
```

Now all the white-spaces transformed into the underscores. (The marker is - as usual - arbitrary. You can use white-spaces as well.) Now we need to insert underscores in the beginning and the end of the text. You should insert "length of the subword n-gram" - 1 of markers. If the length of the subword n-gram would be 4, you need to insert 3 markers in the beginning and 3 in the end.

```
$ P_rer 's{\A(.*)\z}{___$1___}xmsg' deformatted/repaird
```

The text is prepared for processing. We will use P_gnp:

```
$ P_gnp --in deformatted/repaird --out subgrams --delimiter ' ' \
> --cluster count --ngram 4 4 ''
```

The resulting Storable file will be stored as `subgrams/count`. The subword n-grams of length 4 allow you to distinguish the minimal (length 2) subword n-grams in all positions. Consider a subword n-gram 'of'. It can occur as a self standing word (preposition), on the word boundary, or inside the word:

```
_of_      # preposition
eof_      # word boundary (thereof)
rofi      # inside word (profile)
```

You can select/filter the subword n-grams with P_dvf (filter option), or set 'ofilter' option in P_gnp. We will use P_dvf to get subword n-grams occurring **in** the boundaries of a word:

```
$ P_dvf --in subgrams/count --filter '$key =~ m{_\}xms' \
> --out subgrams.sbl --otype storable
```

The resulting file `subgrams.sbl` (in Storable format) consists still of subword n-grams of length 4. To extract subword n-grams of length 2, use auxiliary script `get_subgrams.pl`, which is placed at `$PMSE_BIN/samples/get_subgrams.pl`.

Idea behind the script is easy: provide a regular expression, which will be applied on each subword n-gram of the list. You can match only the type of subword n-grams you want to work with. (The filtering step with P_dvf is not necessary.) The original subword n-gram will be re-created and its frequency re-counted. The output is the new subword n-gram list with re-counted frequencies stored in a Storable file. The subword n-gram list may be printed on STDOUT.

```
# for filtered subgram list:
```

```
./get_subgrams.pl --in subgrams.sbl --regexp '\w(?<sgram>\w{2})\w'
```

```
# for non-filtered subgram list:
```

```
./get\_subgrams.pl --in subgrams.sbl \
--regexp '\A[^\_](?<sgram>[^\_]{2})[^\_]\z'
```

The mentioned command line example(s) will print in-word n-grams of length 2 (and their counts) on STDOUT. Note: While generating the original subgram list, it is important to specify `--cluster count` option (in P_gnp). Only basic counts of subword n-grams may be re-counted.

If you want to get the the probability of subword n-grams, specify `--action prob` when calling `get_subgrams.pl`.

8.17.5 Probability of Neighbors

This recipe will show how to extract probability of occurrence of tokens on nearest contextual positions. Assume following text²⁹:

He was the third child of eight and the eldest surviving son. Shakespeare produced most of his known work between 1589 and 1613. Many of his plays were published in editions of varying quality and accuracy during his lifetime. In 1623, John Heminges and Henry Condell, two friends and fellow actors of Shakespeare, published the First Folio, a collected edition of his dramatic works that included all but two of the plays now recognised as Shakespeare's. Shakespeare was buried in the chancel of the Holy Trinity Church two days after his death.

Now we would like to know probability of tokens occurring on the first position after preposition *of*, which has 8 occurrences in the sample text. There exist namely these pairs:

```
of eight
of his
of his
of varying
of Shakespeare
of his
of the
of the
```

The probability of co-occurrence *of eight*, is 1/8, prob. of *of his* is 3/8, *of the* is 2/8 etc. PMSE has a script called `get_context_probability` that is intended exactly for this operation. The script is placed in `$PMSE_DEVBIN`. It needs two input files: file storing count of bigrams and file storing probability of unigrams (generated from the same source as the file with bigrams). Both input files should be generated with `P_gnp`, see section 8.10. Output is a data-structure where each unigram has a predecessor (marked as `°+1`) and successor (marked as `°-1`) position. Count of tokens on each position may be also set (default is 5) as well as output limit of "root" unigrams (default is 100,000).

```
of => {                                     # root unigram
  P   => 0.0860215053763441,               # probability of root unigram
  °+1 => {                                   # predecessors
    Many   => { P => 0.125 },
    actors  => { P => 0.125 },
    child   => { P => 0.125 },
    edition => { P => 0.125 },
    most    => { P => 0.125 },
  },
  °-1 => {                                   # successors
    Shakespeare => { P => 0.125 },
    eight      => { P => 0.125 },
    his        => { P => 0.375 },
    the        => { P => 0.25  },
    varying   => { P => 0.125 },
  },
}
```

²⁹It is a sample of an article about William Shakespeare published on Wikipedia: <http://en.wikipedia.org/wiki/Shakespeare>

```
    },
}
```

Help for script `get_context_probability` may be invoked as in other PMSE scripts:

```
$ perl get_context_probability -?
```

Output (data structure similar as listed above) is stored in Storable format, it is `PMSE::Visualize::Neighbors` object and has some predefined method of conversion, currently to text and YAML. You can do that with `P_dvf`, see section: [8.9](#).

8.17.6 Co-occurrences

Our concept of co-occurrences is similar to `kocos.pl` in `Text::NSP`³⁰:

Co-occurrences are the words which occur together in the same context. All words which co-occur with a given target word are called its co-occurrences. The concept of 2nd order co-occurrences is explained in the paper Automatic word Sense Discrimination [Schutze98]. According to this paper, the words which co-occur with the co-occurring words of a target word are called as the 2nd order co-occurrences of that word.

The words which co-occur with the 2nd order co-occurring words belong to 3rd order and so on.

What is a Co-occurrence in Linguistics?

The linguistic term of *co-occurrence* is related to the term of *collocation*. Generally speaking, collocation and also co-occurrence provide information about context of a word, or - by collocation - about multi-word units. The strength of the collocation is derived from the probabilities of occurrence its components. The measures of collocational strength take into account the mutual position of the lexical units and the separate occurrence of the units in the corpus.

The concept of a co-occurrence is different. It describes the relation of a word to other lexical units but without a dependency on absolute position of the word in the text.

If we have e.g. following sentences from which we want to extract basic bigrams (n-grams of size 2 from window 2):

I like Chinese language. It is a rich language.

```
I like
like Chinese
Chinese language
language It
It is
is a
a rich
```

³⁰Available from <http://search.cpan.org/dist/Text-NSP/lib/Text/NSP.pm>

rich language

Then co-occurring words for *Chinese* are:

Chinese target

like 1st order

language 1st order

rich 2nd order

There exists dependency chain among *Chinese - language - rich*. Note that *Chinese* and *rich* stay far from each other in the text.

Extract Co-occurrences

Then we will use `P_cqt` to extract the co-occurrences and then we will visualize them with `P_dvf`.

We need to extract bigrams first, thus we will use `P_gnp`. Let the the input file be 'source.txt':

```
$ P_gnp --in source.txt --cluster count --out bigrams --ifilter '-token=\s+'
```

Now, do some filtering³¹ and store the bigrams as non-PMSE object:

```
$ P_dvf --in bigrams/count --out bigrams.sbl --otype storable \  
> --filter '$value < 100 || $key =~ m{\p{P}}xms'
```

Finally, extract the co-occurrences with `P_cop`:

```
$ P_cop --in bigrams.sbl --level 2 --target 'literal=Chinese' --out coocs
```

And display the result with `P_dvf`:

```
$ P_dvf --in coocs/cooccurrences --otype graphic --out cooc
```

`P_dvf` should create a file called `cooc.svg`, which should look like picture [8.17.6](#).

The target (*say*) is marked with white color. Co-occurrences of first order are marked by red, co-occurrences of second order are marked by orange colour. The word *have* is connected with two words, because it co-occurs with both of them.

Convert Text::NSP Bigrams to PMSE

Text::NSP use following format to store bigrams:

```
11  
line<>of<>2 3 2
```

³¹The filtering step is necessary, if you want to do a qualitative inspection of interesting relations among words, you will need to filter out the bigrams containing punctuation and probably grammatical words. Also, you will need to filter out bigrams with low frequencies, because they would cause the graphic output to be poorly arranged.

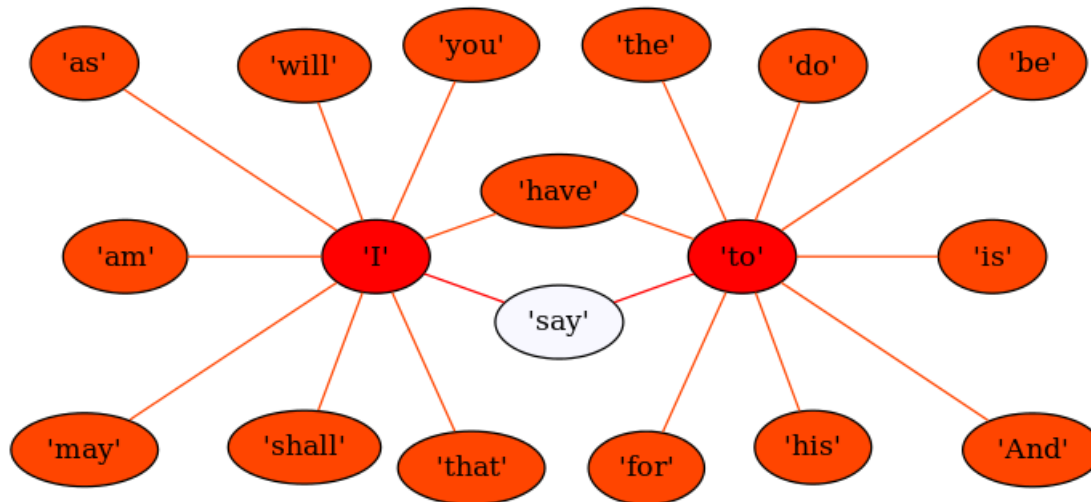


Figure 8.18: The Canterbury Tales, and Other Poems by Geoffrey Chaucer - grammatical co-occurrences for 'say' (filtered input)

```

of<>text<>2 2 2
second<>line<>1 1 3
line<>and<>1 3 1
and<>a<>1 1 1
a<>third<>1 1 1
first<>line<>1 1 3
third<>line<>1 1 3
text<>second<>1 1 1

```

This list can be stored e.g. in a file called 'nsp.txt'. We need to get a format which we can load in P_dvf. Thus we will use P_rer to convert the NSP list:

```
P_rer 's{^(.+?)<>(.*?)<>(\d+?)\s.+?}$}{$1<>$2\t$3}xmsg' nsp.txt
```

We get a list like this one:

```

11
line<>of 2
of<>text 2
second<>line 1
line<>and 1
and<>a 1
a<>third 1
first<>line 1
third<>line 1
text<>second 1

```

The last thing we need to remove is the total count of bigrams at the beginning of the file:

```
P_rer 's{^\d+?\n}{xms}' nsp.txt
```

Now we can load the list into P_dvf:

```
$ P_dvf --in nsp.txt --itype text --otype storable
```

The result is a histogram-like structure in the Storable format.

8.17.7 Text Categorization

The categorization of texts (TextCat) is a use-case for PMSE. It was intended as a simply layer of glue connecting parts of the environment first. However, it has developed in the regular procedure which integrates several parts of PMSE as well as several modules in the libraries.

Great attention was paid for efficiency as well for modularity of the system. ³²

Brief Description of the Procedure

First of all, let us consider a directory structure as is described in the section 8.1.2. The first step is to convert the source files into a plain text. Consider a categorization of Czech text.

The path to the texts will be: \$PMSE_ROOT/c/e/s/original. The conversion is a task for P_dmf:³³

```
$ P_dmf --in /data/library/c/e/s/original
```

When we have created the *derived* directory and the text files, we can run the categorization.

```
$ $PMSE_BIN/samples/categorization/categorization.pl \  
> --root $PMSE_ROOT/c/e/s --report 3 --cpus 2 \  
> --vector frequent = 'count=200,distance=tanimoto,weight=1,preprocess=1'
```

If the categorization process was completed successfully, we should see directory called *runs*. In this directory are stored runs of the TextCat. Each run has own directory called by number, first run of categorization will be stored in directory 0. Input options for each run are reported in *categorization.env* file, which is included in all run directories. You can display the file with:

```
$ P_dvf --in categorization.env
```

Now go into the *runs/0* directory and list all files. You should see a file called *categorized.sbl*. Load this file into P_dvf to visualize it. This file contains the result of categorization run. It is a Storable file containing a binary tree: the structure of clusters based on the lexical proximity of the input texts.

Command below will create a file *g.svg* in the SVG format.

```
$ P_dvf --in categorized.sbl --otype graphic --out g.svg
```

To get detailed info about the output data structure, see section 8.17.8.

³²Instructions how to write a TextCat module are placed in the PMLS manual.

³³You have to use the absolute path.

Categorization.pl - Interface for TextCat

The script *categorization.pl* is rather a wrapper of various PMSE functions than a regular PMSE script. Therefore it is placed in `$PMSE_BIN/samples/categorization` directory and does not have a regular PMSE name. The function of the script is to provide a simply interface for the process of textual categorization.

The script has several options:

categorization.pl Categorize texts

SYNOPSIS

```
categorization.pl OPTIONS
```

OPTIONS

-analyze

Count Entropy and Purity measure for the clusters of the graph.

This assumes you know the categories of texts BEFORE categorization. To get correct results of cluster evaluation, you have to name your input texts correctly. This function is designed for text-names like:

```
perl-1.txt
```

Where 'perl' is a name of the category. Only this string will be taken as a category name. The categories are recognized automatically from the names of the texts. The dash '-' and '.txt' suffix are mandatory.

-cpus <n>

Number of threads that should be using during categorizatio process. 0 means no fork.

-filter <hash>

Defines filtering tokens/files. Default no filtering.

```
files=pareto
tokens=pareto
```

-report <level>

Reporting level used in PMSE scripts. Levels could be usually 1, 2 or 3.

-root <directory>

Directory is of the form

```
*././././
```

and it contains derived directory. Inside there is a file structure with .txt files representing corpora. This structure could be created by `P_dmF`. By default `/data/library/m/u/l/`.

-vector <hash>

Defines categorization criteria. For most common frequent choice we have following parameters:

```
count      number of frequent ngrams
distance   name of distance measure
preprocess what should be considered: 1 for unigrams etc.
```

We have also few universal parameters:

```
weight     importance of this criterion
```

Default frequent='count=200,weight=1,distance=tanimoto,preprocess=1'

Example with 2dimensional vector:

```
-vector frequent='weight=0.8,count=200,preprocess=1,\
distance=tanimoto'
-vector slength='weight=0.2,...'
```

Note: combinations of more --vector criteria is not implemented.

8.17.8 PMSE Visualization

Main visualization tool of the PMSE framework is the P_dvf script. However, the process of visualization begins earlier, when the particular data-structure is being created. Each data-structure created in the PMSE framework is an *object* which may have a specific set of properties or methods.

Objects In PMSE

PMSE objects have pre-defined methods for visualization and format conversion.³⁴ This is handy, because the input data structure may be loaded in the P_dvf script and there will be the data structure automatically converted or visualized according to predefined methods (and input options).

The objects share several common visualization methods, but even so - the difference in the structures affects the attributes and options of these methods. This section of cookbook is intended to provide detailed info about the objects and their methods specifics - visualizing options.

Input from the Outer Space

The visualization of data created by other applications is possible. PMSE will load in the data structure and will create a basic object with five basic methods of visualization:

- printed data structure P_dvf -otype pdump
- storable P_dvf -otype storable

³⁴In the current version, PMSE stores objects information both about visualization and conversion to other formats. This may be changed in the future.

- text P_dvf (text is default otype option)
- yaml P_dvf --otype yaml
- SVG graphic P_dvf --otype graphic

These methods are default for all PMSE objects, however - the complexity of the graphical output is slightly different. The basic PMSE object is a simply hash with a data section which is printed out in few possible formats.

Binary Tree Visualization

Binary tree is a common data structure; in case of PMSE, it is formed by categorization process - see subsection 8.17.7 for details. The output structure consists of clusters - groups of texts which represent the similarity among texts.

PMSE uses the GraphViz tool to visualize this structure. Now consider a simple example. We performed a categorization of 8 texts. The input data stored in `original` directory was like this:

```
original/
--text/
  --Dinoland-0.txt
  --Dinoland-1.txt
  --MarkStone-2.txt
  --MarkStone-3.txt
  --PerryRhodan-4.txt
  --PerryRhodan-5.txt
  --RenDhark-6.txt
  --RenDhark-7.txt
```

After we ran `categorization.pl` script, we should see a directory `runs/<number of run>`. In this directory should be place file called `categorized.sbl`. We can display it simply as:

```
$ P_dvf --in categorized.sbl
```

What should give us something like:

```
{
0 "text/PerryRhodan-5.txt/lvl.last/PerryRhodan-5.txt/PerryRhodan-5.txt",
1 "text/PerryRhodan-5.txt/lvl.last/MarkStone-3.txt/MarkStone-3.txt",
2 "text/PerryRhodan-5.txt/lvl.last/MarkStone-2.txt/MarkStone-2.txt",
3 "text/PerryRhodan-5.txt/lvl.last/RenDhark-6.txt/RenDhark-6.txt",
4 "text/PerryRhodan-5.txt/lvl.last/Dinoland-0.txt/Dinoland-0.txt",
5 "text/PerryRhodan-5.txt/lvl.last/PerryRhodan-4.txt/PerryRhodan-4.txt",
6 "text/PerryRhodan-5.txt/lvl.last/Dinoland-1.txt/Dinoland-1.txt",
7 "text/PerryRhodan-5.txt/lvl.last/RenDhark-7.txt/RenDhark-7.txt",
8 [
  [0] 6,
  [1] 4
],
9 [
```

```

        [0] 7,
        [1] 3
    ],
10 [
        [0] 2,
        [1] 1
    ],
11 [
        [0] 5,
        [1] 0
    ],
12 [
        [0] 11,
        [1] 9
    ],
13 [
        [0] 12,
        [1] 10
    ],
14 [
        [0] 13,
        [1] 8
    ]
]
}

```

The numbers in the first column express the clusters. The lowest clusters 0 - 7 consist of "leaves" - single texts which are combined in higher clusters until the last super-cluster (root) - 14 is reached. The command for forming of the graphical output is:³⁵

```
$ P_dvf --in categorized.sbl --otype graphic --out g.svg
```

It will give us a graph like:

The look of the graph is specified in `PMSE::Visualize::BinaryTree` module. There exist two additional features of the graph:

First is the analysis of the cluster(s). If the `--analyze` option in `categorization.pl` is in use, values for entropy and purity measure will be counted. These measures are used to evaluate the "quality" of the cluster(s) - both measures take in account the ratio of categories contained in the cluster. The ideal value of purity is 1, entropy - on the opposite - has ideal value 0. You have to know the distribution of texts in given categories before you start the categorization.

In order to count entropy and purity, it is necessary to use appropriate names of the input texts. The name of text should contain number of category to which the text belongs, a hyphen, number and the `.txt` suffix. The example from this section shows a group of 8

³⁵The SVG output format is available only from Perl version 5.14.2 and higher. If the user runs lower version of Perl, the output will be stored in a dot format. For explanation of dot see e.g.: http://en.wikipedia.org/wiki/DOT_%28graph_description_language%29

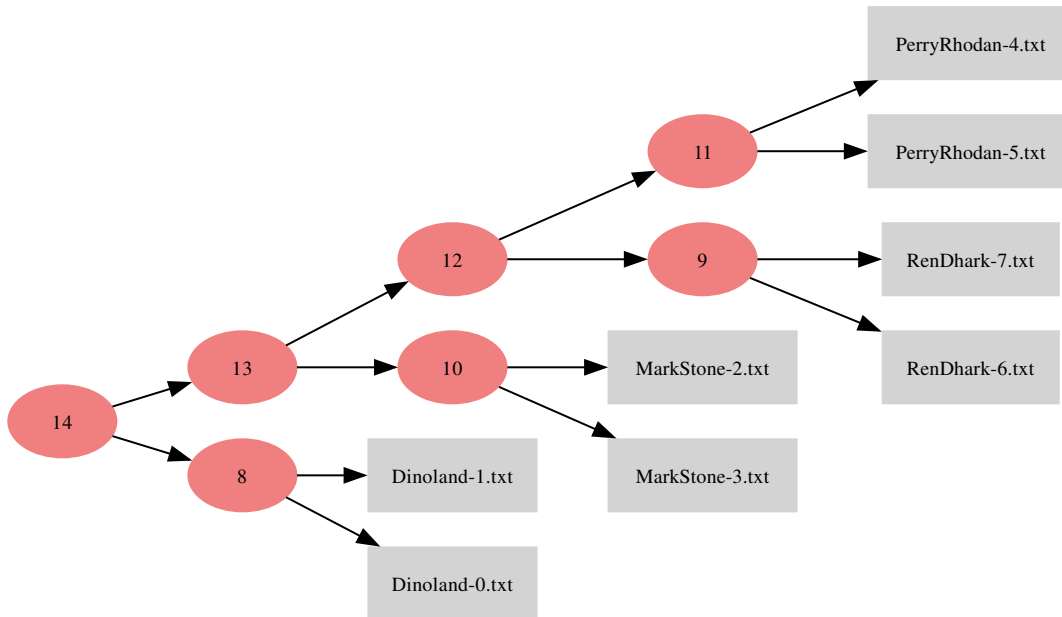


Figure 8.19: Example of clusters visualized by GraphViz

texts divided into 4 categories (4 science-fiction series). Each category consists of 2 (pulp-booklet) novellas belonging to a specific series.

Second the box-like nodes which contain the name of text should contain a link (a path to the file in your file system) aiming on the file. If you open the SVG graphic in a web browser, the name of the text should be clickable. You can easily inspect the content of the text files.

8.17.9 Visualization of Contingency Tables

Contingency table is another type of output created by P_gnp. Contingency tables in PMSE store information about the n-grams generated from given input file. The table stores information about frequency of occurrence of tokens (and their possible combinations) from which is the n-gram comprised. In the case we process n-grams with n bigger than 2, the tables are n-dimensional.

```
$ P_gnp --in <some_text> --contingency --out cont --ifilter '+token=\w+'
```

P_gnp stores the contingency table as a Storable file.

Interpretation of the Data Structure

Consider we have following sentence: *I am home here*. If we generate bigrams from this sentence, we will get a hash like this:

```
my $contingency = {
  'I am'      => [1, 0, 0, 2],
  'am home'   => [1, 0, 0, 2],
```

```
'home here' => [1, 0, 0, 2],
};
```

The left side (keys of the hash) of the table holds the particular n-gram. The right side (values of the hash) contains an array filled with values of occurrences of combinations of tokens from the n-gram.

The combination of tokens is encoded as a binary number in the position of the value in the array. For example the first row of the table is:

```
'I am'      => [1, 0, 0, 2],
```

The first value from the array has in Perl index 0. Decimal 0 is in binary also 0 and 0 denotes the true existence of the token in the n-gram. The first value in the array therefore denotes, that the the combination *I am* occurs exactly once in the input text.³⁶

The second number in the array: it has index 1. We can interpret it as 01³⁷ in the binary form. This denotes combination of *I* with any other possible token (except *am*) from the input text - we see, that there is no such combination in the input text, therefore it has value 0.

The third number in the array has index 2 - in binary form 10. This combination denotes combination of any token (except *I*) and *am*. We see no *non-I am* combination exists in the input text.

The fourth number has index 3 - in the binary form 11. This means any combination of *no-I* and *non-am* in the input text. We see we have two such combinations: *am home* and *home here*, therefore the value is 2.

This encryption of combinations is universal for n-grams of any length.

Methods of Visualization

in P_dvf:

As Text / CSV The basic method is as a plain text. No output format specification is needed, because the plain text is a default output format of P_dvf. In the case of contingency tables, the output format is rather a CSV, because it is more practical.

```
$ P_dvf --in cont/contingency --out cont.csv
```

The output will look like the following:

```
,home,not home
am,1,0
not am,0,2
,here,not here
home,1,0
not home,0,2
```

³⁶The binary 0 could be interpreted as 00 - TOKEN TOKEN.

³⁷TOKEN - NONTOKEN. NONTOKEN means, that we assume any other token except the real one existing on given position.

```
,am,not am
I,1,0
not I,0,2
```

Now you can load cont.csv e.g. into a spread-sheet editor or R.

SVG Graphic To visualize a contingency table, we use R with the combination of **vcd** package.³⁸ We use a mosaic plot (in SVG format) for the visualization, however this functionality is a little bit experimental for now. The command to get the graphic is:

```
$ P_dvf --in cont.sbl --out cont.svg --otype graphic
```

The readability of the graph depends strongly on the count of rows in the contingency table. If there is a lot of them, the graph won't be readable. Therefore you may need to filter them.

The filtering is specified via a Perl code - you can specify the keys (n-grams) and values (the real frequency of occurrence of the given n-gram) you want to remove from the table.

```
$ P_dvf --in cont.sbl --out cont.svg --otype graphic --filter \
> '$key !~ m{\bI\b}'
```

The picture 8.17.9 shows three most frequent bigrams extracted from Chaucer's Canterbury Tales. Each colored box of a row shows the proportion of frequency of occurrence for given combination of tokens.

Distance Visualization

The input data structure is a nested hash consisting of a name of given distance measure(s), info about normalization, compared n-grams and the value(s) of given distance measure(s).

The basic methods of visualization are `spreadsheet` and `text`.

```
$ P_dvf --in dir/distance --otype spreadsheet --out distance.csv
```

FileStat Visualization

The input data structure is a simply hash with statistical information related to a specified text. The basic methods of visualization are `spreadsheet` and `text`.

```
$ P_dvf --in dir/overview --otype spreadsheet --out overview.csv
```

8.17.10 N-grams Histogram Visualization

Consider a following data structure:

```
...
the<>language 3
```

³⁸<http://cran.r-project.org/web/packages/vcd/index.html>

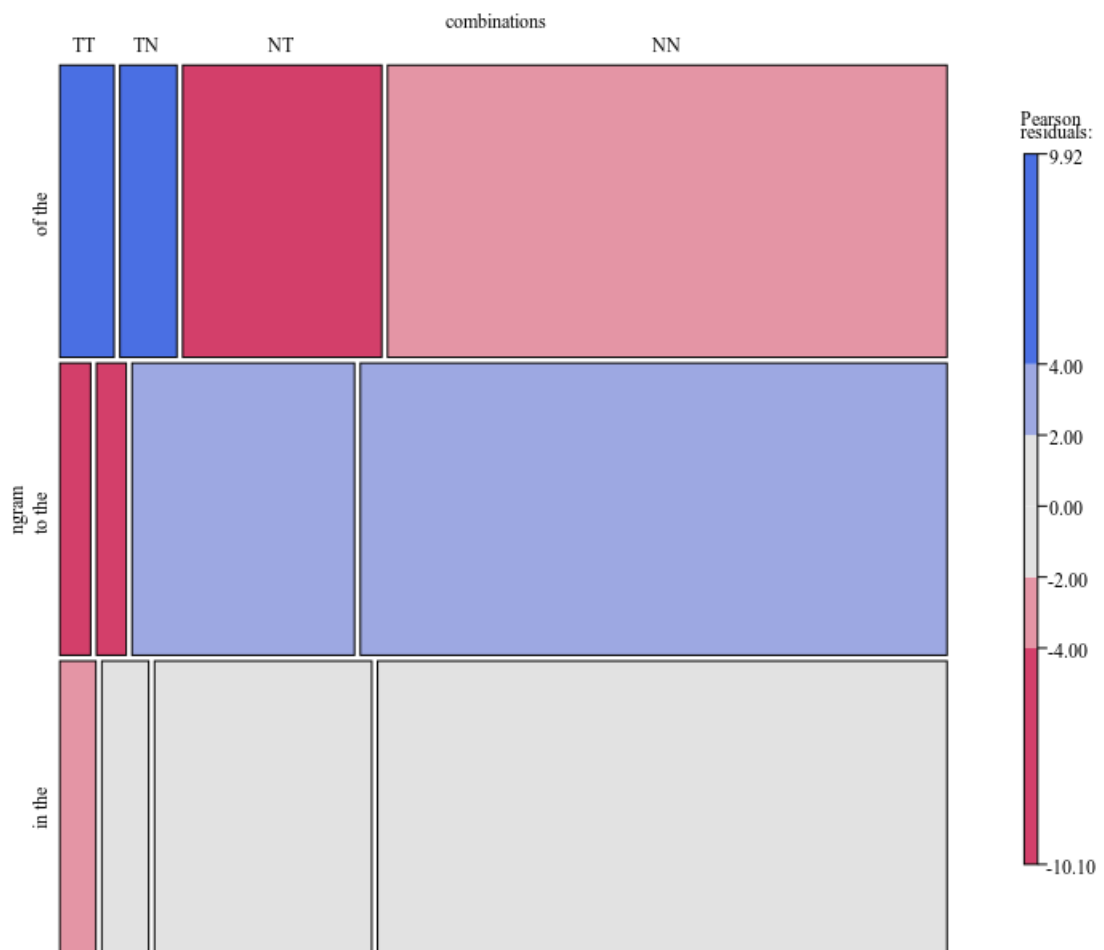


Figure 8.20: The Canterbury Tales, and Other Poems by Geoffrey Chaucer - three most frequent bigrams

```
language<>is 3
to<>use 3
agree<>on 3
programming<>in 3
ten<>percent 2
one<>can 2
which<>ten 2
...
```

These are n-grams generated by P_gnp, e.g.:

```
$ P_gnp --in text.txt --cluster count --ngram 2 2 '<>' \
```

```
> --ifilter '+token=\w+' --out try
```

And visualized with P_dvf:

```
$ P_dvf --in try/prob --sort '+val'
```

P_dvf offers multiple output formats, 'graphic' among others. Word list or a N-gram list has a predefined graphic format - a word cloud. Thus when you give following command:³⁹

```
$ P_dvf --in try/count --otype graphic --out graph
```

You will get an svg file called 'graph.svg', similar to this:

Histogram Visualization Commands

Histogram has several specific methods - options:

- **bulk**: building of more complex regexp for filtering, e.g.: stop-lists
- **filter**: filter keys or values
- **fmt**: specify key - value position on the line
- **limit**: limit the number of lines with output
- **sort**: sort keys or values

Exampmples of commands for P_dvf related to histogram:

```
P_dvf --otype pdump                # printed data
                                   # structure
P_dvf --sort <sort_order> --fmt <fmt> # text
P_dvf --sort <sort_order> --otype yaml # yaml
P_dvf --filter <code>
P_dvf --otype graphic              # svg
P_dvf --bulk <ini file>
```

Here follows short description of histogram-specific options:

-**bulk** <file>

The bulk specifies filtering options. It is handy for specifying of stop-lists.

Define a bulk file with code that will be applied during filtering. We can achieve multiple filtering options with this. There must be defined at least one [section] called 'eval' and hook called 'filter':

```
[eval]
filter = <<END
<code>
<code>
END
```

³⁹This is an early implementation only. If you want to use the 'graphic' output format, you have to generate the input file with P_gnp -object option.

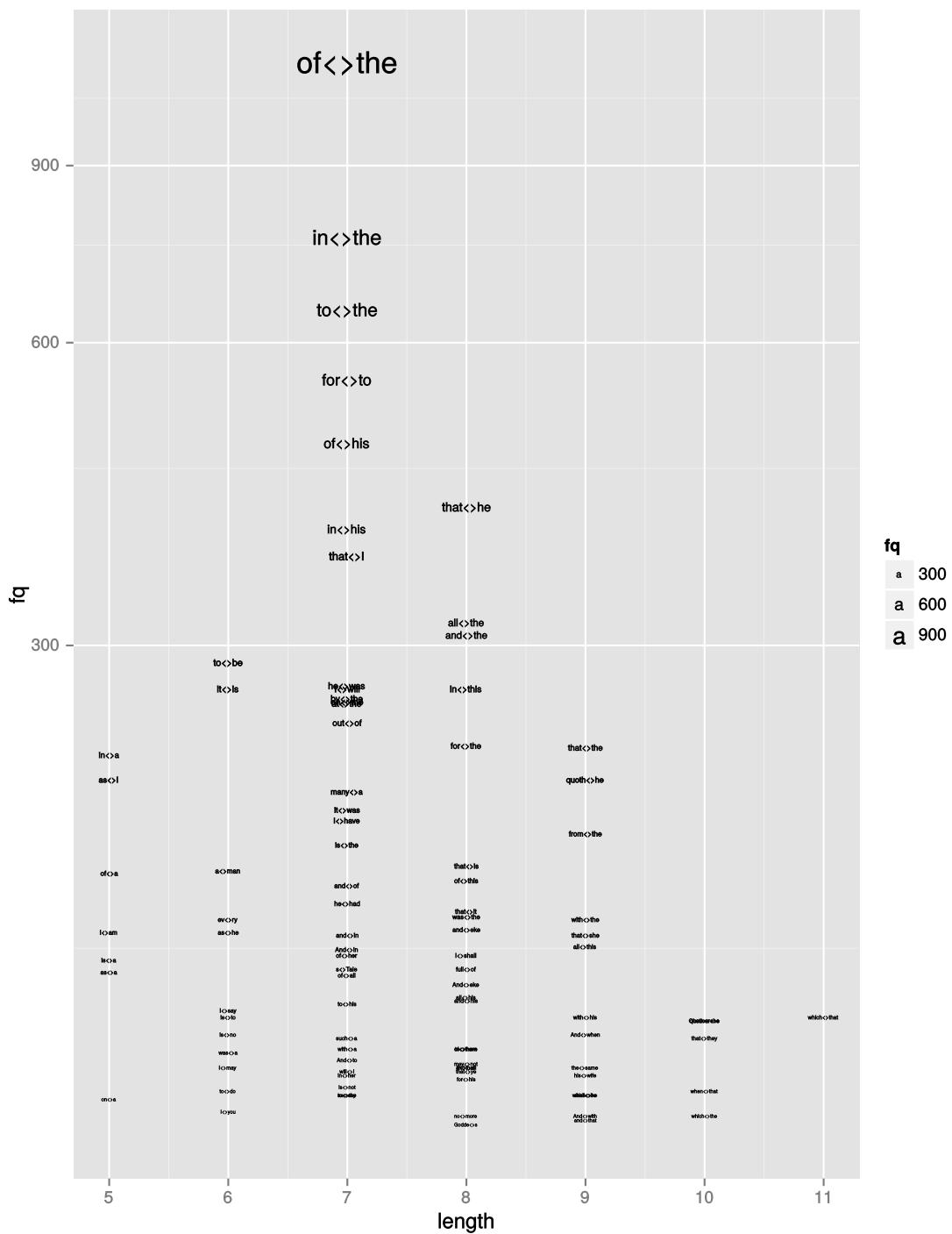


Figure 8.21: The Canterbury Tales, and Other Poems by Geoffrey Chauce: most frequent bigrams

We can use heredoc style to define multiple code blocks. To specify the code, look at the option `--filter`.

-filter <code>

Filter is an optional choice independent on the output format.

Filtering is done via perl code. As all output data is stored in a hash (list of 'key-value' pairs), the user can define filtering expressions like:

```
$key    =~ <regex>
$value  > <value>
```

Data which matches the regex, or pass the comparison, are deleted.

-fmt <template>

Fmt is optional - available only for text (default) output format.

Format string to define the output. The variables %k and %v denote positions of the current key and value. If not given, the default is %k \t %v\n.

-limit <number>

Limit is optional - available only for text (default) output format. Limit the number of output lines to C<num> (integer). The default is 0, in which case the output is unlimited. If this parameter is defined just the first C<num> lines are printed.

-sort <sort_order>

Sort is available for otype(s):

```
as_printed_data_structure
as_yaml
as_text (default)
```

Sort is optional. Valid values for <type> are:

```
+key  sort by key in ascending order (default)
-key  sort by key in descending order (default)
+val  sort by value in ascending order
-val  sort by value in descending order
```

8.17.11 Macros

The idea of a PMSE macro is to provide a predefined sequence (chain) of PMSE commands with already specified input parameters. Macros are often built using shell scripts, but basically every formalism allowing you to chain shell commands will do. Macros are handy to build larger building blocks of functionality from lower level commands.

MAK_1s1l

Consider the application of sentence segmentation for different languages. We describe in the subsection 8.17.3, how to write a sentence segmentator. In the core of this functionality, P_rer is used with language-specific regular expressions. In such case it would be necessary to have different bulk files for files in different languages.

The example below shows a simple macro called *MAK_1s1l* which is basically a simple if-elseif branch:

```

#!/bin/sh
#
# PMSE MAKRO: execute sentence segmentation for a file with
# language specific segmentator. First, the script creates
# a copy of the input file with 'ss-' prefix.
# The copy is segmented in the next step.
#

FILE=$1 # input 1
LANGUAGE=$2 # input 2

cp $FILE "ss-$FILE"

# lang CES

if [ "$LANGUAGE" = "ces" ] ; then

P_rer 's{(?<\sbiol)(?<\sčl)(?<\sdán)(?<\sfyz)
(?<\sgenmjr)(?<\sgenpor)(?<\sIng)(?<\sJUDr)(?<\smat)
(?<\sMgA)(?<\sMgr)(?<\smjr)(?<\sml)(?<\sMUDr)
(?<\sMVDr)(?<\snapř)(?<\snapor)(?<\snaprap)
(?<\sodst)(?<\spism)(?<\ss)(?<\sr)(?<\sspol)
(?<\sPharmDr)(?<\sPhDr)(?<\splk)(?<\spopř)
(?<\spplk)(?<\sppor)(?<\spprap)(?<\sprap)
(?<\sRNDr)(?<\sRSDr)(?<\sThDr)(?<\stzv)(?<\sjm)
(?<\svl\ .jm)(?<\szkr)(?<\szn)(?<\szvl)
(?<STERM>\p{STerm}\p{QMark}?)\s+(?=\p{P}?\s*(\p{Upper}|\d))}
{${+{STERM}\n}xmsg' "ss-$FILE"

# lang ENG

elif [ "$LANGUAGE" = "deu" ] ; then

P_rer 's{(?<\sAttn)(?<\sc\.c)(?<\scf)(?<\se\.g)(?<\senc)
(?<\sencl)(?<\sref)(?<\sf\.o\.a)(?<\sw)(?<\si\.e)
(?<\sinc)(?<\sYrs)(?<\sadmin)(?<\sh\.q)
(?<\sMan\.Dir)(?<\sa\.o\.b)(?<\sassoc)(?<\sdept)
(?<\srep)(?<\srest)(?<\sexec)(?<\sxt)(?<\sXer)
(?<STERM>\p{STerm}\p{QMark}?)\s+(?=\p{P}?\s*(\p{Upper}|\d))}
{${+{STERM}\n}xmsg' "ss-$FILE"

# lang DEU

elif [ "$LANGUAGE" = "eng" ] ; then

P_rer 's{(?<\sapl)(?<\sd)(?<\sDipl)(?<\sDr)(?<\sG)
(?<\sd\.h)(?<\sevtl)(?<\sgepfl)(?<\si\.H\.v)

```

```
(?<!\ssw)(?<!\su\.v\.a)(?<!\sv\.i\.R)(?<!\sz\.B)
(?<!\sMon)(?<!\stgl)(?<!\sProf)(?<!\su\.A\.w\.g)
(?<!\su\.U)(?<!\sv\.a)(?<!\sz\.d\.I)(?<!\sz\.I)
(?<!\sz\.I\.I)
(?<STERM>\p{STerm}\p{QMark}?)\s+?(?=\p{P}?\s*(\p{Upper}|\d))}
{${+{STERM}\n}xmsg' "ss-$FILE"
```

```
else
```

```
echo "Language $LANGUAGE is not known!"
```

```
fi
```

\$1 and \$2 are the input variables - they contain a value specified as an input argument on CLI. The first value is a filename and the second - an iso-639-3 code - the language specification. The second variable is matched with predefined strings (also iso codes).

The script will make a copy of the input file (new file with ss- prefix will be created). Which will be modified afterwards.

If the specific code matches, the appropriate segmentator is used.⁴⁰

If the second input parameter doesn't match any predefined code, the script will print a warning.

Further extensions

It may be useful to integrate some text processing functions in the macro. The name of the macro is *MAK_1s1l* - one sentence per one line. However, the input text could be wrong formatted, it could contain e.g. multiple empty lines or headlines; the resulting file could be affected as well.

P_trt can be used to remove headlines (titles) from the text and to remove all formatting characters (mainly line breaks). Following command should replace the copy `cp $FILE "ss-$FILE"` command:

```
P_trt --action remove_headline --in $FILE --out STDOUT | \
P_trt --in STDIN --action deformat --out STDOUT > "ss-$FILE"
```

Here is a little example⁴¹ of what will happen with the input text. Let the original text be⁴²:

```
Language <br>
<br>
Language is the human capacity for acquiring and using complex
systems of communication, and a language is any specific example
of such a system. The scientific study of <br1>
language          is          called linguistics.<br>
```

⁴⁰The segmentators above are only examples - they contain abbreviations which may cause a problem in the segmentation. The enumeration of the abbreviations is definitely not final.

⁴¹From <http://en.wikipedia.org/wiki/Language>

⁴²The *
* signs denotes line breaks. They are not a part of the original text - they are used to visualize the problem. We want to remove breaks 1, 2 and 3, because they corrupt the sentences

```
<br>
Estimates of the number of languages <br2>
<br3>
in the world vary between 6,000 and 7,000.<br>
```

After the first P_trt command the text should look like:

```
Language is the human capacity for acquiring and using complex
systems of communication, and a language is any specific example
of such a system. The scientific study of
language          is          called linguistics.
```

```
Estimates of the number of languages in the world vary between
6,000 and 7,000.
```

The second P_trt command will remove all other formatting, thus the text will have a form of:

```
Language is the human capacity for acquiring and using complex
systems of communication, and a language is any specific example
of such a system. The scientific study of language is called
linguistics. Estimates of the number of languages in the world
vary between 6,000 and 7,000.
```

The final (segmented) text will look like:

```
Language is the human capacity for acquiring and using complex
systems of communication, and a language is any specific example
of such a system.<br>
The scientific study of language is called
linguistics.<br>
Estimates of the number of languages in the world
vary between 6,000 and 7,000.<br>
```


Appendix (User Manual) C

Command Reference

*Words, words, words! They shut one off from the universe.
Three quarters of the time one's never in contact with things,
only with the beastly words that stand for them.*
Aldous Huxley

Command set help menu. Usage: <command>[; command]*:

The ';' semicolon is used as delimiter between commands. If you want to execute more than one comand in a command line separate them with ';'. Also, you can cluster words to phrases with double quotes: "<phrase1> <phrase2>" or with underscore <phrase1>_<phrase2>.

C.1 ?

C.2 bot

Help menu for the command 'bot'

SYNOPSIS

bot [<name>=<name>] [cmd=<subcommand>]

DESCRIPTION

Performs a subcommand for a given bot. If no bot name is given, 'Elric' is taken as default. If no subcommand is given, the ontology for the particular bot is reloaded. Valid subcommands are:

reload reload ontology for bot <name>

```
load      load a saved bot-state in bot <name>
save      save state of bot <name>
```

C.3 bot_add

Help menu for the command 'bot_add'

SYNOPSIS

```
bot_add [name=<name>] [onto=<onto[,onto]*>] lang=<lang[,lang]*>
```

DESCRIPTION

Add a chatbot with to the bot pool. The following arguments are possible:

name Name of the bot. If <name> is not given, the default is 'Elric'. If a bot with the given name already exists, a warning is issued and the bot is not created. If the special value '_auto_' is given, the system determines the name of a bot automatically, based on parameters in the global configuration file.

onto Ontologies that shall form the bots knowledge base may be given as a comma-separated list. If no ontology is given, the default is defined in the global configuration file.

lang Languages the bot shall be aware of may be given as a comma separated list (of ISO639-3 codes). If no language is given, the default is 'eng'.

The system tries to load for all given languages all given ontologies and skips nonexistant combinations.

EXAMPLES

```
> bot_add name=Alfons
```

Will add a chatbot of the name 'Alfons' (if not already exists) to the bot pool. The bot will comprehend english and use the default ontologies for english.

```
> bot_add onto=petamem,<specific> lang=deu,eng,nld
```

Will add a chatbot of the name 'Elric' (if not already exists) to the bot pool. The bot will comprehend german, english and dutch and use the ontologies 'petamem' and one named '<specific>' for all these languages (if present).

C.4 bot_del

Help menu for the command 'bot_del'

SYNOPSIS

```
bot_del [name=<name>]
```

DESCRIPTION

Remove a chatbot with name <name> from the bot pool. If <name> is not given, the default bot 'Elric' is removed. If no bot with the given name exists in pool, the user is informed with a message.

C.5 bots_list

Help menu for the command 'bots_list'

SYNOPSIS

```
bots_list [name=<regex>]
```

DESCRIPTION

If name is given (regular expression), a list of all bots whose names match this regex is given. If no name is given, a list of all bots is given.

The format of this list is

- 1) Botname
- 2) list of languages bot knows and the number of rules for each language
- 3) number of answers/number of questions asked
- 4) timestamp when bot was born/created
- 5) time in seconds how long bot is idle (since last request)

C.6 cat

Help menu for the command 'cat'

SYNOPSIS

```
cat <glob(s)>
```

DESCRIPTION

This command shows the contents of the files matched by the given name/glob(s).

C.7 cfg-client

Help menu for the command 'cfg-client'

SYNOPSIS

```
cfg-client [in=<input_format>] [out=<output_format>]
```

DESCRIPTION

You can change the local configuration variable for input and output format settings of PMLS at runtime with this command.

Setting input and output format:

```
> cfg-set in=<input_format> out=<output_format>
```

Currently PMLS server supports following <in>/<out> values:

input formats: ? std yaml xml json serverdefault auto plaincmd

output formats: ? std yaml xml json serverdefault same

By specifying value '?' you can get list of possible values (except for '?' itself).

'yaml', 'xml' and 'json' are well known serialization formats.

std is perl data structure format and is valid perl code.

'plaincmd' specify basic, intuitive, human-readable input. For example all commands in the documentation and help are of the 'plaincmd' format.

There is also special value 'serverdefault' to use default server value.

Another special value 'auto' recognizes input format by itself with the help of a very simple heuristics.

If you specify 'out=same', server will always answer in same format as input was. This option makes sense if 'in=auto'. We should also mention that you will get an error when you have 'in=auto out=same' and you send the command in format not being in output formats list (e. g. 'plaincmd').

C.8 cfg-get

Help menu for the command 'cfg-get'

SYNOPSIS

```
cfg-get [<section> [<key>]]
```

DESCRIPTION

This command allows you to inspect the PMLS configuration settings. If no parameters are given, a list of all available sections is printed. If only a section is given as parameter, all keys and their values in this section are printed. If both section and key are given, just the value for this key is printed. For configuration settings see the 'cfg-set' command.

C.9 cfg-reload

Help menu for the command 'cfg-reload'

SYNOPSIS

```
cfg-reload [<path>]
```

DESCRIPTION

This command takes one optional parameter <path>. It will reload the global configuration from the the default file (if no path given) or from the specified <path>. This is useful if you have edited several values in that file and would like to affect PMLS behaviour at runtime, or if the configuration got broken by issuing wrong 'cfg-set' commands and you would like to reset it to the defaults.

C.10 cfg-set

Help menu for the command 'cfg-set'

SYNOPSIS

```
cfg-set <section> [<key> [<value>]]
```

DESCRIPTION

You can change or delete(!) the global configuration variable settings of PMLS at runtime with this command. Wrong values may crash PMLS, so be careful when changing or deleting values. You can inspect the current state of the configuration with the command 'cfg-get' or reload configuration from disk with the 'cfg-reload' command.

Setting values:

```
> cfg-set <section> <key> <value>
```

Will set the value <value> for key <key> in section <section>

Deleting values:

```
> cfg-set <section>
```

Will delete the entire section <section>.

```
> cfg-set <section> <key>
```

Will delete key <key> from section <section>.

C.11 compat

Help menu for the command 'compat'

SYNOPSIS

```
compat <string>
```

DESCRIPTION

You may test a given string for its 'compatibility' in various language encodings. A list of all compatible languages is returned.

C.12 convert

Help menu for the command 'convert'

SYNOPSIS

```
convert <src><dst><dt>
```

DESCRIPTION

This command is a frontend to the PMLIB Data Mining Framework. It allows for the conversion between arbitrary file formats, provided there are the converters installed on the system.

C.13 cp

Help menu for the command 'cp'

SYNOPSIS

```
cp <src> <dst>
```

DESCRIPTION

This command copies a file with the name <src> to a file with the name <dst>. Unlike with the 'mv' command, the <src> file still exists after a successful operation. The <dst> file must not exist prior to command execution.

C.14 dict

Help menu for the command 'dict'

SYNOPSIS

```
dict id=<phrase> dst=<dst-iso> [src=<source-iso>] [infer=(0|1*)]
```

DESCRIPTION

This command provides dictionary functionality and takes these named arguments:

'id' the phrase to translate
'src' the iso639-3 language as source (all if not given)
'dst' the iso639-3 language as destination
'infer' whether to infer translation if direct lookup fails

EXAMPLES

```
> dict id=house dst=deu
```

Will lookup the phrase 'house' in all lexica (because no source has been specified) currently present in the repository. If found it shall be translated to 'deu' (german). Assuming there might be both czech and english dictionaries loaded, this will result in the translations Haus (from english) and 'Gänschen' (from czech).

C.15 doc

Help menu for the command 'doc'

SYNOPSIS

```
doc lang=<iso>
```

DESCRIPTION

Internal command for documentation purposes.

C.16 e-cpy

Help menu for the command 'e-cpy'

SYNOPSIS

```
e-cpy iso=<iso639-3> lex=<string> key=<string> [dst_iso=<iso639-3>] [dst_lex=<string>] [dst_k
```

DESCRIPTION

Will copy an Entry within a Lexicon or in between Lexica. The destination arguments may be omitted if at least one of them differs from the corresponding source arguments, then the default value is identical to that of the source argument. See examples. Already existing entries at the destination position will not be overwritten. Use e-del first to delete them.

EXAMPLE

```
> e-cpy iso=eng lex=base key=house dst_key=hut
```

Will copy the Entry "house" to a new Entry "hut" within Lexicon "eng-base".

```
> e-cpy iso=eng lex=base key=house dst_iso=ces dst_lex=biology
```

Will copy the Entry "house" to a new Entry of the same identifier (key) from Lexicon "mul-base" to the Lexicon "ces-biology".

C.17 e-del

Help menu for the command 'e-del'

SYNOPSIS

```
e-del iso=<iso639-3> lex=<string> key=<string>
```

DESCRIPTION

Will delete the specified Entry from Vault.

C.18 e-describe

Help menu for the command 'e-describe'

SYNOPSIS

```
e-describe iso=<iso639-3> lex=<string> key=<string> loc=<iso639-3>
```

DESCRIPTION

Will describe the specified Entry. The description will happen in language <loc> - if possible. Default for <loc> is 'eng'.

C.19 e-get_arg

Help menu for the command 'e-get_arg'

SYNOPSIS

```
e-get_arg iso=<iso639-3> lex=<str> key=<str> [rel=<str>]
```

DESCRIPTION

Will output the Argument to <rel> in specified Entry. <rel> is optional and defaults to '.'.

C.20 e-get_common

Help menu for the command 'e-get_common'

SYNOPSIS

```
e-get_common iso=<iso639-3> lex=<str> key1=<str> key2=<str>
```

DESCRIPTION

Will output the common concepts for two specified Entries. Both Entries are to be within the same Lexicon.

C.21 e-get_thull

Help menu for the command 'e-get_thull'

SYNOPSIS

```
e-get_thull iso=<iso639-3> lex=<str> key=<str> rel=<str>
```

DESCRIPTION

Will output the transitive hull of the specified Entry and Relation <rel>. If <rel> is not given, it defaults to '☒' (the hyponym/hypernym relation).

C.22 e-mov

Help menu for the command 'e-mov'

SYNOPSIS

```
e-mov iso=<iso639-3> lex=<string> key=<string> [dst_iso=<iso639-3>] [dst_lex=<string>] [dst_k
```

DESCRIPTION

Will move an Entry 'key' from LexCluster 'iso', Lexicon 'lex' to another location in LexCluster 'dst_iso', Lexicon 'dst_lex' and 'dest_key'. After the operation the original source Entry is not existant anymore - only the destination Entry. The command will refuse to move an Entry to itself.

C.23 e-new

Help menu for the command 'e-new'

SYNOPSIS

```
e-new iso=<iso639-3> lex=<string> key=<string> [data=<struct>]
```

DESCRIPTION

Will create a new Entry anchored under key <key> in LexCluster <iso> in the Lexicon 'lex' and with content defined in <data>. If no <data> is given, it defaults to the empty hash.

C.24 e-show

Help menu for the command 'e-show'

SYNOPSIS

```
e-show iso=<iso639-3> lex=<string> key=<string>
```

DESCRIPTION

Will output the structure of Entry with Key <key> that is located in the Lexicon 'lex' (being in LexCluster <iso>).

C.25 e-translate

Help menu for the command 'e-translate'

SYNOPSIS

```
e-translate iso=<iso639-3> lex=<str> key=<str> [loc=<iso639-3>] [infer=<int>]
```

DESCRIPTION

Will perform a translation of the specified Entry to language <loc>. If <loc> is not given, it defaults to 'eng'. The <infer> parameter defaults to '1' and defines if the engine shall attempt a translation inference (i.e. using other languages translation information) if direct translation information is not available.

C.26 encode

Help menu for the command 'encode'

SYNOPSIS

```
encode <src><dst>[<dt><st>]
```

DESCRIPTION

This command is a frontend to the PMLIB::Encoding 'convert_encoding' function. It allows for the conversion between arbitrary file encodings, provided there are the converters installed on the system.

C.27 h

Command set help menu. Usage: <command>[; command]*:

The ';' semicolon is used as delimiter between commands. If you want to execute more than one comand in a command line separate them with ';'.
Also, you can cluster words to phrases with double quotes:

"<phrase1> <phrase2>" or with underscore <phrase1>_<phrase2>.

C.28 help

Command set help menu. Usage: <command>[; command]*:

The ';' semicolon is used as delimiter between commands. If you want to execute more than one comand in a command line separate them with ';'. Also, you can cluster words to phrases with double quotes: "<phrase1> <phrase2>" or with underscore <phrase1>_<phrase2>.

C.29 help-syntax

Help menu for the command 'help-syntax'

SYNOPSIS

help-syntax

DESCRIPTION

You can enter commands to the PMLS in the following form:

<command>[; command]*

The ';' semicolon is used as delimiter between commands. If you want to execute more than one command in a command line, separate them with ';'. <command> is

<cmd_key> [<parameters>]

i.e. a command key "keyword" and potentially a list of parameters. But there are also commands that take no parameters. You find the list of command keywords your user is allowed to issue via the "perms_list" command.

The <parameters> is a space separated list of parameters. These may be either positional or in a key=value format.

As whitespace is used as a delimiter, you can cluster space-separated words to phrases with double quotes: "<phrase1> <phrase2>". Please be aware, that in order to engulf a space-separated phrase as a single parameter, you hve to put quotes around the whole key value construct.

e.g.

> say to=Botname "this=Hello, my name is Michael"

C.30 l-cpy

Help menu for the command 'l-cpy'

SYNOPSIS

```
l-cpy iso=<iso639-3> lex=<string> [dst_iso=<iso639-3>] dst_lex=<string> [force=(0*|1)]
```

DESCRIPTION

Will copy content of Lexicon 'lex' in LexCluster <iso> to a new position in LexCluster <dst_iso> under the name 'dst_lex'. <dst_iso> is optional and defaults to <iso>. This operation will not overwrite an existing target Lexicon, except if the <force> parameter is set to true.

EXAMPLE

```
> l-cpy iso=eng lex=arithmetics dst_iso=mul dst_lex=math
```

Will copy the Lexicon 'eng-arithmetics' to 'mul-math'. So after the operation, there will be both 'eng-arithmetics' and 'mul-math' lexica with exactly the same content.

C.31 l-del

Help menu for the command 'l-del'

SYNOPSIS

```
l-del iso=<iso639-3> lex=<string> [force=(0*|1)]
```

DESCRIPTION

Will delete Lexicon with the name 'lex' in the appropriate LexCluster <iso>. Operation is blocked if the Lexicon has been changed since its start, except if the force parameter is set to true.

C.32 l-info

Help menu for the command 'l-info'

SYNOPSIS

```
l-info iso=<iso639-3> lex=<string>
```

DESCRIPTION

Print information about the Lexicon in LexCluster <iso> of the name 'lex'.

C.33 l-list

Help menu for the command 'l-list'

SYNOPSIS

```
l-list iso=<regex> lex=<regex>
```

DESCRIPTION

Will list all those available Lexica on disk, that match the constraints given by the <iso> and the 'lex' regexes.

C.34 l-load

Help menu for the command 'l-load'

SYNOPSIS

```
l-load iso=<iso639-3> lex=<string> [path=<path>] [force=(0*|1)] [verbose=(0*|1)]
```

DESCRIPTION

Will load the Lexicon of language <iso> and name 'lex' from disk, as well as all Lexica that are required for the operation of this particular Lexicon (prerequisites). This Lexicon will be stored in the <iso> LexCluster under the name 'lex'. If any Lexicon (this or prerequisites) of the same name already exists, it will not be loaded thus not overwrite the existing Lexicon, except if the <force> parameter is set to true. If the <verbose> parameter is set, the loading process will emit some status messages.

The <path> parameter may be given optionally, in which case the Lexicon is loaded from that given path directly instead of constructing it from <iso> and 'lex'.

C.35 l-mov

Help menu for the command 'l-mov'

SYNOPSIS

```
l-mov iso=<iso639-3> lex=<string> [dst_iso=<iso639-3>] dst_lex=<string> [force=(0*|1)]
```

DESCRIPTION

Will move (same as rename) Lexicon 'lex' in LexCluster <iso> to its new position in LexCluster <dst_iso> under the name 'dst_lex'. Except for the name change, the Lexicon is not changed. <dst_iso> is optional and defaults to <iso>. This operation will not overwrite an existing target Lexicon, except if the <force> parameter is set to true.

EXAMPLE

```
> l-mov iso=eng lex=arithmetics dst_iso=mul dst_lex=math
```

Will move the Lexicon 'eng-arithmetics' to 'mul-math'. So after the operation, there will be no 'eng-arithmetics' lexicon anymore, but a 'mul-math' instead with exactly the same content.

C.36 l-new

Help menu for the command 'l-new'

SYNOPSIS

```
l-new iso=<iso639-3> lex=<string> [force=(0*|1)]
```

DESCRIPTION

Creates a new Lexicon with the name 'lex' in the appropriate LexCluster <iso>. Operation is blocked if a Lexicon of the same name exists already, except if the force parameter is set to true. In that case, the already existing lexicon gets overwritten with an empty lexicon - thus effectively cleared.

EXAMPLE

```
> l-new iso=eng lex=my-ontology force=1
```

Creates a Lexicon with the name 'my-ontology' in the LexCluster 'eng' (English). If there was already such a Lexicon, it gets cleared.

C.37 l-reversify

C.38 l-save

Help menu for the command 'l-save'

SYNOPSIS

```
l-save iso=<iso639-3> lex=<string> [path=<path>] [verbose=(0*|1)]
```

DESCRIPTION

Will save the Lexicon of language <iso> and name 'lex' to disk. If the <verbose> parameter is set, the saving process will emit some status messages.

The <path> parameter may be given optionally, in which case the Lexicon is saved to that given path directly instead of constructing it from <iso> and 'lex'.

C.39 lc-chart

C.40 lc-cpy

Help menu for the command 'lc-cpy'

SYNOPSIS

```
lc-cpy iso=<iso639-3> dst_iso=<iso639-3>
```

DESCRIPTION

Will copy LexCluster <iso> - including all contents - to a new LexCluster <dst_iso> in Vault. Both arguments are mandatory and for the operation to succeed, <iso> must exist, while <dst_iso> must not exist in Vault. After the operation, two LexClusters <iso> and <dst_iso> exist in Vault.

C.41 lc-del

Help menu for the command 'lc-del'

SYNOPSIS

```
lc-del iso=<iso639-3>
```

DESCRIPTION

Will delete LexCluster <iso> from Vault. For the operation to succeed, <iso> must exist in Vault.

C.42 lc-info

Help menu for the command 'lc-info'

SYNOPSIS

```
lc-info iso=<iso639-3> [loc=<iso639-3>]
```

DESCRIPTION

Print information about LexCluster <iso> stored in Vault. Optionally, the information can be localized to <loc>. Default for <loc> are the system settings.

C.43 lc-list

Help menu for the command 'lc-list'

SYNOPSIS

```
lc-list iso=<regex>
```

DESCRIPTION

Will list all those available LexClusters on disk, that match the constraints given by the <iso> regex.

C.44 lc-mov

Help menu for the command 'lc-mov'

SYNOPSIS

```
lc-mov iso=<iso639-3> dst_iso=<iso639-3>
```

DESCRIPTION

Will move LexCluster <iso> - including all contents - to a new LexCluster <dst_iso> in Vault. Both arguments are mandatory and for the operation to succeed, <iso> must exist, while <dst_iso> must not exist in Vault. After the operation, only LexCluster <dst_iso> exists in Vault.

C.45 lc-new

Help menu for the command 'lc-new'

SYNOPSIS

```
lc-new iso=<iso>
```

DESCRIPTION

Will create a new LexCluster <iso> in Vault. For the operation to succeed, <iso> must not exist in Vault.

C.46 lcmp

C.47 lcover

C.48 li

Help menu for the command 'li'

SYNOPSIS

```
li type=<type> text=<text>|file=<file>
```

DESCRIPTION

This command can identify the language of a given text that is either given directly on command line as named argument 'text' or that resides in file <file> in the users home directory (where it might have been transferred with the 'put' command.

If both 'text' and 'file' are given, 'text' has priority.

The parameter <type> determines the method the language is identified with and can have these values:

- 'ngram' a simple but effective NGram-based text identification (default)
- 'nvect' the PetaMem proprietary n-dimensional vector distance, (should not be used on long texts)
- 'dict' a dictionary based identification
- 'tfreq' top-frequency based identification - this is a hybrid between statistical and dictionary based identification
- 'smart' an intelligent combination of various li methods ensuring the most accurate result possible.

If you give '?' as the type of language identification, it will return a list of valid identification methods.

If you give '?' as the text argument, it will return the supported languages for the identification method given.

EXAMPLES

```
> li type=?
```

Will return a list of supported language identification methods.

```
> li text=?
```

Will return the list of supported languages for 'ngram' identification method. (Because it is default)

```
> li type=tfreq "text=This is my house."
```

Will identify the text given as being english.

C.49 li-llm

Help menu for the command 'li-llm'

SYNOPSIS

```
li-llm type=<type> name=<name> files=<files>
```

DESCRIPTION

This command takes a LI method name <type> and comma-separated list of files being in home directory. It creates language model for language identification for given method.

Resulting file name is <name> dot default extension. It will be stored in default model directory.

You may also use glob-wildcards in the <sfile> specification.

C.50 ls

Help menu for the command 'ls'

SYNOPSIS

```
ls [<glob(s)>]
```

DESCRIPTION

This command without parameters lists all the contents of the user's PMLS home directory. Optionally a list of globs can be given as parameters. These globs are then expanded and only the matching files are displayed.

C.51 mt

Help menu for the command 'mt'

SYNOPSIS

```
mt [<file> <tl[:sl]> [<qual>]]
```

DESCRIPTION

This command is the basic interface to the PMLS machine translation engine. It translates a given textfile (must be UTF-8 encoded) with source language <sl> (optional - if not given it is autodetected) to a target language <tl>. The behaviour of this function is influenced by the <qual> parameter which itself can be set via \$e1 mtconfig\$e0. <qual> is optional and if not given, the lowest quality level available will be assumed. If you just issue the mt command with no further parameters, you get a matrix of supported language-pairs and

the maximum translation quality supported for them.

C.52 mv

Help menu for the command 'mv'

SYNOPSIS

```
mv <from> <to>
```

DESCRIPTION

This command moves (or renames - which actually is the same semantics) a file with the name <from> to a file with the name <to>. Unlike with the 'cp' command, the <from> file doesn't exist after a successful operation. The <to> file must not exist prior to command execution.

C.53 nla-list

Help menu for the command 'nla-list'

SYNOPSIS

```
nla-list str=<str> [iso=<iso639>] [type=<log>]
```

DESCRIPTION

This command will analyze and convert a natural language list (or enumeration) and return its elements as a formal list. 'iso' (default: eng) and 'type' (default: - which means AND) are optional parameters.

If you enter '?' for argument 'iso' a list of supported languages is returned;

EXAMPLES

```
> nla-list "str=alpha, beta, delta and gamma"
```

Will return a list with the elements [alpha beta delta gamma].

```
> nla-list "str=gestern, heute oder morgen" iso=deu type=
```

Will return a list with the elements [gestern heute morgen].

C.54 nla-morph

C.55 nla-number

Help menu for the command 'nla-number'

SYNOPSIS

```
nla-number str=<numeral> [iso=<iso639>]
```

DESCRIPTION

This command will compute the decimal number corresponding to the textual (cardinal) representation of the numeral. It is the inverse operation of the 'nlg-number' command.

PARAMETERS

str=<numeral>

The parameter 'str' is the string to analyze and it is usually mandatory. As it may be a space-separated string do not forget to quote it in that case.

iso=<iso639>

An optional parameter is 'iso' which is the language hint for the analysis engine. If not given, it defaults to '*', which means "autodetect".

If '?' is given instead of the iso code, the list of supported languages is returned. In that case, 'str' is optional.

C.56 nla-tag

Help menu for the command 'nla-tag'

SYNOPSIS

```
nla-tag [iso=<iso639>] [lex=<str>[,<str>]*] text=<text>|file=<filename>
```

DESCRIPTION

This command implements various taggers for various languages.

PARAMETERS

iso=<iso639>

Specify the language (in ISO639-3 encoding) of the text to be tagged. If not specified, 'eng' - english - is taken. Because of this default, this parameter is optional.

lex=<str>[,<str>]*

Give the lexicon/lexica that are consulted for morphosyntactic information tagging. If no lexica are given, all lexica currently present in repository are assumed. Therefore this argument is optional.

text=<text>

Enter the text to get information about on the command line resp. in your client application.

file=<filename>

Enter the filename to read text content from. The 'text' and file' parameters are exclusive - only one is needed. If both are given, the text from the commandline is taken.

C.57 nlg-list

Help menu for the command 'nlg-list'

SYNOPSIS

```
nlg-list list=<comma-separated string> iso=<iso639> type=<type>
```

DESCRIPTION

This command will generate a natural language list representation of a given list of items.

EXAMPLES

```
> nlg-list "list=a,b,c,d"
```

Will return a string "a, b, c and d", as "eng" is the default value for 'iso' and "∩" (logical AND) is the default value for 'type'.

```
> nlg-list "list=a,b,c,d" iso=ces type=∩
```

Will return the string "a, b, c nebo d".

C.58 nlg-morph

C.59 nlg-number

Help menu for the command 'nlg-number'

SYNOPSIS

```
nlg-number [num=<number>] [iso=<iso639>]
```

DESCRIPTION

This command handles the conversion of numbers to a textual representation in a given (and supported) language.

PARAMETERS

```
num=<number>
```

The parameter 'num' is the integer that is to be converted into a textual representation (cardinal numeral). If not given, and 'iso' contains a valid and supported iso639-3 code, the supported conversion interval for this language is given.

iso=<iso639>

The parameter 'iso' may be a valid language iso code, a '?' or undefined (not given). If it is a valid and supported iso code and 'num' has been given, the number is converted to a textual representation. If num has not been given, returns an interval for the given language where conversion is supported. If a '?' has been given, the list of supported languages is returned.

If neither 'num' nor 'iso' has been given, returns a bulk data structure containing the supported iso codes as keys and the valid intervals for each key as the corresponding value.

C.60 nlp-m_test

C.61 perm_add

Help menu for the command 'perm_add'

SYNOPSIS

```
perm_add [<user>] <permlist>
```

DESCRIPTION

You can set permissions specified in (space separated) <permlist> for the user <user>. Of course, you must have permission for the command itself to issue it. The current username (as obtained by whoami command) is used if <user> omitted. An user being able to issue this command is superuser/administrator by definition.

Permissions can be removed using the 'perm_del' command.

C.62 perm_del

Help menu for the command 'perm_del'

SYNOPSIS

```
perm_del [<user>] <permlist>
```

DESCRIPTION

You can unset permissions specified in <permlist> for the user <user>. Of course, you must have permission for the command itself to issue it. The current username (as obtained by whoami command) is used if <user> omitted. Be careful, as you can disable any command with this (even 'perm_add', 'shutdown' and other potentially vital commands).

Permissions can be again set using the 'perm_add' command.

C.63 perms_list

Help menu for the command 'perms_list'

SYNOPSIS

```
perms_list [<user>]
```

DESCRIPTION

This command will show the current permissions of an user that is specified by the parameter <user>. The current username is used if <user> omitted.

A list of all users known to the system is obtained with the command users_list.

C.64 purge

Help menu for the command 'purge'

SYNOPSIS

```
purge [<refid>]+
```

DESCRIPTION

Immediately close client connection(s) specified by <refid> - reference connection number(s) which can be determined from 'who' command output. It is possible to specify more than one connection to be closed. No action is taken for invalid connection reference number. Also, it is not possible to purge the "root-connection" with id 0.

C.65 riter

Help menu for the command 'riter'

SYNOPSIS

```
riter <ldef> <cmd>
```

DESCRIPTION

this command is the global iterator. It takes <ldef> - a description of lexica to iterate and iterates these issuing an arbitrary <cmd> command for each of these. <ldef> can be of the following format:

```
'*'           wildcard. Simply takes all lexica present in repository
'l1:l2:..'    a colon-delimited list of lexica names
'xxx'        everything else is considered a regular expression
              describing the required lexicon name(s)
```

<cmd> can be an arbitrary command the user has permissions to execute and may contain a '%' as parameter. This is a placeholder that will be replaced by the lexicon name of the current iteration

EXAMPLES

```
> riter g$ normalize %
  performs a normalization for every lexicon whose name is ending on 'g'

> riter sv:de:xx ladd %
  same as 'ladd sv de xx'
```

C.66 rm

Help menu for the command 'rm'

SYNOPSIS

```
rm <glob(s)>
```

DESCRIPTION

This command removes the given contents from the user's PMLS home directory. This command expects at least one parameter. A list of globs can be given as parameters. These globs are then expanded and only the matching files are removed.

C.67 say

Help menu for the command 'say'

SYNOPSIS

```
say [to=<name>] "this=<content>"
```

DESCRIPTION

Say a sentence, specified in <content> to chatbot with name <name>. If <name> is not given, the default bot, Elric, is considered to be the target. If a bot with the specified name doesn't exist, a message is

generated. Please be aware, that if your text contains spaces (which it almost always will, you have to put the whole "this" parameter into quotes e.g. "this=Wie geht's Dir?"

C.68 shutdown

Help menu for the command 'shutdown'

SYNOPSIS

shutdown

DESCRIPTION

This command will immediately terminate server execution. It takes no parameters.

C.69 sys-info

Help menu for the command 'sys-info'

SYNOPSIS

sys-info

DESCRIPTION

This command gives you detailed information about the internals of the PMLS system. It provides following values:

Vsize
debug
history
input_format
input_formats
log_level
log_verbose
msg_nosend
msg_verbose
output_format
output_formats
revision
revision_date
revision_time
start
startcmd
time
Modules/Versions

C.70 txt-info

C.71 user_add

Help menu for the command 'user_add'

SYNOPSIS

```
user_add <user> <pass>
```

DESCRIPTION

This command is to be used by the administrator to create an account with default permissions for a new user.

C.72 user_del

Help menu for the command 'user_del'

SYNOPSIS

```
user_del <user>
```

DESCRIPTION

With this command, the system administrator can remove obsolete users. Only the user account (with all permission information) is removed, but not the users home directory.

C.73 users_list

Help menu for the command 'users_list'

SYNOPSIS

```
users_list
```

DESCRIPTION

This command will show a comma-separated list of all usernames known to the system. It takes no parameters.

C.74 v-cchk

Help menu for the command 'v-cchk'

SYNOPSIS

```
v-cchk [lc=<rx>] [l=<rx>] [e=<rx>] [rs=<rx>]
```

DESCRIPTION

Performs a consistency check of the Vault. The optional parameters given for 'lc' (LexCluster), 'l' (Lexicon), 'e' (Entry) and 'rs' (RelSet) will constrain the searchspace and are interpreted as regular expressions. Every parameter not given, defaults to the whole search space.

EXAMPLES

```
> v-cchk
```

Will perform a consistency check on the whole Vault. Thus every LexCluster in Vault, every Lexicon in every LexCluster and every Entry in every Lexicon and every Relation Set in every Entry are checked. Depending on the number of loaded Lexica and their size this operation can take quite a long time.

```
> v-cchk lc=eng
```

Will perform a consistency check on everything within the 'eng' LexCluster, but constrain the search space to only this LexCluster even if others are present in the Vault also.

C.75 v-get_clique

Help menu for the command 'v-get_clique'

SYNOPSIS

```
v-get_clique iso=<iso> lex=<name> key=<key> [rel_rx=<regex>]
```

DESCRIPTION

For an Entry - defined by <iso>, <lex> and <key> - finds the clique of entries that are connected with this Entry and among each other by Relation(s), that match <regex>. If the 'rel_rx' parameter is not given, it defaults to '☒' (which effectively returns the reflexive-transitive hull of hyponyms/hypernyms).

C.76 v-info

Help menu for the command 'v-info'

SYNOPSIS

```
v-info <no arguments>
```

DESCRIPTION

Print information about the Vault.

C.77 v-query

Help menu for the command 'v-query'

SYNOPSIS

v-query [lc=<any>] [l=<any>] [e=<any>]

DESCRIPTION

Prints occurrences where key has been found in Vault;

C.78 who

Help menu for the command 'who'

SYNOPSIS

who [<sort_type>]

DESCRIPTION

Use this command to determine who is currently logged in. Each row describe one user connection and consist of fields

<refid> - reference connection number
<user> - user login name
<IP-from> - IP address user came from
<since> - time user connected since
<idle> - idle time in format hh:mm
<current> - actual connection marked with '*'

Sorting is possible for columns 'refid' (default), 'user', 'since' and 'idle'. These column names can be used as <sort_type> param for this command.

EXAMPLE OUTPUT

232	edeo	67.45.113.202	2004-04-13 15:21	0:12
1111	alis	194.223.145.112	2004-04-13 13:55	0:55
3243	alis	194.223.145.112	2004-04-12 12:01	24:35
65014	admin	127.0.0.1	2004-04-14 15:22	0:01 *

C.79 whoami

Help menu for the command 'whoami'

SYNOPSIS

whoami

DESCRIPTION

This command will show the username you are currently using. It takes no parameters.

C.80 debug

C.81 get

This comand is designed for file retrieval from server. Required param <rfile> is name of the file located @ server in the user's PMLS home directory which should be retrieved. If the optional parameter <dst> is omitted, the same filename as stated in <rfile> is assumed. If such a file already exists at the client-side, its content will be replaced with the new one without asking!

C.82 put

This command will take a local file <lfile> at the clientside, transfer it to the server and store it there in the user's PMLS home directory under the filename <dst>. If the optional parameter <dst> is omitted, the same filename as stated in <lfile> is assumed. If such a file already exists at the server-side, the server stores it under an alternative name and returns it.

C.83 quit

Takes no parameters. Use this command to terminate a client. (Server keeps running, for termination of a server, see command 'shutdown').

ALIASES

q
qu
qui

C.84 user

Use this command for changing user identity (and also privileges) without closing the client application. You will be asked for the password of the target user.

Empty passwords are not supported and interrupt this command if entered. Command execution can be also interrupted by Ctrl+C. Command is also interrupted automatically after user inactivity. Timeout is set to 25 seconds for the next keystroke.

user admin - change user identity to admin (after successful password verification)

Appendix (User Manual) D

KR Formalism

*Data is not information,
information is not knowledge,
knowledge is not understanding,
understanding is not wisdom.*
Clifford Stoll

D.1 Class Logic...

Knowledge representation (KR) is the effort to use a set of symbols to represent a set of facts within a knowledge domain. PMLS deploys a KR formalism implementing the as defined by Oberschelp et al.

This axiomatic class logic has been developed by A. Oberschelp on top of work done by Willard Van Orman Quine. It is a consistent extension of predicate logic and allows for the unconstrained use of class terms. It can also use all those classes as terms that generate antinomies in the naive set theory, which is possible because the class logic has no axioms of existence for classes.

D.1.1 ...And Beyond

Furthermore, we deploy probabilistic values instead of the boolean true/false values implicitly used. This extension is neutral to the original definitions of the formalism, because all operations on junctions are exchanged consistently with their probabilistic counterparts. Every entity in the formalism has a given probability. While in regular predicate or boolean logic this probability is just true (1) or false (0), in probabilistic logic the value can be anywhere in between.

Other logical junctions can be construed and are in fact implemented in `PMLIB::Stochastic::P`.

D.2 Original Nomenclature...

Individual These are the most interesting objects.

- individuals are elements of classes
- there may be relations between individuals
- individuals may be both arguments and values of functions

Atom A special kind of individual which is not a class and thus has no elements itself.

Object The most generic description that includes *individuals* and *classes*. There are **real objects** and **virtual objects**. We consider “real object” being a synonym for “individual”.

Class A collection of individuals.

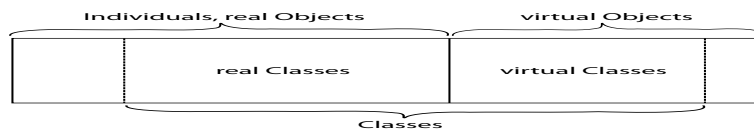
Set A special class, which can also be considered an individual.

Pair A special *individual* consisting of two ordered individuals. So from a pair both individuals as well as their order may be reconstrued.

To put objects and classes into a conceptual frame, we can say that:

- The world (everything, omega) consists of objects.
- It is split in two categories: real objects (aka individuals) and virtual objects.
- The set of real objects has a subset of real classes
- The set of virtual objects has a subset of virtual classes
- Therefore, classes are a subset of the world.
- Therefore, there do exist real objects that are not classes.
- Therefore, there do exist virtual objects that are not classes.

This picture should visualize that conceptual frame:



D.2.1 Terms and Formulae

Class logic defines two types of information-carrying entities. **Terms** (we often use X, Y, Z, \dots for these) and **Formulae** (we use φ, ψ, \dots for these), where Terms are Objects and Formulae are statements. Formulae are relational triples of the form

$\langle \text{L-Term} \rangle \langle \text{Relation} \rangle \langle \text{R-Term} \rangle$

and can be connected to more complex formulae using junctions like

$\neg \varphi$	not φ
$\varphi \wedge \psi$	φ and ψ
$\varphi \vee \psi$	φ or ψ
$\varphi \Rightarrow \psi$	if φ , then ψ
$\varphi \Leftrightarrow \psi$	if and only if φ , then ψ

We also have the 0-arity junctions like \top (also “top” or “verum”) with the constant value 1, as well as \perp (also “bottom” and “falsum”) with the constant value 0.

Furthermore, we assume OR and AND junctions to be n-ary (iterative conjunctions and disjunctions).

D.3 ...Our Annotations

The original definitions are (as usual) constructivistic, so only arities of 0 (constants, verum, falsum, ...), 1 (negation, choice, ...) and 2 (binary relation) are defined. With the (ordered)

Pair, the formalism is able to cope with arbitrary n-arity. In our implementation, we do want to skip this constructivistic approach and implement ordered lists straightforward.

Therefore, e.g. the triple $\langle\langle A, B \rangle, C \rangle$ or quadruple $\langle\langle\langle A, B \rangle, C \rangle, D \rangle$ are simply lists of length 3 $[A, B, C]$ resp. 4 $[A, B, C, D]$ in our formalism.

To make the nomenclature more vivid to our application of KR in a semantic network, this is how the respective objects map to a real world description:

Atom This is in our perception the typical physical object, like a *concrete* apple, a hammer, a car etc. which can be used to form a class, but are not classes themselves. They have no elements themselves.¹

Class However, in our formalism, entries are by default **not** Atoms, but classes. So the entry “apple” denotes not a concrete apple, but instead “a class of the name ‘apple’ defined by the features in its definition.” and as such describes features of that class. Of course, we could describe with “apple” the class of some specific computers and/or fruits. Or butterflies.

So what is denoted as $\{v|\varphi\}$ (“class of the v with φ ”) is in our formalism simply

$v \Rightarrow \{\varphi\}$

Where v is the id (key) of an Entry, and φ is the Meaning which itself is a formula. As a formula is also a set of By default all formulae-sets are considered to be AND-concatenated. So

```
v => {
  'x' => 'w',
  'x' => ['x', 'y'],
}
```

means “v is a kind of w AND contains as member (=has) x AND y”.

Some equivalence definitions for our formalism used:

D.3.1 Truth Value and Probability

```
[a,b]      := { a => 1, b => 1 }
-a         := { a => 0 }
{a => 1}    := { a => { P => 1 } }
x         := { x => 0 }
```

So e.g.

```
bird => {
  'x' => 'fish',
},
```

could be equally represented as

```
bird => {
```

¹In a detailed ontology, they of course do, like a car “has” or “consists of”, several other components. But somewhere the ontology has to stop, so these individuals without inner structure are atoms.

```

'☒' => {
  'fish' => 1,
  'P'    => 0,
},
},

```

Meaning “bird is not a kind of a fish”. As verbatim as possible: “The probability for a true fish to be a kind of a bird is 0”. Observe how

```

bird => {
  '☒' => {
    'fish' => 0,
  },
},

```

which - in fact - is

```

bird => {
  '☒' => '-fish',
},

```

would be a wrong interpretation of that data structure. So

```

'☒' => {
  'fish' => 1,
  'P'    => 0,
},

```

is the virtual class of “under no circumstances being a kind of true fish”.

Appendix (User Manual) E

Discourse Macro Reference

*And this unpolished rugged verse I chose
As fittest for discourse and nearest prose.
John Dryden - Religio Laici*

macro	parameter & description
ANT	Returns a regular expression matching antonyms (opposites) of the given key. lexical lookup: with diacritics
BRX	Builds a regular expression from given words/arguments. All characters are handled verbatim (therefore if you give regular expressions as arguments, these will be considered simple strings and probably not do what you intended) no lexical lookup: no diacritics
EQU	<key[,pre[,post]]> Returns a regular expression matching equivalents (synonyms) of given key. The optional pre/post parameters define a regular expression that is prepended or appended to the resulting regular expression. If not given, these default to \b (word boundary). lexical lookup: with diacritics
EVL	<code> Will evaluate the given code (arbitrary perl code) at runtime when loading the respective ontology. This happens at runtime and only if an ontology is loaded (or reloaded). Changes to the ontology at runtime do not affect running instances of the discourse engine unless a <code>bot cmd=reload</code> command is issued.
INV	Returns a regular expression matching inverses of the given key. lexical lookup: with diacritics
HPR	Returns a regular expression matching hypernyms of the given key. lexical lookup: with diacritics
HYP	Returns a regular expression matching hyponyms of the given key. lexical lookup: with diacritics
MER	Returns a regular expression matching meronyms of the given key. lexical lookup: with diacritics

OCC	<key1[,keyn]*> Optional Composita Concatenation. Given a list of strings, this macro inserts an optional delimiter. Consider this call <code>OCC<umsatz,steuer,gesetz></code> the resulting regex matches <code>umsatzsteuergesetz</code> OR <code>umsatz-steuer-gesetz</code> OR <code>umsatzsteuer gesetz</code> OR <code>umsatzsteuer-gesetz</code> etc. no lexical lookup: no diacritics
SWP	<left,center,right> Will swap the given arguments and provide an alternation regex of both the original order as well as the swapped order: <code>(leftcenterright rightcenterleft)</code> no lexical lookup: no diacritics

Table E.1: supported macros for Discourse Engine Ontologies.

Please be aware, that all arguments within a macro are taken and replaced verbatim. Therefore whitespace also is taken 1:1 as argument into account. `EQU<schön,\b,\w>` is quite different from `EQU<schön,\b, \w>`, so please take this into account when building macros.

E.1 Examples

`EQU<beautiful,\A>` This will return a regular expression matching only equivalents (synonyms) of 'beautiful' and also prepend '\A' (start of line) to the regular expression, requiring the input to start with exactly one of the given synonyms.

`EQU<schön,\b,(er|e?ste)\b>` Will return a regular expression matching only equivalents (synonyms) of 'schön' and also prepend '\b' (word boundary) so words like 'unschön' are not matched. Also, it replaces the trailing word boundary default with a german comparative/superlative suffix which results in matching words like "schön", "schöner", "schönste" but also "hübsch", "hübscher", "hübscheste".

`SWP<schoen,(.+?),\ball>` Will match input like "schöner Ball" and "der Ball ist schön", but e.g. not "Der Fußball ist schön". Because "schoen" is not prepended with a word boundary, it will also match "Der ball ist unschön".

Part III

Developer

Chapter 9

Collaborative Development

9.1 Subversion

PetaMem uses the Subversion¹ (SVN) version control system for development. There are several scenarios where you will need or want to have access to your specific data within the PetaMem version control system.

- You obtained your copy of PMLs with custom data/lexica, and use PetaMem maintenance services. This implies the need to access to the newest available version of your custom data.
- You - as a customer - perform in-house modifications to your specific lexica/data and seek for an easy and reliable way to synchronize your changes with development that has been done by PetaMem.
- As a PetaMem solution provider you perform development on behalf of a customer and need to synchronize your changes with the main development line.

If any of these scenarios apply to your situation, you will need to install Subversion on at least one of your development systems, set up a local sandbox (see below) and from time to time keep it synchronized with the PetaMem repository. In case you make your changes, commit them to the PetaMem repository.

9.1.1 SVN Access

<https://vcs.petatech.eu/svn/clients/<yourname>> is where PetaMem has set up a SVN repository for its clients. <yourname> as well as the access credentials will be - or have been - given to you by PetaMem support (support@petamem.com).

Please note that you will need at least a Subversion 1.6.x with webdav and SSL support to be able to access the repository. In case you have trouble setting up this environment, do not hesitate to contact us for help.

Both read and write access is possible with any Subversion client (native - command line, tortoise and others). Please be aware, that you always get both read and write access to your

¹see <http://subversion.apache.org/>

specific repository. Therefore great care should be taken when committing modifications.

Also, there is no restriction to how many of your in-house developers will have access to this repository. If you want to synchronize the work of 50 people, no problem.

9.1.2 Development Cycle Tutorial

This short tutorial cannot replace the Subversion documentation, but it will allow you to perform basic changes in your specific data section as well as to contribute changes done by you. We also cover the command line subversion client only, as it is the reference client to subversion. Should you prefer a client with graphical frontend like TortoiseSVN², please consult the documentation of this client.

Creating a Sandbox (checkout) A sandbox is a local directory where you can do changes to the code, test them and commit them later back to the main repository:

```
bash> svn co https://vcs.petatech.eu/svn/clients/<name> <localdir>
```

Will create a new directory <localdir> in your current directory and fetch the data from the PetaMem Subversion repository into that directory. The contents of this directory will be your local sandbox. If you omit <localdir> a directory <name> is created instead.

Updating a Sandbox (update) From time to time, or before you start doing changes to some files where possibly someone else might have done changes, you should update your sandbox. In your local sandbox, simply do:

```
bash> svn up
```

This will update your local sandbox to the newest version available in the repository. After such an update, the probability that you do changes that someone else has done already is low. Of course you should make sure that no one else is working on the same things as you do right now.

Adding and Deleting Files (add/del) Editing present files in your sandbox is just one aspect of development. You might want to extend a dataset by creating new files or consolidate old/obsolete data and then remove the obsolete files.

```
bash> svn add <file>
```

This will mark an already created file to be managed by subversion in the future. After your next commit, this file will be added to the repository. Analogously, a

²see <http://tortoisesvn.tigris.org/>

```
bash> svn del <file>
```

will mark an already existing file “for deletion from the SVN repository” which will be executed with your next commit.

Committing your changes (commit) If you have changed/added/deleted some data (and well tested your changes), you might want to make them available to other developers, so that they - as soon as they perform a `svn up` in their sandboxes receive your changes. Again, in your local sandbox simply do:

```
bash> svn commit -m <comment>
```

Where the comment is a short summary of the changes you have done. You can omit that parameter, but then probably an editor will pop up and you will have to write this comment in the editor - which is ok if you had to do a multiline comment anyway.

There are of course several other operations you will probably need to perform in your sandbox sooner or later (moving directories around (`svn mv`), resolving conflicts if someone did conflicting changes to the same files you did (`svn resolve`) and so on. It is, however, out of the scope of this small tutorial to provide a complete reference of the `svn` commands. A first help is available with the documentation the SVN provides with the `svn help` command itself, else please consult the original documentation.

9.2 Priority Of Base Ontologies

When - in the previous section - we were talking about obsolete data within your sandbox, we specifically had in mind lexica or ontologies, that became “of general interest”.

Say you had to create a customer-specific ontology for wood products, because that was not available from PetaMem. Moreover, to be able to realize this specific ontology, you also had to craft some of the underlying basic concepts for this ontology. With new revisions of the PMLS software suite, PetaMem also releases new versions of the accompanying lexica and ontologies. These versions are - in general - extensions and supersets of the old versions.

And so there might occur a situation, where some, all or even a superset of the basic concepts you had in your customer-specific sections are now available in the ontologies that are part of every PMLS shipment.

First, don't panic. Your data will not break as the specific ontologies have always higher priority than the base ontologies. On the other hand, you might want to abandon the portions of your data that are now available in the standard PMLS ontologies as this will relieve you from maintenance and development effort and might - in our humble opinion - even raise the quality of your specific ontologies, as the base ontologies tend to be very well worked out.

Chapter 10

Plugins for Text Categorization

Text categorization is a modular framework for corpora clustering. The process is consisting of following steps:

1. preprocessing corpora according to specified vector (ngrams separation)
2. ngrams filtering (inappropriate ngrams are omitted)
3. files filtering (inappropriate corpora are omitted)
4. basic groups are created according to specified vector
5. groups are being joined and recomputed according to specified vector until we have single group

Categorization process can't be generic enough to satisfy all user requirements. That's why there is a possibility to write plugins.

Plugins are simple modules providing some functionality. Our text categorization provides 3 kinds of plugins:

1. for ngrams filtering
2. for files filtering
3. for specifying vector

Let's look at all the plugin types more particularly.

10.1 Ngrams filtering

Function `filter_ngrams_generic` in `PMSE::Categorization::FilterNgrams` module is a wrapper for ngrams filtering. You can add support for new filtering into `%dispatch` variable by adding row like this:

```
my_ngram_filtering => \&my_ngram_filtering
```

where `my_ngram_filtering` is a name used in the categorization process.

Appropriate namespace for different kinds of ngrams filtering is `PMSE::Categorization::FilterNgrams::MyNgramFiltering`.

New module should contain a function that is linked from dispatcher in `PMSE::Categorization::FilterNgrams`. This function gets named argument hash with following fields:

1. `corpora_list` - list of relative paths (regarding to language root directory) to corpora
2. `cpus` - number of threads to use
3. `file` - where is expected the sum of all corpora to be stored
4. `root` - language root directory
5. `vector` - vector parameter given as option to categorization process

The categorization process expects that the function extends preprocessed data for each corpus. These data are stored in file `<language_root>/derived/<relative_path_to_corpus>/preprocessing/preprocessing.sbl`. It is storable file where the `$hash` is stored. New ngrams should be added into the form of the set (see `l2h_do_set` in `PMLIB:l2h`) into `$hash->vector_keymy_ngram_filtering`.

10.2 Files filtering

Similarly like above, function `filter_files_generic` in `PMSE::Categorization::FilterFiles` module is a wrapper for files filtering. You can add support for new filtering into `%dispatch` variable by adding row like this:

```
my_files_filtering => \&my_files_filtering
```

where `my_files_filtering` is a name used in the categorization process.

Appropriate namespace for different kinds of ngrams filtering is `PMSE::Categorization::FilterFiles::MyFilesFiltering`.

New module should contain a function that is linked from dispatcher in `PMSE::Categorization::FilterFiles`. This function gets named argument hash with following fields:

1. `corpora_list` - list of relative paths (regarding to language root directory) to corpora
2. `root` - language root directory

Return value of this function is expected to be new `corpora_list`.

10.3 Specifying vector

Vector plugins are a little more complicated than files and ngrams filtering. The new module is expected to contain 3 functions which should provide some actions.

A wrapper for vector plugins is in `PMSE::Categorization::Vector` and it is consisted of 3 functions; each corresponding to a function that we should create later.

1. `get_vector_generic`: getting vector for some corpus; this means that preprocessing data should be used to generate a structure which is sufficient to comprise all necessary information for computing distance between categories
2. `get_vector_for_cluster_generic`: getting vector for a cluster; this should store information about group created from two different groups
3. `distance_key_generic`: computing distance between two groups

To create a new plugins we should create a module like `PMSE::Categorization::Vector::MyVector` with 3 functions linked from `%dispatch` hashes from all 3 functions mentioned above.

10.4 Functions for getting corpus vector

This functions get named argument hash of the following structure:

1. `corpus` - relative path (regarding to language root directory) to corpus
2. `vector` - vector definition; i. e. a hashref with fields you need
3. `root` - language root directory
4. `filter_tokens` - which data from preprocessing.sbl file should be used (e. g. `raw`)

It should return a structure representing group. This structure will be used as input in `distance_key_generic` function later.

10.5 Functions for getting cluster vector

This functions get named argument hash of the following structure:

1. `vector` - vector definition; i. e. a hashref with fields you need
2. `group1` - structure representing 1st group
3. `group2` - structure representing 2nd group

Again, it should return a structure representing new cluster of given groups. This structure will be used as input in `distance_key_generic` function later.

10.6 Functions for computing distance

This functions get named argument hash of the following structure:

1. `group1` - structure representing 1st group
2. `group2` - structure representing 2nd group
3. `vector` - vector definition; i. e. a hashref with fields you need

It is expected to return a number; distance between `group1` and `group2`.

Appendix (Developer Manual) F

API References

Real programmers can write assembly code in any language.
Larry Wall

F.1 Semantic Lexicon

\$Rev: 560 \$

This module provides the "Semantic Lexicon Object". This type of lexicon is the designated successor of the old Lexicon.pm object (Tree::Nary based meanings). This new formalism takes more advantage of native perl structures for speed and ease of use. Also the underlying theoretical framework is more concise, yet powerful (class logic/axiomatized set theory) to offer an euivalently powerful knowledge representation formalism.

Even more, the new Semlex allows for better inference algorithms and unifies knowledge representation (no separate lexicon and morphology formalisms).

Glossary

Lexicon

This is the top-level data structure. It has a "data" and a "meta" section. The data section contains the lexical information. This information is organized as a SET OF ITEMS. Where an Item is the Pair "Key => Entry". The meta section holds non-lexical meta-information about the lexicon, such as time/date of start/last change, supported languages, size etc.

Key

In the data section of the lexicon, this is the pointer referring to an Entry. This pointer can be several things:

Word	a simple string such as "house", which will match exactly just that string of characters and is used for fast O(1) lookup of lexical information.
------	---

- Phrase** similar to word, but a whitespace separated list of words, such as "new york", "big apple" etc. This will also result in fast $O(1)$ lookup of lexical information. See PMLS::Core::NLP::Phrase for more info about Phrases and their handling.
- N-Tuple** In the normal case, both words and phrases denote CLASSES of things, not the things themselves. Of course, a particular meaning of a Word/Phrase-Key can be a single (named) element. In that case, they denote a Singleton-Class. Compound classes are denoted by a N-Tuple, which is an ordered list of words/phrases separated by an `\N{INFORMATION SEPARATOR ONE}` (which we will show here as `'.'` for visualization purposes). N-Tuples are a generalization of Pairs.
- So a N-Tuple such as `'john·box'` denotes a class. It is a compound class assembled of the classes "john" and "box". The purpose is to be compatible to a ternary relation R_3 , which is also given as a binary relation "a R_3 b". But has to describe something like "Richard gives the box to John". so "a = Richard" " R_3 = to give" "b = john·box".
- Regex** In cases, where fast $O(1)$ lookup will not deliver results, because we cannot state an exact string of characters a priori, a regular expression can be defined as key. Then, a slower $O(n)$ lookup can be performed where every key is inspected if it matches as if it were a regular expression. That lookup may deliver one or more matches.
- This is often used in cases, where we want to catch morphologically transformed word forms, but certainly do not want to have a full-form lexicon for every language.
- To speed things up, caching seems appropriate, where in a 1st run all relevant regular expressions are gathered and stored for subsequent use, so the algorithm does not need to iterate the whole lexicon with possibly millions of keys, but instead just probably some hundred regexps.

Meaning

A meaning is a hash consisting of Relations (keys) and Arguments (values). As the name says, this defines the meaning of a particular "Key" (lexical item). E.g. if you want to describe "house", one meaning of it could be the set of informations like "is a building, is an english noun, has rooms, has a chimney,..." and so on. These informations are stored by the form of relational triples a R b, where the Key pointing to this meaning, is 'a'. The keys in the Meaning-Hash are the relations R (is a, has,...) and the arguments (=values) are the 'b' of the relational triples.

Entry

As stated, an Entry is in the data-hash section of a lexicon the corresponding value to each "Key" key. Because every key (word/phrase/...) may have more than just one meaning, an Entry is a LIST OF MEANINGS, where each of these meanings denotes a distinct meaning of the referring key.

Assume again the key "house". This could mean in english 1) the building, 2) the process to house something somewhere and several other meanings (like the house of lords etc.). If our semantic lexicon would hold multilingual information - which is perfectly possible - it could also mean 3) the czech word for 'gosling'.

So an entry would consist of these 3 meanings, something like

```
0) isa => eng·N
   has => room, chimney
1) isa => eng·V
2) isa => ces·N
```

In order to be able to distinguish between the meanings when referring to such an entry, the key can optionally hold a MEANING INDEX information like /n, where n is a number denoting the index of the meaning in the Entry (=list of meanings). For 3 meanings, the index - starting at 0 - can be 0, 1 or 2.

So a reference to the 'gosling' meaning would be house/2. If no index is given, /0 is assumed implicitly. Therefore, it is good design to make the first meaning within an entry that which occurs most frequently.

Relation

The relation R plays a central role in the relation triple 'a R b', where it brings two objects 'a' and 'b' IN RELATION. As mentioned in the Meaning section above, a is the Key of the meaning the Relation is part of, b is the Argument following the relation. If there are several arguments, this is the short form of an AND-composition of relational triples. Given our example above

```
house has room, chimney
```

The 'house' is the Key (a), 'has' is our Relation (R), 'room' (b1) and 'chimney' (b2) are the arguments to the relation R. So this representation encodes two relational triples that are both valid and therefore in AND-conjunction. It is equivalent to

```
house has room AND house has chimney
```

Other predicate-logic oriented formalisms use something like has(house,room) AND has(house,chimney).

Remember, that in our formalism the Key "house" does not refer to a specific house, but rather is a representant for the "class of (all) houses". In class logic, a relation can exist between any two arbitrary objects, even relations. As such we make no distinction between what can be a relation and what is an object. Instead, our relations can also appear as Objects in our lexicon (e.g. if we have an item with the key "give" in our lexicon, which we can describe in detail).

Given the fact, that verbs have relational behaviour in natural language, this formalism translates their role in a natural way. Given e.g. the Key 'has', which is an english

verb and as such part of our lexicon. If there were several meanings to a relation rel - which can occur anytime - they'd have to be disambiguated by adding a meaning index to the relation. E.g. something like:

```
time fly/1 arrow
```

Where 'fly' (which is fly/0) would denote the insect.

Argument

Arguments are simply the "right side objects" in relational triples. As such, there is not much distinction between them and the Keys "left side objects". Probably the biggest difference is that Keys do not contain the Meaning index attached to them, while Arguments could (albeit optionally, because the default is /0 - see above.)

METHODS

Internal/Low-Level Access/Retrieve

get_keys (positional)

```
1  coderef  (optional) code that will be applied to every key.
=> listref  listref of keys satisfying given coderef-constraint
          (or all)
```

Get all keys in lex filtered by given coderef (all if no coderef given). The coderef is called by iterating every key in lexicon and passing this key as 1st argument to the coderef, the semlex object reference is passed as 2nd argument.

get_entryref (positional)

```
1  scalar   key to look up
=> listref  listref of meanings for given key, or [] if no such key
```

given entry[/midx], discard midx info and return ALL meanings (listref), or [] if no such key

get_meanref (named)

```
mean  hashref  direct meaning hashref argument
key   string   Key(+ evtl. meaning index) pointing to the meaning
                in question
midx  integer  (optional) meaning index
=>    hashref  Meaning data, empty hashref if no such meaning
```

return the meaning hashref from the given parameters. If no meaning hashref is given directly (arg: mean), it is tried to be derived from Key/Meaning Index information - which can be either "key/X" or "key" & "midx".

This method is used by other methods that need to operate on the meaning data structure.

get_rels (named)

```
mean      -> see _get_meanref
key       -> see _get_meanref
```

```

midx          -> see _get_meanref
code coderef  code that will be applied to every relation
=> listref    listref with relations matching all given constraints

```

This method uses `_get_meanref` to get the meaning reference. It gets all keys of the meaning data structure, which are all relations present. If a `coderef` is given, it is applied on every key (relation) depending on the return value (true/false), this relation is included in the result or not. If no `coderef` is given, all relations are returned.

`_get_argref (named)`

```

mean          -> see _get_meanref
key           -> see _get_meanref
midx         -> see _get_meanref
rel          -> relation or list of relations
=> listref    list of arguments for the given relation, empty list
              if no such relation

```

Returns a list reference to the arguments of a given relation (or relations). An empty listref is returned if no such relation exists in the given meaning.

`_expand_keys (positional)`

```

1 listref keys without meaning indices
=> listref returns a list of keys with all existing meaning indices

```

given a listref of keys without meaning indices [house, car, ...], this method returns a listref of these keys with all meaning indices that are present in their entries in the semlex. [house/0, house/1, car/0, car/1, car/2, ...].

Lexicon Data

`add_item (positional)`

```

1 string      key where to add the entry
2 listref     entry to add, listref of meanings. MUST be listref,
              even if a single meaning is given
=> false/listref false if error occurred, listref if all ok
              (= the new entry at key)

```

Will add a new item to lexicon. If such a key was already present, the entry is appended to the already existing entry, if it was not present, a new item in lexicon is created. Return value is the new entry.

`cmp_items`

`cpy_items (positional)`

```

1 hashref     hashref with source (key) / destination (value) pairs
              to copy
=> bool       true if all went well, false if error occurred

```

Will copy one or more items (= key + entry) to new destinations in the lexicon. If the key at the destination is not present, it will be created. If it is present, the new meaning

will be unconditionally appended to the new meaning. Basically this iteratively calls the `add_item` method for each destination key, using cloned data of the source key.

del_items (positional)

```
1 listref list of keys to delete
=> true returns always true
```

Given a listref of keys, these keys are deleted from lexicon. Returns always true, as deletion of nonexistant entries doesn't matter.

mov_items

```
1 hashref hashref with source (key) / destination (value) pairs
           to move
=> bool true if all went well, false if error occurred
```

Similar to `cpy_items`, but will move instead of copy. That is, after a copy it will remove the source from the lexicon. This is very similar to a renaming operation. The only difference is, that the moved data are being appended to evtl. existing data at the destination.

setop_lex (named)

Perform set operations on lexica.

_union_lex

Compute the set union on two lexica.

check

load

_load_lex

merge (named)

```
srclex semlex-objref source lexicon semlex objref. The lexicon which
           is merged to the method-calling semlex object.
```

This will merge two lexica, more precisely add the semlex object given as argument to the semlex object calling the merge method.

The data of the object calling this method will be assumed "BASE data", so if there are any key clashes, the keys of the added lexicon are transposed, i.e. their meanings are put behind the already existing meanings in the base data. All references in the `srclex` are modified accordingly.

save

sweep (positional)

```
1 coderef code reference to a subroutine that will be called on
           each visit of an atom/argument
```

This iterator wades through every argument in every relation in every meaning of every key in a lexicon. The `coderef` is called and informed about the current level: 0

= argument iteration 1 = relation (ended args iteration) 2 = meaning (ended relations iteration) 3 = key (ended meanings iteration) 4 = final call (sweep is about to end))

The coderef also gets a named arguments hashref from the sweep iterator with several named parameters to use:

```

lvl  iteration level (see above)
lex  the lexicon hashref      (levels: -3 to 3)
key  the current key         (levels: -3 to 3)
means listref of all meanings (levels: -3 to 3)
midx the meaning index       (levels: 0 to 2)
rel  the iterated relation   (levels: 0 to 1)
rels listref of all relations (levels: -2 to 2)
arg  the current argument    (levels: 0)
aidx the arguments index     (levels: 0)
args a listref of all args    (levels: -1 to 1)

```

If NO coderef is given, sweep will use a default that des a count of all atoms in the given lexicon, thus returning the number of all single atoms/arguments in the lexicon.

ensure_lexpath

Lexicon Meta

info (positional)

Internal method - backend for info().

info (positional)

```
1 listref list of meta-data keys to force recomputation.
```

Print out meta information of current Semlex. The given list specifies those keys whose values will be recomputed.

clr_meta (positional)

```
1 listref list of keys whose values are to clear in meta structure.
1 '*'      alternatively this wildcard indicates that all values
           shall be cleared
```

This will take a list of keys and in the meta structure set their values to undef. Is required by the internal `_info` method as it defines which values are to be recomputed.

get_meta_arg (positional)

```
1 scalar key of meta structure to define value to retrieve
=> any   will return the stored value for a given key.
```

Basic semlex meta structure. Gets value of a given key.

set_meta_arg

```
1 scalar key of meta structure to define value to set
2 any   value to set for key
=> any  will return the old value for the given key
```

Basic semlex meta structure accessor. Sets value for a given key. Although parameters are positional and could be written such as

```
$self->set_meta_arg($key, $value)
```

you will often find the "fat comma" notation

```
$self->set_meta_arg($key => $value)
```

Which is equivalent and should not confuse you.

get_sltl (void)

```
=> listref list with 2 elements: 1. hashref with SL,
          2. hashref with TL
```

Will return a list of 2 elements, 1st element is a hashref containing all source languages covered by this semlex, where key is the iso code of the language and value is an integer with the number of occurrences. 2nd element is analogous for target languages covered by this semlex.

get_sxl (positional)

```
1 scalar 'ssl' (default) - source languages
          or 'stl' - target languages
=> hashref hash containing keys of supported languages and values
          of number of entries for this language
```

This will return a hash with the information of language support for the method-calling semlex object. If given no argument, 'ssl' or an unknown string, the hash with the "supported source languages" (hence ssl) is returned. The keys are the iso639-3 codes of the supported languages, the values are the numbers of entries in that respective lexicon for that given language.

'stl' is similar, but covers the "supported target languages".

Evidently this is used for assessment whether the method-calling lexicon has translational coverage for a given source/target language combination, although there is no guarantee that every ssl/stl combination is supported, as the data is gathered as a total.

This data is fetched from the lexicon meta-information data structure. If the values for ssl or stl are undefined, the get_sltl method is called which fills these values to the meta structure. Calling get_sltl can be very slow with large lexica, therefore the caching of its output in the meta data structure. Because of this caching, the information retrieved may not be accurate.

discussion: there were concerns if storing all supported key/value pairs would not be a better solution. It has turned out, however, that by inferring translational info using symmetric features of the translational relation, we in fact might get all possible combinations anyway.

set_sxl (positional)

```
1 scalar "see get_sxl"
2 hashref hash with <iso> => <number> combinations for the given
          source/target languages
```

=> hashref old value previously stored

Basically a frontend to the `set_meta_arg` method, this adds arguments checking for ssl/stl validity.

get_date

set_date

modified (void)

=> bool Will return true if lexicon has been modified, false else

The timestamps for meta data 'change' and 'start' are compared. If 'change' is set to a later point than 'start', true is returned, else false.

name (positional)

1 scalar (optional), if given, stores this as name of the lexicon
 => scalar if arg 1 given: returns the old value before the set
 => scalar if arg 1 not given: returns the current value

This method will get or set the lexicon name. If argument1 is given, this will store it as the new name of the lexicon. Please be aware, that by this the key-name under which the lexicon is stored in the repository is not changed! The old value (previous name) is returned. (therefore this could be used to perform a subsequent "lex_mov old-name new-name" in Repository.

If no argument is given, the current lexicon name is returned.

Entry

get_meanings_num (positional)

uses `_get_entryref` to return the number of meanings

out_item

Output item in a specified format.

Meaning

Meanings/Meaning

cmp_meanings

Compare 2 meanings.

Relation

get_rel_chain (named)

get_rel_complements (positional)

```

1  scalar  key of relation we want the complementary relations of
=> listref reference to list of relations that are complementary
      to the relation given

```

Will consider the given key as a relation and try to find the converse relations to it. An empty listref is returned if there are no converse relations. Which basically should not happen, as the system will try hard to find an converse relation, even by heuristically construing it.

get_rel_inverses (positional)

```

1  scalar  key of relation we want the inverse relations of
=> listref reference to list of relations that are inverse to the
      relation given

```

In a similar fashion to `get_rel_complements`, this method will return a list of inverse relations to a given relation.

is_transitive (positional)

```

1  string  relation to check

```

add_arguments

del_arguments

has_argument

get_translations

get_common

```

### START COMPATIBILITY THROWAWAY

```

isin

```

### END COMPATIBILITY THROWAWAY

```

Syntax

morph_key (positional)

```

1
=>

```

_morph_key_direct (positional)

```

1
=>

```

_morph_key_eval (positional)

```

1
=>

```

Inference

get_equ

get EQU-clique from a single key.

EQU (positional)

```
1
=>
```

get EQU-clique from a list of keys.

_get_clique (named)

key	scalar	object to look for
limit	int	recursion limit (default: -1 = no limit)
mem	hashref	memory hash for visited keys
midx	int	(optional) meaning index to <key>
rel	scalar	relation to apply, default: EQU
rtype	scalar	return type (default: 'lr' = listref, 'hr' = hashref)
=>	listref	if rtype=lr specified, list of clique objects
=>	hashref	if rtype=hr specified, hash of clique objects

Given an object and a relation, find all objects in same "clique" (sharing that relation and be connected). For -transitive relations, this is identical to { '_get_argref' + \$key } For transitive relations, the clique is returned.

As the default relation is EQU, _get_clique with no relation given is the same as "get synonyms/equivalents".

_get_hyper (named)

get hypernyms of a given key (those forming a clique by the ISA relation).

Helper Functions (no methods)

ki2k (positional)

1	scalar	key or key/idx form of key
=>	scalar	just the key-part of key/idx

internal function to just return the 1st part of a key/index combination (= key)

l2l_ki2k (positional)

1	listref	list of key/index values (implicit or explicit)
=>	listref	unique keys of input list

This function takes a listref of key/index values, where there might be duplicates, keys may have implicit or explicit form. It returns a list of just the key-parts and guarantee that the returned keys are unique.

ki2i (positional)

1	scalar	key or key/idx form of key
=>	scalar	(integer) just the index-part of key/idx

internal function to just return the 2nd part of a key/index combination (= meaning index)

build_inv (positional)

```
1 scalar key (interpreted as relation) to build inverse version of
=> scalar inverse relation (construed) of given key
```

This will interpret the given key as being a relation and construct its inverse. If the key was already a construed inverse relation, the squashing of double-inverse information, will result in returning the original relation again.

build_neg (positional)

```
1 scalar key (interpreted as relation) to build negated version of
=> scalar negated/complementary relation (construed) of given key
```

Similar to `build_inv`, this will interpret the given key as being a relation and construct its negation/complement. If the key was already a construed negate/complement relation, the squashing of double-negated information, will result in returning the original relation again.

expkey (positional)

```
1 scalar key to build an explicit version of
```

impkey

```
1 scalar key to build an implicit version of
```

joinkey (positional)

```
1 scalar key
2 scalar meaning-index
=> scalar "key/meaning index"
```

Basically the inverse operation to `splitkey`, see there.

Be sure to give only valid values to this function, as there are no sanity checks made. I.e. giving non-numeric values as meaning-index may result in unexpected behavior. Also, giving an already combined "key/value" as key, may prove fatal later on.

splitkey (positional)

```
1 scalar "key/meaning index"
=> list (key, meaning_index)
```

The `joinkey/splitkey` functions handle constructing and deconstructing operations on the key/meaning pair. As there can be several meanings for a given key (word, phrase, ...), it is necessary to be able to distinguish between them. Therefore, the topmost structure as value for any given key, is the listref. In this list, meanings (each a hashref) are kept. To be able to determine to what meaning a given key refers to, the suffix `"/<number>"` may be **OPTIONALLY** added.

If no such suffix is added, the default meaning index is 0. Else, the number denotes the index of the meaning in the listref structure. Because of this, "key" and "key/0" are equivalent.

F.2 Discourse Engine ChatBot

\$Rev: 560 \$

The PMLS Discourse Engine Bot/Chatbot implementation. This is intended for interactive communication with humans.

METHODS

AUTOMETHOD

Catch unknown methods. If the ontology file contains a method reference that has no counterpart in this code, we have to catch it and return some message instead of failing miserably.

BUILD

Object constructor.

Top-Level Functionality

get_answer

Toplevel method. Calls `set_input`, after that performs an all-over search in the ontology until it finds a match. If a match is found, the return value of `construct_answer` is returned.

info

get_number_of_rules

Input Processing

set_input

This method stores and processes the user's input. There are 4 data structures this user input is stored in:

input_orig

The original user input as string. The most authentic source.

input_orig_rtl

A raw tokens list made of the original user input string. The split is made with the regular expression `(\p{Z}|\n|\p{P})` to split after a "separator" a newline and a punctuation. (remark: it is not fully clear why we need `\n` because this should be covered by the `\p{Z}` class.)

input_proc

input_proc_rtl

get_token_orig

Given a processed token or tokens, return the original token.

Bot Knowledge Processing

The bot knowledge block contains 3 different sources of knowledge:

- 1) the 'common sense' which is provided by the global repository object
- 2) the bot-specific ontology data, which are the ontologies for the languages of the bot
- 3) the bot-specific facts which is knowledge the bot obtains during discourse. Stored in a private repository object

get_ontology

Returns hashref to the ontologies loaded.

set_ontology

Expects a hashref with the ontologies, such as the return value of the `load_ontology` function.

load_ontology

Function (not oobject method). Loads the ontology/ontologies for requested languages (default 'eng') from disk. Returns a hashref with the ontologies loaded.

reload_ontology

Reloads the required ontologies from disk. This is triggered by the 'bot cmd=reload' command and useful when developing an ontology, so you do not have to restart PMLS for every check you made to the ontology.

get_myfacts

get_bot_langs

Returns listref of iso639-3 codes of languages the bot is aware of.

get_bot_langs_txt

Returns the list of languages the bot is aware of in natural language format, in the respective language given for localization.

Bot State Processing

get_bot_age

Get the age of the bot by comparing birth timestamp with current timestamp.

get_bot_birth

Retrieve the timestamp of bots inception.

get_bot_lastact

Retrieve timestamp of bots last activity.

set_bot_lastact

Set bots last activity timestamp to timestamp given or current timestamp if none given.

get_bot_mood

add_topic_to_memo

get_topics_memo

get_bot_numinput

get_bot_numanswers

inc_bot_numinput

inc_bot_numanswers

answer

ONTOLOGY-REFERENCED METHOD. This is the most basic answering method referenced in an ontology, it will basically generate an answer without too much hassle. Other ontology-referenced methods such as calc, date or parse will do special things that are out of the scope of a simple (store-able) question-answer scheme.

calc

ONTOLOGY-REFERENCED METHOD. This will perform user-initiated calculations. Say the user needs a calculator or just wants to test the "smartness" of the chatbot. He might ask some calculation-related questions and this method is here to provide/compute the answers. Because this kind of information can not be stored in advance, it could not be answered with the answer method.

date

ONTOLOGY-REFERENCED METHOD. Will perform all sorts of date manipulations and provide answers to date-related questions. Same as with calc method, these are informations that cannot be stored in advance and thus have to be computed ad hoc.

interpolate

Callback function for the ReadINI function of Config::IniHash. Will perform variable replacement of the variables that were defined in vars.ont thus providing us with a macro-like feature.

load_state

replace

save_state

set_memo

F.3 APIs for PMLIB:: Namespace

F.3.1 PMLIB::Association

Wrapper for various association measures.

Functions Reference

score (named)

```
function str    name of measure
args         href  params for selected measure
=>          href  scores
```

function could be 't' for t-score or 'mi' for MI-score. The others depends on implementation.

For example 'mi' and 't' functions accepts href like this one:

```
ngram          str    ngram as a string
separ          char   character that separates tokens in ngram
tm_table       href   true-marginals table
dependence     href   which score?
n              int    size of 'n'gram
marginal_char  str    indicates 'marginal token' in tm_table
trust_dependence bool  check structure of dependence or trust it?
```

It computes selected score (function) of some ngram separated by separ based of true-marginals table tm_table with defined dependence.

<tm_table> is of the structure key => count for key as ngram and true-marginals. True-marginals contain marginal_char (by default *) which indicates 'sum over all tokens'. Please, look at tm_table data structure example for bigram:

```
{'a *' => 10, 'a b' => 4, '* b' => 6, '* *' => 25};
```

This means:

```
4times occurrence of observed bigram
10times occurrence of bigram with our 1st token
6times occurrence of bigram with our 2nd token
25times occurrence of anything
```

Here is dependence data structure example for 5gram:

```
[[1, 2], [3, 5], [4]]
```

It means that score of 5gram abcde will be computed as:

```
log2 [P(abcde)/P(ab)P(ce)P(d)]
```

You can also omit independent tokens. Equivalent parameter:

```
[[1, 2], [3, 5]]
```

scores (named)

```
function str    name of measure
args         href  params for selected measure
=>          href  scores
```

function could be 't' for t-score or 'mi' for MI-score. The others depends on implementation.

For example 'mi' and 't' function accepts href like this one:

ngram	string	ngram as a string
separ	char	character that separates tokens in ngram
tm_table	href	true-marginals table
n	int	size of 'n'gram
marginal_char	string	indicates 'marginal token' in tm_table
trust_dependence	bool	check structure of dependence or trust it?

Computes selected score (function) of some ngram separated by separ based of true-marginals table tm_table for all dependences.

Parameters are same as in x2s_score, except we needn't to specify dependences, because we want to compute all of them.

Returns href like

```
'1|2 3' => 0.25153876...,
'1 3|2' => -0.04963076...,
'1 2|3' => 0.25153876...,
'1|2|3' => 0.40354186...,
```

where keys are individual dependences. For example '1|2 3' means that second and third tokens are dependent and they are independent with first one.

F.3.2 PMLIB::Association::Mi

Routines for MI-score computations.

Functions Reference

mi_score (named)

ngram	str	ngram as a string
separ	char	character that separates tokens in ngram
tm_table	href	true-marginals table
dependence	lref	which MI score?
n	int	size of 'n'gram
marginal_char	str	indicates 'marginal token' in tm_table
trust_dependence	bool	check structure of dependence or trust it?
=>	real	mutual information

computes MI score - mutual information - of some ngram based of true-marginals table. The parameter href:

tm-table data structure example for bigram:

```
{'a *' => 10, 'a b' => 4, '* b' => 6, '* *' => 25};
```

This means:

```
4times occurrence of observed bigram
10times occurrence of bigram with our 1st token
6times occurrence of bigram with our 2nd token
```

25times occurrence of anything

Dependence data structure example for 5gram:

```
[[1, 2], [3, 5]]
```

It means that MI score of 5gram abcde will be computed as:

$$\log_2 [P(abcde)/P(ab)P(ce)P(d)]$$

mi_score_vector (named)

ngram	str	ngram as a string
separ	char	character that separates tokens in ngram
tm_table	href	true-marginals table
n	int	size of 'n'gram
marginal_char	str	indicates 'marginal token' in tm_table
trust_dependence	bool	check structure of dependence or trust it?
=>	href	all mutual informations

Calls MI score function for all possible dependencies or specified one.

Returns hashref like

```
'1|2 3' => 0.25153876...,
'1 3|2' => -0.04963076...,
'1 2|3' => 0.25153876...,
'1|2|3' => 0.40354186...,
```

where keys are individual dependences

F.3.3 PMLIB::Association::T

Routines for t-score computations.

Functions Reference

t_score (named)

ngram	str	ngram as a string
separ	char	character that separates tokens in ngram
tm_table	href	true-marginals table
dependence	lref	which T score?
n	int	size of 'n'gram
marginal_char	str	indicates 'marginal token' in tm_table
trust_dependence	bool	check structure of dependence or trust it?
=>	real	t-score

Computes t-score - measure of contrast.

t_score_vector (named)

ngram	str	ngram as a string
separ	char	character that separates tokens in ngram
tm_table	href	true-marginals table

n	int	size of 'n'gram
marginal_char	str	indicates 'marginal token' in tm_table
trust_dependence	bool	check structure of dependence or trust it?
=>	href	all t-scores

Call t-score for all possible dependencies or specified one. Returns href like:

```
'1|2 3' => 0.25153876...,
'1 3|2' => -0.04963076...,
'1 2|3' => 0.25153876...,
'1|2|3' => 0.40354186...,
```

where keys are individual dependences.

F.3.4 PMLIB::Config

Provides the API for handling the "central configuration object". This is basically a perl structure representing the parsed config file

Methods Reference

get (positional)

1	str	section
2	str	key
=>	str	value

load (positional)

1	str	file
=>	struct	loaded conf

Loads config from file.

out (positional)

1	list	sections
=>	str	formatted string

set (positional)

1	str	section
2	str	key
3	str	value
=>	undef	undef

F.3.5 PMLIB::Conv

Wrapper for various file conversions.

Functions Reference

active_template (named)

```

cmd      str      shell cmd name
args     str      params for C<cmd>
stdout   rx        stdout check that must pass
stderr   rx        stderr check that must pass
=>      bool     all cmds available?

```

Warnings for non-installed convertors are printed if `PM_CONVERTOR_WARNINGS` is set.

cmd (positional)

```

1  str      cmd
2  str      params
3  bool     output stderr?
=> (str,str) (stdout, stderr) of command if C<3>
=> str      stdout of command unless C<3>

```

Faster alternative of `myrun` from `PMLIB::System`.

conv_generic (named)

```

to      str      format
file    str      abs. filename to convert from detected format to C<to>
store   str      where to store results
=>      lref     resulting files

```

Creates chains of conversion to get desired conversion. In `store` directory there will be created file level based structure. Each level represents one derivation step. The most important directory here is `lvl.last` containing symlinks to all `.txt` files in resting `lvls`.

dump_converters (positional)

```

1  bool     flag: dump all
=> lref     listref of converters

```

If no or 0 value given, function will return installed converters only converters.

find_convertor (positional)

```

1  str      conversion module name
=> lref     name of the module for picked convertor

```

work_collector_template (positional)

```

1  str      conversion module name
=> lref     list of new files

```

Calls some tool module to make an action.

work_template (named)

```

in          str      file to convert
sub         cref     conversion handler with parameter = out file
outdir      str      dir to create result file (default: as C<in>)
extension_from str    input format

```

```

extension_to    str    output format
=>              any    retval of C<sub> (usually new filename)

```

F.3.6 PMLIB::Conv::7z2_

PetaMem Library: 7z2_ convertors collector

Functions Reference

active (void)

```

=>    bool    do we have all dependencies to make this conversion?

```

work (positional)

```

1     str    file to convert
2     str    outdir (optional)
=>    lref   resulting file names

```

Convert file from 7z. Filename and suffix are guessed automatically.

F.3.7 PMLIB::Conv::7z2_::p7zip

PetaMem Library: 7z2_ converter

Functions Reference

active (void)

```

=>    bool    do we have all dependencies to make this conversion?

```

work (positional)

```

1     str    file to convert
2     str    outdir (optional)
=>    lref   resulting file names

```

Convert file from 7z using 7z. Filename and suffix are guessed automatically.

F.3.8 PMLIB::Conv::ace2_

PetaMem Library: ace2_ convertors collector

Functions Reference

active (void)

```

=>    bool    do we have all dependencies to make this conversion?

```

work (positional)

```

1     str    file to convert
2     str    outdir (optional)
=>    lref   resulting file names

```

Convert file from ace. Filename and suffix are guessed automatically.

F.3.9 PMLIB::Conv::ace2_::ACE_Compression_Software

PetaMem Library: ace2_converter

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from ace using unace. Filename and suffix are guessed automatically.

F.3.10 PMLIB::Conv::ar2_

PetaMem Library: ar2_convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from ar. Filename and suffix are guessed automatically.

F.3.11 PMLIB::Conv::ar2_::GNU

PetaMem Library: ar2_converter

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from ar using ar. Filename and suffix are guessed automatically.

F.3.12 PMLIB::Conv::arc2_

PetaMem Library: arc2_convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from arc. Filename and suffix are guessed automatically.

F.3.13 PMLIB::Conv::arc2_::ARC

PetaMem Library: arc2_converter

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from arc using arc-5.21p. Filename and suffix are guessed automatically.

F.3.14 PMLIB::Conv::arj2_

PetaMem Library: arj2_convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from arj. Filename and suffix are guessed automatically.

F.3.15 PMLIB::Conv::arj2_::ARJ_Software

PetaMem Library: arj2_converter

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from arj using unarj. Filename and suffix are guessed automatically.

F.3.16 PMLIB::Conv::azw32txt

PetaMem Library: azw32txt convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from azw3 to txt. Filename and suffix are guessed automatically.

F.3.17 PMLIB::Conv::azw32txt::calibre

PetaMem Library: azw32txt converter

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from azw3 to txt using ebook-convert. Filename and suffix are guessed automatically.

F.3.18 PMLIB::Conv::bz22_

PetaMem Library: bz22_ convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from bz2. Filename and suffix are guessed automatically.

F.3.19 PMLIB::Conv::bz22_::Seward

PetaMem Library: bz22_ converter

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from bz2 using bzip2. Filename and suffix are guessed automatically.

F.3.20 PMLIB::Conv::cab2_

PetaMem Library: cab2_ convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from cab. Filename and suffix are guessed automatically.

F.3.21 PMLIB::Conv::cab2_::cabextract

PetaMem Library: **cab2_converter**

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from cab using cabextract. Filename and suffix are guessed automatically.

F.3.22 PMLIB::Conv::chm2pdf

PetaMem Library: **chm2pdf_convertors collector**

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from chm to pdf. Filename and suffix are guessed automatically.

F.3.23 PMLIB::Conv::chm2pdf::chm2pdf

PetaMem Library: **chm2pdf converter**

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from chm to pdf using chm2pdf. Filename and suffix are guessed automatically.

F.3.24 PMLIB::Conv::cpio2_

PetaMem Library: cpio2_ convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from cpio. Filename and suffix are guessed automatically.

F.3.25 PMLIB::Conv::cpio2_::GNU

PetaMem Library: cpio2_ converter

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from cpio using cpio. Filename and suffix are guessed automatically.

F.3.26 PMLIB::Conv::cpio2_::bsdcpio

PetaMem Library: cpio2_ converter

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from cpio using cpio. Filename and suffix are guessed automatically.

F.3.27 PMLIB::Conv::djvu2txt

PetaMem Library: djvu2txt convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from djvu to txt. Filename and suffix are guessed automatically.

F.3.28 PMLIB::Conv::djvu2txt::DjVuLibre

PetaMem Library: djvu2txt convertor

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from djvu to txt using DjVuLibre. Filename and suffix are guessed automatically.

F.3.29 PMLIB::Conv::doc2txt

PetaMem Library: doc2txt convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from doc to txt. Filename and suffix are guessed automatically.

F.3.30 PMLIB::Conv::doc2txt::antiword

PetaMem Library: doc2txt converter

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from doc to txt using antiword. Filename and suffix are guessed automatically.

F.3.31 PMLIB::Conv::docx2txt

PetaMem Library: docx2txt convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from docx to txt. Filename and suffix are guessed automatically.

F.3.32 PMLIB::Conv::docx2txt::Kumar

PetaMem Library: docx2txt converter

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file name
```

Convert file from docx to txt using docx2txt. Filename and suffix are guessed automatically.

F.3.33 PMLIB::Conv::dvi2pdf

PetaMem Library: dvi2pdf convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from dvi to pdf. Filename and suffix are guessed automatically.

F.3.34 PMLIB::Conv::dvi2pdf::Artifex

PetaMem Library: dvi2pdf converter

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from dvi to pdf using dvipdf. Filename and suffix are guessed automatically.

F.3.35 PMLIB::Conv::epub2txt

PetaMem Library: epub2txt convertors collector

Functions Reference

active (void)

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```

1   str   file to convert
2   str   outdir (optional)
=>  lref  resulting file names

```

Convert file from epub to txt. Filename and suffix are guessed automatically.

F.3.36 PMLIB::Conv::epub2txt::calibre

PetaMem Library: epub2txt converter

Functions Reference

active (void)

```

=>  bool   do we have all dependencies to make this conversion?

```

work (positional)

```

1   str   file to convert
2   str   outdir (optional)
=>  lref  resulting file names

```

Convert file from epub to txt using ebook-convert. Filename and suffix are guessed automatically.

F.3.37 PMLIB::Conv::fb22txt

PetaMem Library: fb22txt convertors collector

Functions Reference

active (void)

```

=>  bool   do we have all dependencies to make this conversion?

```

work (positional)

```

1   str   file to convert
2   str   outdir (optional)
=>  lref  resulting file names

```

Convert file from fb2 to txt. Filename and suffix are guessed automatically.

F.3.38 PMLIB::Conv::fb22txt::calibre

PetaMem Library: fb22txt converter

Functions Reference

active (void)

```

=>  bool   do we have all dependencies to make this conversion?

```

work (positional)

```

1   str   file to convert
2   str   outdir (optional)
=>  lref  resulting file names

```

Convert file from fb2 to txt using ebook-convert. Filename and suffix are guessed automatically.

F.3.39 PMLIB::Conv::fodt2txt

PetaMem Library: fodt2txt convertors collector

Functions Reference

active (void)

```

=>  bool   do we have all dependencies to make this conversion?

```

work (positional)

```

1   str   file to convert
2   str   outdir (optional)
=>  lref  resulting file names

```

Convert file from fodt to txt. Filename and suffix are guessed automatically.

F.3.40 PMLIB::Conv::fodt2txt::unoconv

PetaMem Library: fodt2txt converter

Functions Reference

active (void)

```

=>  bool   do we have all dependencies to make this conversion?

```

work (positional)

```

1   str   file to convert
2   str   outdir (optional)
=>  lref  resulting file names

```

Convert file from fodt to txt using unoconv. Filename and suffix are guessed automatically.

F.3.41 PMLIB::Conv::gif2png

PetaMem Library: gif2png convertors collector

Functions Reference

active (void)

```

=>  bool   do we have all dependencies to make this conversion?

```

work (positional)

```

1   str   file to convert
2   str   outdir (optional)
=>  lref  resulting file names

```

Convert file from gif to png. Filename and suffix are guessed automatically.

F.3.42 PMLIB::Conv::gif2png::ImageMagick

PetaMem Library: gif2png converter

Functions Reference**active (void)**

```

=>  bool   do we have all dependencies to make this conversion?

```

work (positional)

```

1   str   file to convert
2   str   outdir (optional)
=>  lref  resulting file names

```

Convert file from gif to png using ImageMagick. Filename and suffix are guessed automatically.

F.3.43 PMLIB::Conv::gz2_

PetaMem Library: gz2_ convertors collector

Functions Reference**active (void)**

```

=>  bool   do we have all dependencies to make this conversion?

```

work (positional)

```

1   str   file to convert
2   str   outdir (optional)
=>  lref  resulting file names

```

Convert file from gz. Filename and suffix are guessed automatically.

F.3.44 PMLIB::Conv::gz2_::GNU

PetaMem Library: gz2_ converter

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from gz using gzip. Filename and suffix are guessed automatically.

F.3.45 PMLIB::Conv::html2txt**PetaMem Library: html2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from html to txt. Filename and suffix are guessed automatically.

F.3.46 PMLIB::Conv::html2txt::mech_dump**PetaMem Library: html2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting filenames
```

Convert file from html to txt using mech-dump. Filename and suffix are guessed automatically.

F.3.47 PMLIB::Conv::htmlz2txt**PetaMem Library: htmlz2txt convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from htmlz to txt. Filename and suffix are guessed automatically.

F.3.48 PMLIB::Conv::htmlz2txt::calibre**PetaMem Library: htmlz2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from htmlz to txt using ebook-convert. Filename and suffix are guessed automatically.

F.3.49 PMLIB::Conv::jpg2txt**PetaMem Library: jpg2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from jpg to txt. Filename and suffix are guessed automatically.

F.3.50 PMLIB::Conv::jpg2txt::tesseract**PetaMem Library: jpg2txt converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from jpg to txt using Tesseract OCR. Filename and suffix are guessed automatically.

F.3.51 PMLIB::Conv::lha2_**PetaMem Library: lha2_ convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lha. Filename and suffix are guessed automatically.

F.3.52 PMLIB::Conv::lha2_::Howard**PetaMem Library: lha2_ converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lha using lha. Filename and suffix are guessed automatically.

F.3.53 PMLIB::Conv::lha2_::Okamoto**PetaMem Library: lha2_ converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lha using lha. Filename and suffix are guessed automatically.

F.3.54 PMLIB::Conv::lit2txt**PetaMem Library: lit2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lit to txt. Filename and suffix are guessed automatically.

F.3.55 PMLIB::Conv::lit2txt::calibre**PetaMem Library: lit2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lit to txt using ebook-convert. Filename and suffix are guessed automatically.

F.3.56 PMLIB::Conv::lrf2txt**PetaMem Library: lrf2txt convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lrf to txt. Filename and suffix are guessed automatically.

F.3.57 PMLIB::Conv::lrf2txt::calibre**PetaMem Library: lrf2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lrf to txt using ebook-convert. Filename and suffix are guessed automatically.

F.3.58 PMLIB::Conv::lrz2_**PetaMem Library: lrz2_ convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lrz. Filename and suffix are guessed automatically.

F.3.59 PMLIB::Conv::lrz2_::lrzip**PetaMem Library: lrz2_ converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting filenames
```

Convert file from lrz using lrzip. Filename and suffix are guessed automatically.

F.3.60 PMLIB::Conv::lzma2_**PetaMem Library: lzma2_ convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lzma. Filename and suffix are guessed automatically.

F.3.61 PMLIB::Conv::lzma2_::XZ_Utils**PetaMem Library: lzma2_ converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lzma using lzma. Filename and suffix are guessed automatically.

F.3.62 PMLIB::Conv::lzx2_**PetaMem Library: lzx2_ convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lzx. Filename and suffix are guessed automatically.

F.3.63 PMLIB::Conv::lzx2_::unlzx**PetaMem Library: lzx2_ converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from lzx using unlzx. Filename and suffix are guessed automatically.

F.3.64 PMLIB::Conv::mobi2txt**PetaMem Library: mobi2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from mobi to txt. Filename and suffix are guessed automatically.

F.3.65 PMLIB::Conv::mobi2txt::calibre**PetaMem Library: mobi2txt converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from mobi to txt using ebook-convert. Filename and suffix are guessed automatically.

F.3.66 PMLIB::Conv::odt2txt**PetaMem Library: odt2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from odt to txt. Filename and suffix are guessed automatically.

F.3.67 PMLIB::Conv::odt2txt::Stosberg**PetaMem Library: odt2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting filenames
```

Convert file from odt to txt using odt2txt. Filename and suffix are guessed automatically.

F.3.68 PMLIB::Conv::odt2txt::unoconv**PetaMem Library: odt2txt converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting filenames
```

Convert file from odt to txt using odt2txt. Filename and suffix are guessed automatically.

F.3.69 PMLIB::Conv::ott2txt**PetaMem Library: ott2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from ott to txt. Filename and suffix are guessed automatically.

F.3.70 PMLIB::Conv::ott2txt::unoconv**PetaMem Library: ott2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from ott to txt using unoconv. Filename and suffix are guessed automatically.

F.3.71 PMLIB::Conv::pbz22_**PetaMem Library: pbz22_ convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from pbz2. Filename and suffix are guessed automatically.

F.3.72 PMLIB::Conv::pbz22_::Gilchrist**PetaMem Library: pbz22_ converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from pbz2 using pbzip2. Filename and suffix are guessed automatically.

F.3.73 PMLIB::Conv::pdb2txt**PetaMem Library: pdb2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from pdb to txt. Filename and suffix are guessed automatically.

F.3.74 PMLIB::Conv::pdb2txt::txt2pdoc**PetaMem Library: pdb2txt converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from pdb to txt using txt2pdbdoc. Filename and suffix are guessed automatically.

F.3.75 PMLIB::Conv::pdf2txt**PetaMem Library: pdf2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from pdf to txt. Filename and suffix are guessed automatically.

F.3.76 PMLIB::Conv::pdf2txt::Poppler**PetaMem Library: pdf2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from pdf to txt using pdftotext. Filename and suffix are guessed automatically.

F.3.77 PMLIB::Conv::pet2tgz**PetaMem Library: pet2tgz convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from pet to tgz. Filename and suffix are guessed automatically.

F.3.78 PMLIB::Conv::pet2tgz::Puppy**PetaMem Library: pet2tgz converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from pet to tgz using pet2tgz. Filename and suffix are guessed automatically.

F.3.79 PMLIB::Conv::png2txt**PetaMem Library: png2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from png to txt. Filename and suffix are guessed automatically.

F.3.80 PMLIB::Conv::png2txt::tesseract**PetaMem Library: png2txt converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from png to txt using Tesseract OCR. Filename and suffix are guessed automatically.

F.3.81 PMLIB::Conv::ps2pdf**PetaMem Library: ps2pdf convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from ps to pdf. Filename and suffix are guessed automatically.

F.3.82 PMLIB::Conv::ps2pdf::gs**PetaMem Library: ps2pdf converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from ps to pdf using ps2pdf. Filename and suffix are guessed automatically.

F.3.83 PMLIB::Conv::rar2_**PetaMem Library: rar2_ convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from rar. Filename and suffix are guessed automatically.

F.3.84 PMLIB::Conv::rar2::Roshal**PetaMem Library: rar2_converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from rar using unrar. Filename and suffix are guessed automatically.

F.3.85 PMLIB::Conv::rpm2cpio**PetaMem Library: rpm2cpio_convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from rpm to cpio. Filename and suffix are guessed automatically.

F.3.86 PMLIB::Conv::rpm2cpio::Troan**PetaMem Library: rpm2cpio_converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting filenames
```

Convert file from rpm to cpio using rpm2cpio. Filename and suffix are guessed automatically.

F.3.87 PMLIB::Conv::rst2pdf**PetaMem Library: rst2pdf convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from rst to pdf. Filename and suffix are guessed automatically.

F.3.88 PMLIB::Conv::rst2pdf::rst2pdf**PetaMem Library: rst2pdf converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from rst to txt using rst2pdf. Filename and suffix are guessed automatically.

F.3.89 PMLIB::Conv::rtf2txt**PetaMem Library: rtf2txt convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from rtf to txt. Filename and suffix are guessed automatically.

F.3.90 PMLIB::Conv::rtf2txt::GNU**PetaMem Library: rtf2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from rtf to txt using unrtf. Filename and suffix are guessed automatically.

F.3.91 PMLIB::Conv::rz2_**PetaMem Library: rz2_ converters collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from rz. Filename and suffix are guessed automatically.

F.3.92 PMLIB::Conv::rz2_::Tridgell**PetaMem Library: rz2_ converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from rz using rzip. Filename and suffix are guessed automatically.

F.3.93 PMLIB::Conv::sea2sit**PetaMem Library: sea2sit convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from sea to sit. Filename and suffix are guessed automatically.

F.3.94 PMLIB::Conv::sea2sit::unsea**PetaMem Library: sea2sit converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from sea to sit using unsea. Filename and suffix are guessed automatically.

F.3.95 PMLIB::Conv::shar2_**PetaMem Library: djvu2txt convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from shar to txt. Filename and suffix are guessed automatically.

F.3.96 PMLIB::Conv::shar2_::sh**PetaMem Library: shar2_ converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from shar using shar. Filename and suffix are guessed automatically.

F.3.97 PMLIB::Conv::sit2_**PetaMem Library: sit2_ convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from sit. Filename and suffix are guessed automatically.

F.3.98 PMLIB::Conv::sit2_::unstuff**PetaMem Library: sit2_ converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from sit using unstuff. Filename and suffix are guessed automatically.

F.3.99 PMLIB::Conv::snb2txt**PetaMem Library: snb2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from snb to txt. Filename and suffix are guessed automatically.

F.3.100 PMLIB::Conv::snb2txt::calibre**PetaMem Library: snb2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from snb to txt using ebook-convert. Filename and suffix are guessed automatically.

F.3.101 PMLIB::Conv::stw2txt**PetaMem Library: stw2txt convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from stw to txt. Filename and suffix are guessed automatically.

F.3.102 PMLIB::Conv::stw2txt::unoconv**PetaMem Library: stw2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from stw to txt using unoconv. Filename and suffix are guessed automatically.

F.3.103 PMLIB::Conv::sxw2txt**PetaMem Library: sxw2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from sxw to txt. Filename and suffix are guessed automatically.

F.3.104 PMLIB::Conv::sxw2txt::unoconv**PetaMem Library: sxw2txt converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from sxw to txt using unoconv. Filename and suffix are guessed automatically.

F.3.105 PMLIB::Conv::tar2_**PetaMem Library: tar2_ convertor****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from tar using tar. Filename and suffix are guessed automatically.

F.3.106 PMLIB::Conv::tar2_::GNU**PetaMem Library: tar2_ convertor****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from tar using GNU tar. Filename and suffix are guessed automatically.

F.3.107 PMLIB::Conv::tar2_::bsdtar**PetaMem Library: tar2_ converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from tar using tar. Filename and suffix are guessed automatically.

F.3.108 PMLIB::Conv::tcr2txt**PetaMem Library: tcr2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from tcr to txt. Filename and suffix are guessed automatically.

F.3.109 PMLIB::Conv::tcr2txt::calibre**PetaMem Library: tcr2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from tcr to txt using ebook-convert. Filename and suffix are guessed automatically.

F.3.110 PMLIB::Conv::tex2pdf**PetaMem Library: tex2pdf convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from tex to pdf. Filename and suffix are guessed automatically.

F.3.111 PMLIB::Conv::tex2pdf::pdflatex**PetaMem Library: tex2pdf converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from tex to pdf using pdflatex. Filename and suffix are guessed automatically.

F.3.112 PMLIB::Conv::texi2pdf**PetaMem Library: texi2pdf convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from texi to pdf. Filename and suffix are guessed automatically.

F.3.113 PMLIB::Conv::texi2pdf::Esser**PetaMem Library: texi2pdf converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from texi to txt using texi2pdf. Filename and suffix are guessed automatically.

F.3.114 PMLIB::Conv::tiff2txt**PetaMem Library: tiff2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from tiff to txt. Filename and suffix are guessed automatically.

F.3.115 PMLIB::Conv::tiff2txt::tesseract**PetaMem Library: tiff2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from tiff to txt using Tesseract OCR. Filename and suffix are guessed automatically.

F.3.116 PMLIB::Conv::troff2txt**PetaMem Library: troff2txt convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from troff to txt. Filename and suffix are guessed automatically.

F.3.117 PMLIB::Conv::troff2txt::GNU**PetaMem Library: troff2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from troff to txt using troff. Filename and suffix are guessed automatically.

F.3.118 PMLIB::Conv::txtz2txt**PetaMem Library: txtz2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from txtz to txt. Filename and suffix are guessed automatically.

F.3.119 PMLIB::Conv::txtz2txt::calibre**PetaMem Library: txtz2txt converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from txtz to txt using ebook-convert. Filename and suffix are guessed automatically.

F.3.120 PMLIB::Conv::uot2txt**PetaMem Library: uot2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from uot to txt. Filename and suffix are guessed automatically.

F.3.121 PMLIB::Conv::uot2txt::unoconv**PetaMem Library: uot2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from uot to txt using unoconv. Filename and suffix are guessed automatically.

F.3.122 PMLIB::Conv::wpd2txt**PetaMem Library: wpd2txt convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from wpd to txt. Filename and suffix are guessed automatically.

F.3.123 PMLIB::Conv::wpd2txt::libwpd**PetaMem Library: wpd2txt converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from wpd to txt using libwpd-tools. Filename and suffix are guessed automatically.

F.3.124 PMLIB::Conv::xar2_**PetaMem Library: xar2_ convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from xar. Filename and suffix are guessed automatically.

F.3.125 PMLIB::Conv::xar2_::xar**PetaMem Library: xar2_ converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from xar using xar. Filename and suffix are guessed automatically.

F.3.126 PMLIB::Conv::xz2_**PetaMem Library: djvu2txt convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from xz. Filename and suffix are guessed automatically.

F.3.127 PMLIB::Conv::xz2_::XZ_Utils**PetaMem Library: xz2_ converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from xz using xz. Filename and suffix are guessed automatically.

F.3.128 PMLIB::Conv::zip2_**PetaMem Library: zip2_ convertors collector**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from zip. Filename and suffix are guessed automatically.

F.3.129 PMLIB::Conv::zip2_::Spieler**PetaMem Library: zip2_ converter****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from zip using unzip. Filename and suffix are guessed automatically.

F.3.130 PMLIB::Conv::zoo2_**PetaMem Library: zoo2_ convertors collector****Functions Reference****active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from zoo. Filename and suffix are guessed automatically.

F.3.131 PMLIB::Conv::zoo2_::Dhesi**PetaMem Library: zoo2_ converter**

Functions Reference**active (void)**

```
=> bool    do we have all dependencies to make this conversion?
```

work (positional)

```
1  str    file to convert
2  str    outdir (optional)
=> lref   resulting file names
```

Convert file from zoo using zoo. Filename and suffix are guessed automatically.

F.3.132 PMLIB::Email**Email tools****Functions Reference****send (named)**

```
to      str    email address
cc      str    email address
bcc     str    email address
subject str    email subject
from    str    email address
msg     str    message to send
=>     href   mail command stdout/stderr etc.
```

Wrapper for unix mail command.

F.3.133 PMLIB::Encoding**PetaMem Library: encoding-related tools****Functions Reference****convert_encoding (named)**

```
from  str    C<src> file encoding
to    str    C<dst> file encoding
src   str    source file
dst   str    destination file
=>    bool   indicate success
```

Converts file between various encodings.

create_encoding_model (named)

```
input_enc  str    encoding of C<files> (used in resulting file name)
files      lref   txt file names
model_path str    dir to store models
```

```

sizes      lref  sizes to create models from
=>         undef undef

```

Compute and store encoding models into specified directory.

guess (named)

```

file       lref  txt file to guess encoding to
model_path str  dir where to find models
sizes      lref  sizes to take into account
=>         href  values for all encodings (smaller value -> closer)

```

Returns hash of the form encoding => value. Lower value means higher probability that this encoding is what we're looking for.

F.3.134 PMLIB::Encoding::List

PetaMem Library: encoding list and its languages

Functions Reference

get_languages_for_encoding (positional)

```

1      str  encoding name
=>     lref  languages using C<1>

```

get_all_encodings (void)

```

=>     lref  all supported encodings

```

get_encodings_for_lang (positional)

```

1      str  lang iso 639-1 or 639-3 code
=>     lref  supported encodings using C<1>

```

F.3.135 PMLIB::Error

Provides functions for PMLIB error handling.

Functions Reference

pmlib_err (positional)

```

1      enum  type (default: noargs)
2      list  params
=>     -      croaks for fatal
=>     str    error message for no fatal

```

Fatal could be set with variable `$PMLIB::Error::PMLIB_ERR_FATAL`.

F.3.136 PMLIB::Facade

PetaMem Library: PMLIB specific tools Facade for creating wrappers easily.

Functions Reference

facade (positional)

```

1 href  params for dispatched functions
2 href  dispatch functions
=> any  return value of function

```

2 is of the form name => [module_name, func_name].

1 is of the form

```

function => 'function_name'
args     => {args for function}
ref      => dereference args with @?
croak    => 'croak message' if you wanna croak

```

F.3.137 PMLIB::Hash

PetaMem Library: Implementation of a rich Hash Object This class serves as workhorse (base class) for many requirements that recur often in PetaMem projects. While native Perl hashes are a versatile data structure, we were missing things like regex-based match of keys, meta information about the hash (has data changed) etc.

The structure itself is used at many places, such as config files, feature registry, lexica etc.

SYNOPSIS

```

my $obj1 = __PACKAGE__->new();
my $obj2 = __PACKAGE__->new({k1 => v1})
my $obj3 = __PACKAGE__->new(data => {k1 => v1})
my $obj4 = __PACKAGE__->new(__PACKAGE__)

$obj1->get()
$obj2->get('k1')
$obj3->get(qr{\Ak.\z})
$obj4->get([qw(k1 k2 k3)])

```

Methods Reference

BUILDARGS

new (positional)

```

1 any hash
=> obj new PMLIB::Hash object

```

Create a new PMLIB::Hash object. There are several ways to do it:

```

undef      creates an object with an empty data section
hashref    the data section is set to this value
list/hash  will set the data section and may also set the meta info
__PACKAGE__ will clone the given object

```

deserialize (named)

format	enum	format to serialize into
data	str	serialized PMLIB::Hash object
=>	str	constructed PMLIB::Hash object

get (positional)

1	any	polymorphic argument
=>	lref href -	values or croaks

Retrieve values from hash. Accepts different types of arguments:

undef	returns whole hashref
list	returns list of value(s) of the key(s) that are given in list
listref	returns listref of all values of keys in listref
regex	returns listref of all values for keys that match regex
code	returns listref of keys that satisfy given coderef

This method returns a list of the corresponding value if it has been given a list of arguments and it returns a listref containing the values if it has been given a listref, regular expression or coderef.

If the polymorphic argument (1) was given, but it is not any from above mentioned types, the process will end with an error via Carp::croak.

serialize (named)

format	enum	format to serialize into
args	href	args for serialization if necessary
=>	str	serialized PMLIB::Hash object

set (positional)

1	any	polymorphic argument
=>	lref undef	old values

The polymorphic argument may be one of following:

scalar	transform scalar values into a hash
list	interprets list as list of K/V pairs and sets these in hash
listref	transform listref to a hash
hashref	modify the data in a hash

Giving a list arguments to this method implies that this list has to have an even number of elements. If the number of list elements is odd, method will return an undef value.

This method returns a listref of old values.

statistics (void)

=>	any	execute command
----	-----	-----------------

This method gets a number of elements in a hash.

clone (void)

=>	obj	object
----	-----	--------

This method returns a clone of input object. The clone is duplicated via `Clone::clone()`.

del (positional)

```
1  scalar|list|rx  reference, which content we want to delete
=> lref           values of the deleted elements
```

The reference can refer to:

```
scalar  one scalar is deleted
array   hash slice is deleted
regex   keys will be sorted and filtered according to given regexp,
        then a hash slice would be deleted
```

exist (positional)

```
1  str  string is representing the key
=> bool indicate success
```

This method uses `exists` builtin for the key `1` in a hash object and returns the resulting Boolean value.

get_allkeys (void)

```
=> lref  list of keys
```

Takes no parameters and returns a list to all keys in a hash.

get_allkeys (void)

```
=> lref  sorted list of keys
```

Like `get_allkeys` also takes no parameters but returns a sorted list to all keys in a hash.

get_allvalues (void)

```
=> lref  sorted list of values
```

Takes no parameters and returns sorted listref of values in a hash.

get_allvalues (void)

```
=> lref  list of values
```

Takes no parameters and returns a listref of all values in a hash.

get_init_kchars (positional)

```
1  str  define the return type; default: 'hr'
=> lref  if C<1> ne 'hr'
=> href  else
```

Initial characters of keys in the hash are returned in a hash reference or in sorted list reference. (Depends on the input parameter.)

get_keys (positional)

```
1  any  polymorphic value
=> lref  if ok
=> -    croak if input parameter is unknown type
```

The input parameter may be a reference on:

```
scalar
array
hash
regexp
code
```

Keys of a hash are filtered according to the specified content of the reference. If no input argument is given, listref of all keys is returned. If the input argument is given, but its value is not one above (scalar, ...), the process will end with an error via `Carp::croak()`;

get_numkeys (positional)

```
1    rx    pattern for keys
=>  int    number of matching entries
```

Returns number of entries in a hash. If regexp given, only keys that match given regexp are counted in the returned value.

get_statistics (void)

```
=>  href  stats
```

This method returns a hashref with info about given object.

get_values (positional)

```
1    rx    regex
=>  lref   filtered values
```

Returns a listref of values filtered according to the regexp reference. If no regexp given, all values are returned.

pp (positional)

```
1    href  a hash reference with parameters for print
=>  str    string with given content
```

This method calls `PMLIB::Serialize::pp_struct()`. The input parameter is hashref and serves as a named input for `pp_struct` method. It must contain `struct` key with given data structure to print.

set_if_notexist (positional)

```
1    str    represents a key of a hash
2    str    represents a value of a hash
=>  lref   set
=>  undef  if C<1> already exists
```

This method sets a key/value pair in a hash, if the key already doesn't exist in the hash.

_fr_keys (positional)

```
1    cref  filter matching keys
=>  lref   matching keys
```

Will return listref of filtered keys. If no coderef given, listref of all keys will be returned.

`_rx_keys` (positional)

```
1   rx      positive-match of the keys in a hash
=>  lref    matching keys
```

Will return a listref of keys matching given regexp. If no regexp given, returns sorted listref with all keys.

`_rx_keyval_hr` (positional)

```
1   rx      regexp (positive match)
=>  href    matching keys
```

This method returns a hashref - only keys that MATCH given regexp, will pass through the filter. If no regexp given, the original data structure is returned.

`_rx_skeys` (positional)

```
1   rx      positive-match of the keys in a hash
=>  lref    matching keys
```

Like `_rx_keys`, but will return a listref to a sorted list of keys matching given regexp. If no regexp given, returns sorted listref of all keys in a hash.

F.3.138 PMLIB::Hash::Tied

PetaMem Library: Implementation of a tied hash Implementation of a tied hash.

Methods Reference

`pm_tie` (positional)

```
1   href    hash to tie
2   str     file
=>  href    tied C<1>
```

`pm_untie` (positional)

```
1   href    tied hash
=>  href    untied hash
```

F.3.139 PMLIB::Help

PetaMem Library: PetaMem Help

Functions Reference

`all_help` (positional)

```
1   str     command to get help for
2   str     iso localization for the help (default: eng)
=>  str     generated help text from all sections (formatted)
```

call `gen_help` for all sections present

gen_help (positional)

```
1  str  section
2  str  command to get help for
3  str  iso localization for the help (default: eng)
=> str  generated help text (formatted)
```

get_use (positional)

```
1  str  command to get _USE information for
=> str  text containing CLI usage information for given command,
=> undef if not found
```

get _USE information for a given command

help_from (positional)

```
1  str  section
2  str  command to get help for
3  str  iso localization for the help (default: eng)
=> str  raw help text
```

help_syntax (positional)

```
1  str  iso localization for the help (default: eng)
=> str  help syntax text
```

read_help (positional)

```
1  str  directory with ini files
=> str  read help sections
```

Read in all help texts to module-global structure.

F.3.140 PMLIB::ISO

This is a parent class for ISO modules. It provides generic operations over data-hashes.

Methods Reference

get_all_codes (void)

```
=> lref  a list of all ISO codes
```

List all codes within given ISO namespace

get_all_names_4l10n (named)

```
l10n str  ISO 639-3 code
=> href  hash ref with l10n
```

This method iterates all codes in given ISO name space and returns a name for given ISO code in specified l10n. If wrong or no l10n was specified, returns l10n for English.

get_l10n_for (named)

```

iso_code  str    given ISO code
l10n      str    target language - 'eng' by default
=>        str    name in target language
=>        undef  if wrong or no iso_code was given

```

Example: ISO code for English language is - within ISO 639-3 namespace -'eng'. If we want the name of 'English' in German language:

```

iso_code => 'eng',
l10n     => 'deu',

```

If the specified code is not available (or it is identical with english localization), a localization for English is returned instead.

supported_l10n_4code (named)

```

iso_code  str    given ISO code
l10n      str    target language
=>        bool   name in target language
=>        href   hash ref. of available l10ns
=>        undef  if wrong or no iso_code was given

```

If only iso code was specified, this method will return a hash reference with all available languages for given iso code.

If an iso code AND target language were specified, a boolean value is returned in order to indicate if given target language is available for the iso code.

Some localizations may be identical with English and therefore are not mentioned in source data. Please, check l10n for English!

valid_iso_code (named)

```

iso_code  str    name of the output file
=>        bool   a boolean if given code is valid

```

Check if code is valid within given ISO namespace.

F.3.141 PMLIB::ISO::11940

PetaMem Library: ISO 11940 functionality ISO standard for the romanization of the Thai alphabet

Functions Reference

F.3.142 PMLIB::ISO::11941

PetaMem Library: ISO 11941 functionality Korean romanization system used in ISO

Functions Reference

F.3.143 PMLIB::ISO::15919

PetaMem Library: ISO 15919 functionality Transliteration of Devanagari and related Indic scripts into Latin characters

Functions Reference**F.3.144 PMLIB::ISO::15924**

PetaMem Library: ISO 15924 functionality Codes for the representation of names of scripts

Methods Reference**BUILD (void)**

```
=> - private
```

Private Moose method.

digit2alpha (named)

```
digit int    15924 numeric code
=>     str    15924 alpha code
=>     undef  croaks / ret. undef if there is no ref for <1>
```

ISO 15914 has also numeric codes. This method allows you to convert numeric code to alphabetic.

get_all_codes (void)

```
=> lref  a list of all 4217 ISO codes
```

List all codes within given ISO namespace

get_l10n_for (named)

```
iso_code str    given ISO-15924 code
l10n     str    target language - 'eng' by default
=>       str    name in target language
=>       undef  if wrong or no iso_code was given
```

Example: ISO code for 'Tibetan' is - within ISO-15924 namespace - 'tib'. If we want the name of 'Tibetan' in German language:

```
iso_code => 'tib',
l10n     => 'deu',
```

should return something like 'Tibetische Schrift'.

If the specified code is not available (or it is identical with english localization), a localization for English is returned instead.

supported_l10n_4code (named)

```

iso_code  str    given ISO-15924 code
l10n      str    target language
=>        bool   name in target language
=>        href   hash ref. of available l10ns
=>        undef  if wrong or no iso_code was given

```

If only iso code was specified, this method will return a hash reference with all available languages for given iso code.

If an iso code AND target language were specified, a boolean value is returned in order to indicate if given target language is available for the iso code.

Some localizations may be identical with English and therefore are not mentioned in source data. Please, check l10n for English!

valid_iso_code (named)

```

iso_code  str    name of the output file
=>        bool   a boolean if given code is valid

```

Check if code is valid within given ISO-15924 namespace.

F.3.145 PMLIB::ISO::233

PetaMem Library: ISO 233 functionality System for Arabic transliteration (Romanization).

Functions Reference

F.3.146 PMLIB::ISO::259

PetaMem Library: ISO 259 functionality Series of international standards for the romanization of Hebrew.

Functions Reference

F.3.147 PMLIB::ISO::3166

PetaMem Library: ISO 3166 functionality Codes for the representation of names of countries and their subdivisions.

Methods Reference

BUILD (void)

```
=> - private
```

Private Moose method.

get_all_codes (void)

```
=> lref   a list of all 3166-1 alpha2 ISO codes
```

List all codes within given ISO namespace

get_info_about (named)

```
iso_code    str    3166-1 alpha2 code
wanted_info str    specification of category
=>          xref   returns different data-structs
=>          undef  croaks if input args are badly specified
```

This method returns information for given category. Categories return different data-structs, because content of categories is specific.

Currently supported categories (and their data-structs) are:

category	data str.	note
TLD	=> 'str',	top-level domain
alpha3	=> 'str',	3166-1 alpha3 code
stanag	=> 'str',	STANAG code
area_sq_km	=> {},	area in sq km
borders_km	=> {},	borders in km
currency	=> 'str',	currency
d3166	=> 'str',	3166-1 numeric code
gec	=> 'str',	GEC - previously FIPS PUB 10-4 code
l10n	=> {},	l10n
neighbors	=> [],	neighbors of given country
rel_languages	=> {},	distribution of langs

To get info about specific category, call:

```
$ISO_object->get_info_about({
    iso_code    => 'zw',
    wanted_info => 'currency',
});
```

Value of scalar data structs may be 'N/A' if info is not available. Other categories may be empty - {} or [] if there is no info.

get_l10n_for (named)

```
iso_code str    given ISO-3166-1 alpha2 code
l10n     str    target language - 'eng' by default
=>       str    name in target language
=>       undef  if wrong or no iso_code was given
```

Example: ISO code for 'Zimbabwe' is - 'zw'. If we want the name of 'Zimbabwe' in German language:

```
iso_code => 'zw',
l10n     => 'deu',
```

should return something like 'Simbabwe'.

If the specified code is not available (or it is identical with english localization), a localization for English is returned instead.

supported_l10n_4code (named)

```

iso_code  str    given ISO-3166-1 alpha2 code
l10n      str    target language
=>        bool   name in target language
=>        href   hash ref. of available l10ns
=>        undef  if wrong or no iso_code was given

```

If only iso code was specified, this method will return a hash reference with all available languages for given iso code.

If an iso code AND target language were specified, a boolean value is returned in order to indicate if given target language is available for the iso code.

Some localizations may be identical with English and therefore are not mentioned in source data. Please, check l10n for English!

valid_iso_code (named)

```

iso_code  str    name of the output file
=>        bool   a boolean if given code is valid

```

Check if code is valid within given ISO-3166-1 alpha2 namespace.

F.3.148 PMLIB::ISO::3602

PetaMem Library: ISO 3602 functionality Japanese romanization system Kunrei-shiki rōmaji

Functions Reference**F.3.149 PMLIB::ISO::4217**

PetaMem Library: ISO 4217 functionality ISO code for currencies.

Methods Reference**BUILD (void)**

```

=> - private

```

Private Moose method.

get_all_codes (void)

```

=> lref   a list of all 4217 ISO codes

```

List all codes within given ISO namespace

get_l10n_for (named)

```

iso_code  str    given ISO-4217 code
l10n      str    target language - 'eng' by default
=>        str    name in target language
=>        undef  if wrong or no iso_code was given

```

Example: ISO code for 'Zimbabwean dollar' is - within ISO-4217 namespace - 'zwr'. If we want the name of 'Zimbabwean dollar' in German language:

```
iso_code => 'zwr',
l10n     => 'deu',
```

should return something like 'Simbabwe-Dollar'.

If the specified code is not available (or it is identical with english localization), a localization for English is returned instead.

supported_l10n_4code (named)

```
iso_code  str    given ISO-4217 code
l10n      str    target language
=>        bool   name in target language
=>        href   hash ref. of available l10ns
=>        undef  if wrong or no iso_code was given
```

If only iso code was specified, this method will return a hash reference with all available languages for given iso code.

If an iso code AND target language were specified, a boolean value is returned in order to indicate if given target language is available for the iso code.

Some localizations may be identical with English and therefore are not mentioned in source data. Please, check l10n for English!

valid_iso_code (named)

```
iso_code  str    name of the output file
=>        bool   a boolean if given code is valid
```

Check if code is valid within given ISO-4217 namespace.

F.3.150 PMLIB::ISO::639

PetaMem Library: ISO 639 functionality This module offers a comprehensive set of ISO 639-related functionality. ISO 639 is the set of international standards that lists short codes of two to four letters for language names. We mainly cover ISO 639-1 and ISO 639-3 functionality.

Functions Reference

_1to3_639 (positional)

```
1      str    iso639-1 code
=>     str    iso639-3 code
=>     undef  if invalid code given
```

Convert iso 639-1 to iso 639-3 representation.

_3to1_639 (positional)

```

1   str    iso639-3 code
=>  str    iso639-1 code
=>  undef  if invalid code given or no representation exists

```

Convert iso 639-3 to iso 639-1 representation if possible. Return undef if given iso 639-3 code is invalid or has no representation in iso 639-1.

`_1or3_639` (positional)

```

1   str    iso639-3 or iso639-1 code
=>  str    iso639-3 code
=>  undef  if invalid code given or no representation exists

```

Convert iso 639-1 or iso 639-3 to iso 639-3 representation. Return undef if given input iso 639 code is invalid. This function serves as robust interface to get iso 639-3 representation when the input may be in either iso 639-1 or iso 639-3 format.

`get_iso1_639` (positional)

```

1   rx     positive filter
=>  lref   list of iso 639-1 codes

```

If no parameter is given, returns a sorted list of all known/supported iso639-1 codes. If given, the 1st parameter is interpreted as regular expression and applied as filter to each element in the list of iso 639-1 codes. Those that match this regular expression are returned in a sorted listref. ATTENTION: do not use this to find out whether a given iso code is valid. Instead, use `is_validiso_639($code)` for that. See also `get_iso3_639`.

`get_iso3_639` (positional)

```

1   rx     positive filter
=>  lref   list of iso 639-3 codes

```

Returns a sorted list of all known/supported iso639-3 codes. See also `get_iso1_639`.

`get_iso3_from_name_639` (positional)

```

1   str    name of language
2   str    language code of this name (default: eng)
=>  str    code for language from C<1>

```

`get_iso3_from_any_name_639` (positional)

```

1   str    name of language
=>  str    code for language from C<1>

```

`get_name_639` (positional)

```

1   str    language code
2   str    language code (default: eng)
=>  str    name of language with code C<1> in language C<2>
=>  -      croaks if wrong C<1>

```

`validate_lr_iso639` (positional)

```

1   lref   iso codes
=>  lref   valid ones

```

Filter invalid codes from array of codes.

getall_names_iso639 (void)

```
=> href natural language name => iso code
```

NYI. Returns a reference to a hash where the keys are natural language names of languages and values are the corresponding iso-codes a fallback to english is provided for nonexisting iso-codes

supported_iso639 (positional)

```
1 str iso639 code (optional)
=> bool exists?
=> lref existing codes
```

returns reference to a hash of iso639 codes for whose you can expect to gain localized names of languages if an argument is given, it is interpreted as iso code and TRUE/FALSE is returned whether that iso code is an existent or nonexistent localization

is_validiso_639 (positional)

```
1 str iso639-1 or iso639-3 code
=> bool is valid?
```

Given argument can be in upper- or lowercased iso 639-1 or 639-3 format.

get_names_639 (named)

```
loc str iso code of the desired localization (def: 'eng')
langs lref list of isos to localize, default: ALL
list bool flag whether to return listref, default: 0 (=NO)
=> href|lref hash of the form iso => localized language
```

If the 'list' parameter has true value, just a listref of the values (= localized language names) is returned;

getiso639 (positional)

```
1 str language name
=> str iso code for language whose name given
```

NIY. Given a language name (in plain text) and an optional iso639 code for the expected localization, this returns the iso639 code for the language whose name was given. If no iso639 information is found, undef is returned.

F.3.151 PMLIB::ISO::7098

PetaMem Library: ISO 7098 functionality Pinyin - official system to transcribe Chinese characters into Latin script.

Functions Reference

F.3.152 PMLIB::ISO::843

PetaMem Library: ISO 843 functionality System for the transliteration of Greek characters into Latin characters

Functions Reference**F.3.153 PMLIB::ISO::9**

PetaMem Library: ISO 9 functionality System for the transliteration of Cyrillic characters into Latin characters

Functions Reference**F.3.154 PMLIB::ISO::9984**

PetaMem Library: ISO 9984 functionality System for the transliteration of modern Georgian characters into Latin characters

Functions Reference**F.3.155 PMLIB::ISO::9985**

PetaMem Library: ISO 9985 functionality Standard for transliteration of the modern Armenian alphabet

Functions Reference**F.3.156 PMLIB::IdentificationModeling**

PetaMem Library: PMLIB specific tools Learning models and data fitting.

Functions Reference**learn_model (named)**

files	lref	txt files to make model from
store	str	model name
max	int	model size
enc	str	input encoding (read raw data by default)
=>	href	model

Store and returns model. Model are ngrams.

fit (named)

file	str	txt file
models	lref	list of models to compare C<file> to
max	int	model size
enc	str	input encoding (read raw data by default)
=>	href	distances to models

F.3.157 PMLIB::IdentificationModeling::Ngram

PetaMem Library: PMLIB specific tools

Functions Reference

learn_model_ngram (named)

files	lref	files to create model from
store	str	where to store model
max	int	model size to be stored
enc	str	files encoding to consider (default: raw)
allow_smaller	bool	allow smaller models than C<max>?
letter_regex	str	regex that all letters in model must pass
=>	href	model

fit_ngram (named)

models	lref	models to compare file to
file	str	file we wanna fit to somewhere
distance	str	distance function (default: euclid)
enc	str	encoding of file (default: raw)
max	int	max number of ngrams to store
maxlen	int	allowed length of chars
=>	href	distances to models

Maxlen: if defined, only maxlen chars will be read from file.

F.3.158 PMLIB::IdentificationModeling::Nvect

PetaMem Library: PMLIB specific tools

Functions Reference

learn_model_nvect (named)

files	lref	files to create model from
store	str	where to store model
enc	str	files encoding to consider (default: raw)
letter_regex	str	regex that all letters in model must pass
nvect_delimiter	str	token delimiter
=>	href	model

Currently supports character probability only.

fit_nvect (named)

```

models  lref  models to compare file to
file    str   file we wanna fit to somewhere
distance str  distance function (default: euclid)
enc     str   encoding of file (default: raw)
maxlen  int   allowed length of chars
delimiter str token delimiter
=>      href  distances to models

```

Maxlen: if defined, only maxlen chars will be read from file.

F.3.159 PMLIB::IdentificationModeling::Tfreq**PetaMem Library: PMLIB specific tools****Functions Reference****learn_model_tfreq (named)**

```

files      lref  files to create model from
store     str   where to store model
max       int   model size to be stored
letter_regex str  regex that all letters in model must pass
=>        href  model

```

fit_tfreq (named)

```

models  lref  models to compare file to
file    str   file we wanna fit to somewhere
=>      href  distances to models

```

F.3.160 PMLIB::List**PetaMem Library list/array tools****Methods Reference****columns_element (positional)**

```

1  lref  list
2  int   number of columns
3  int   width
=>  str   formatted output

```

Returns formatted 1 to string.

columns_list (named)

```

lol  lref  list of lists
spc  int   number of spaces between columns
fmt  str   manual format string (default: no)
=>  str   formatted list

```

Format lists to columns as string.

F.3.161 PMLIB::Logic::Belief

PetaMem Library: logical operations for "Belief Logic". This is the generalization of 3-valued logic Where we take especially K3 (Kleene's 3-valued Logic) into consideration. Part of this module contains wrappers to PMLIB::Stochastic::P.

Functions Reference

lb_and (positional)

```
1 list belief values
=> list and of belief values
```

Wrapper for and_p

lb_or (positional)

```
1 list belief values
=> list or of belief values
```

Wrapper for or_p

lb_strong_not (positional)

```
1 list belief values
=> list strong negations of belief values
```

For every belief value in given list, compute the strong negation to this belief.

lb_strong_nand (positional)

```
1 list belief values
=> belief belief value of (strong) NAND-conjunction of belief values
```

See lb_and, then strong-negate that resulting value. I.e. return the complement of minimum belief.

lb_strong_nor (positional)

```
1 list belief values
=> belief belief value of (strong) NOR-conjunction of belief values
```

See lb_or, then strong-negate that resulting value. I.e. return the complement of maximum belief.

lb_xor (positional)

```
1 list belief values
=> belief belief xor
```

NYI

lb_strong_nxor (positional)

```
1 list belief values
=> belief belief nxor
```

See `lb_xor`, then strong-negate that resulting value.

lb_imp (positional)

```
1 belief belief value of "if"
2 belief belief value of "then"
=> belief resulting belief value of subjunction
```

Compute resulting belief value of IF belief THEN belief. Subjunction or material implication.

lb_strong_nimp (positional)

```
1 belief belief value of "if"
2 belief belief value of "then"
=> belief resulting belief value of (strong) NIMP-conjunction
```

See `lb_imp`, then strong negate that belief.

lb_ded (positional)

```
1 belief belief value of "then"
2 belief belief value of "if"
=> belief resulting belief
```

Converse material implication.

lb_strong_nded (positional)

```
1 belief belief value of "not then"
2 belief belief value of "if"
=> belief resulting belief
```

See `lb_ded`, then strong negate that belief.

lb_weak_not (positional)

```
1 list belief values
=> list weak negations of belief values
```

Everything that is solid true becomes false, everyting else becomes true.

lb_weak_yes (positional)

```
1 list belief values
=> list weak yes of belief values
```

Everything that is solid false becomes true, everyting else becomes false.

lb_weak_not (positional)

```
1 list belief values
=> list weak affirmation of belief values
```

Everything that is solid false becomes true, everything else becomes false.

lb_avg (positional)

```
1 list belief values
=> belief the average belief value out of given list
```

This will return a value that can be interpreted as resulting certainty of belief values of (the same) input data. If we have the information x and $\text{NOT}(x)$, the resulting certainty is 0.5; if these values were both 0.5, the resulting certainty is also 0.5

lb_strong_navg (positional)

```
1 list belief values
=> belief the complementary average belief value out of given list
```

See `lb_avg`. Something as "the certainty to believe the opposite".

lb_min (positional)

```
1 list belief values
=> belief the minimum belief value out of given list
```

lb_max (positional)

```
1 list belief values
=> belief the maximum belief value out of given list
```

F.3.162 PMLIB::Loop**PetaMem Library loops****Functions Reference****for_slice_parallel (named)**

```
list lref list to loop over
sub cref body of the loop
cpus int how many threads to use
=> undef undef
```

Analogy of standard perl for loop with threads.

while_slice_parallel (named)

```
list cref code to generate next data
sub cref sub to process over each slice
cpus int number of threads to use
limit int how many operations in one thread ()
start any argument for first run of C<list> coderef
=> undef undef
```

Each thread runs `list` coderef `limit` times and put generated data to `sub`. `sub` is the body of the loop.

F.3.163 PMLIB::Math::Combinatorics**Perform combinations on lists****Functions Reference****combinations (positional)**

```

1   int   n
2   int   k
=>  int   nCk

```

combine (positional)

```

1   lref   list to combine
2   int    size of groups
=>  lref   list of all possible combinations

```

Returns k-subsets of 1.

factorial (positional)

```

1   int   k
=>  int   k!

```

next_combination (positional)

```

1   lref   current combination
2   int    size of groups
3   int    number of elements
=>  lref   next combination
=>  undef  no more combinations defined

```

Returns combination after given one.

permute (positional)

```

1   list   list
=>  lref   list of all permutations of C<1>

```

Returns listref of all permutations.

subsets (positional)

```

1   lref   set
=>  lref   list of all subsets of C<1>

```

Returns all subsets.

subsets_pos (named)

```

n     int    specify set (0 .. n-1)
char  int    specify empty element
=>   lref   list of all subsets of (0 .. n-1)

```

F.3.164 PMLIB::Math::Combinatorics::Bell**PetaMem Library: Combinatorics** Bell

Bell numbers theory.

Functions Reference**get_bell_number (positional)**

```

1   int    n
=>  int    bell(n)

```

get_bell_partitions (positional)

```

1   int    n
=>  lref   list of bell partitions

```

F.3.165 PMLIB::Math::Interval**PetaMem Library: Math Section** Interval Arithmetics**Functions Reference****within (positional)**

```

1   int    number
2   lref   [int,int] as interval [from, to]
=>  int    C<1> if number in interval
=>  int    C<from> if number below it
=>  int    C<to> if number above it

```

Ensure given number is within a given interval.

is_within (positional)

```

1   int    number
2   lref   [int,int] as interval [from, to]
=>  bool   C<1> in C<2>?

```

interval_length (positional)

```

1   any    interval
=>  int    interval size

```

Converts anything to interval with `interval_x2p_ify` and compute its length.**getrel_intervals (positional)**

```

1   str|pair interval 1 either in string or pair format
2   str|pair interval 2 either in string or pair format
=>  str     type of relationship (2-char string) - see below

```

Determine the relation between two intervals. The relation type is encoded in a 2-character string, where the characters can be the following (B = before, S = start, I =

in, E = end, A = after). First character is the starting point of interval 1, 2nd character is the end point of interval 1. Both RELATIVE TO interval 2. There are $5 + 4 + 3 + 2 + 1 = 15$ possible relations, namely:

```

BB  i1 starts and ends before i2
BS  i1 starts before and ends at start of i2
BI  i1 starts before and ends within i2
BE  i1 starts before and ends together with i2
BA  i1 starts before and ends after i2
SS  i1 starts and ends when i2 starts
SI  i1 starts with i2 and ends within i2
SE  i1 starts and ends with i2
SA  i1 starts with i2 and ends after i2
II  i1 starts and ends within i2
IE  i1 starts within i2 and ends together with i2
IA  i1 starts within i2 and ends after i2
EE  i1 starts and ends when i2 ends
EA  i1 starts when i2 ends and ends after i2
AA  i1 starts and ends after i2

```

getrel_inverse (positional)

```

1  enum    relation (see getrel_intervals)
=> enum    inverse relation

```

are_numbers (positional)

```

1  lref    some values
=> bool    are they numbers?

```

is_computable (positional)

```

1  str     value
=> bool    is number or interval?

```

is_correct (positional)

```

1  str     value
=> bool    is correct interval?

```

interval_x2s_ify (positional)

```

1  str     interval in string representation
2  lref    interval in pair representation
=> str     interval in string representation

```

Convert string/pair to string representation. If string was given already, return that unchanged. Inverse of the interval_x2p_ify function.

interval_x2p_ify (positional)

```

1  str     interval in string representation
2  lref    interval in pair representation
=> lref    interval in pair representation

```

Convert pair/string to interval pair (listref). If listref was given already, return that unchanged. Inverse of the `interval_x2s_ify` function.

add_intervals (positional)

```
1 interval interval 1
2 interval interval 2
=> interval sum of C<1> and C<2>
```

Example:

$$[a,b] + [c,d] = [a + c, b + d].$$

sub_intervals (positional)

```
1 interval interval 1
2 interval interval 2
=> interval difference of C<1> and C<2>
```

Example:

$$[a,b] - [c,d] = [a - d, b - c]$$

mul_intervals (positional)

```
1 interval interval 1
2 interval interval 2
=> interval multiplication of C<1> and C<2>
```

Example:

$$[a,b] \times [c,d] = [\min (ac, ad, bc, bd), \max (ac, ad, bc, bd)]$$

div_intervals (positional)

```
1 interval interval 1
2 interval interval 2
=> interval div of C<1> and C<2>
```

Example:

$$[a,b] / [c,d] = [\min (a/c, a/d, b/c, b/d), \max (a/c, a/d, b/c, b/d)]$$

sqr_interval (positional)

```
1 interval interval
=> interval C<1>^2
```

F.3.166 PMLIB::Math::Iterator

PMLIB::Math::Iterator Looping constructs: NestedLoops

F.3.167 PMLIB::Math::Matrix

Object-oriented matrix operations.

Functions Reference

new (constructor)

```
matrix => arrayref    matrix data (array of arrays)
=>      Matrix      Matrix object
```

Creates a Matrix object with the provided matrix data.

rows (method)

Returns the number of rows in the matrix object.

```
=>  scalar    number of rows
```

cols (method)

Returns the number of columns in the matrix object.

```
=>  scalar    number of columns
```

product (method)

Computes product of this matrix with another matrix object.

```
1  Matrix    another Matrix object
=> Matrix    product of this matrix and C<1>
```

difference (method)

Computes difference of this matrix with another matrix object.

```
1  Matrix    another Matrix object
=> Matrix    difference of this matrix and C<1>
```

transpose (method)

Computes transposition of this matrix.

```
=>  Matrix    transposed matrix
```

sqrt (method)

Computes Cholesky square root of this positive-semidefinite matrix.

```
=>  Matrix    square root matrix
```

determinant (method)

Computes determinant of this matrix.

```
=>  scalar    determinant value
```

inverse (method)

Computes inverse of this matrix.

```
=>  Matrix    inverse matrix (or undef if not invertible)
```

matrix (attribute)

The matrix data structure (array of arrays).

```
=> arrayref the matrix data
```

F.3.168 PMLIB::Math::SetTheory

PetaMem Library: Provide elementary set operations. The purpose of this module is to provide a comprehensive implementation of elementary set operations needed in higher level algorithms.

Functions Reference

get_set_cartesian (positional)

```
1 list list of lists
=> list cartesian product of C<1>
```

Build Cartesian product over n sets. These sets are given as a list of listrefs to the function. Return value is a list of all combinations.

get_set_difference (positional)

```
1 any set A that is subtracted from (list or hashref)
2 any set B that is subtracted
=> lref set difference
```

Compute the set-theoretic difference between two sets A and B. Also called the relative complement of A in B.

get_set_intersection (named)

```
sets lref list of lists
trust bool sets have unique elements
type enum return list (default) or hash
=> href|lref isect
```

or

```
1 list list of xrefs
=> lref isect
```

Compute the intersection (\cap) of N sets. Basically a wrapper to `get_set_intersection_union`, where it returns the 1st element of its return value, so for arguments see this base function.

get_set_intersection_union (named)

```
sets lref list of lists
trust bool sets have unique elements
type enum return list (default) or hash
=> lref [href,href]|[lref,lref] as [isect, union]
```

or

```
1 list list of xrefs
=> lref [lref,lref] as [isect, union]
```

Compute the intersection (\cap) and the union (\cup) of N sets.

Takes N sets (either listref or hashref, may be mixed too) and returns a 2-element listref, with the intersection set (hashref) as the 1st element and the union set (hashref) as the 2nd element.

If we trust the elements in the given sets are already unique (because listrefs could contain dupes), we indicate this with the `trust=> 1` flag (which is else 0 by default). This can give us a little performance gain.

get_set_named_symmetric_difference (positional)

```
1  lref    list 1
2  lref    list 2
=> href    hash of the form {to_add => lref, to_del => lref}
```

'Named symmetric difference': Given two lists, determine what should be done to the first list, to contain the same elements as the second one, where order of the elements is not an issue. Result is a hash reference with 2 keys

```
to_add => listref of elements to be added
to_del => listref of elements to be deleted
```

get_set_relation (positional)

```
1  any     set 1
2  any     set 2
=> enum    relation between set 1 and set 2
```

Determines relation between set A & B.

Possible return values

```
A_IS_B - intersection
A_EQ_B - equal
A_DI_B - disjoint
A_SS_B - superset
B_SS_A - superset
```

get_set_symmetric_difference (named)

```
sets    lref    list of lists
trust   bool    sets have unique elements
=>      href    symmetric difference
```

Compute the symmetric difference of N sets. This calls `get_set_union` and removes all elements of union that have a count of > 1 .

get_set_union (named)

```
sets    lref    list of lists
trust   bool    sets have unique elements
type    enum    return list (default) or hash
=>      href|lref union
```

or

```

1      list  list of xrefs
=>    lref  union

```

Compute the union (\cup) of N sets. Basically a wrapper to `get_set_intersection_union`, where it returns the 2nd element of its return value, so for arguments see this base function.

get_list_intersection_sorted (named)

```

lists  lref  lists of lists
=>    lref  intersection

```

Eric Sadit algorithm for intersection of sorted lists. Faster than `get_set_intersection_union` when lists are sorted.

F.3.169 PMLIB::Math::Vector

Functions Reference

vector_abs (positional)

```

1  lref  vector
=> lref  absolute value of C<1>

```

In fact absolute value of the n-dimensional vector is distance from 0 in n-dimensional space.

vector_minus (positional)

```

1  lref  vector
=> lref  negation of C<1>

```

vector_sum (positional)

```

1  list  vectors
=> lref  sum of C<1>

```

vector_product (positional)

```

1  lref  vector
2  lref  vector
=> lref  vector by-component product

```

F.3.170 PMLIB::Metric

Wrapper for various distance functions.

Functions Reference

distance_generic (named)

```

function  str  name of distance function
args      href  parameters for chosen distance function
=>        real  distance

```

This function is a switch for different distance functions.

function is usually string corresponding some distance function for strings without prefix "distance_". Here is a list of possible distances:

```

block      canberra  chebyshev  correlation
cosine     czekonowski damerau_levenshtein
dice       discrete  euclid     hamming
jaccard    jaro       jaro_winkler kulezinski
levenshtein mahalanobis manhattan  minkowski
needleman_wunsch          overlap    renkonen
russel_rao tanimoto   smith_waterman
sokal_sneath          tversky yule

```

Some aliases are also supported:

```
bray_curtis  chess_board  city_block  edit  sorensen
```

args depends on chosen distance function. Usually there are following compulsory parameters for comparing as lists:

```

l1  lref  first list
l2  lref  second list

```

or following ones for comparing as histograms:

```

s1  href  first histogram
s2  href  second histogram

```

For specific parameters inspect documentation of individual distance function. You can find it below (for distance functions derived from its numeric version) or in PMLIB::Metric namespace (for those in basic form)

list_metrics (void)

```
=>  lref  list of metrics
```

Returns list of available metrics.

F.3.171 PMLIB::Metric::Block

Functions Reference

distance_block (named)

```

l1|s1      lref|href  point 1
l2|s2      lref|href  point 2
=>         real[0>]  distance

```

F.3.172 PMLIB::Metric::Canberra

Functions Reference

distance_canberra (named)

```

l1      lref  first list
l2      lref  second list
s1      href  first histogram
s2      href  second histogram
=>     real  distance

```

Either `l1`, `l2` (lists) or `s1`, `s2` (histograms) must be given. Returns Canberra distance between histograms (bigger priority) or between lists.

For specifics about Canberra distance see section about `P_dmp` in PMSE manual.

F.3.173 PMLIB::Metric::Chebyshev

Functions Reference

distance_chebyshev (named)

```

l1|s1      lref|href  point 1
l2|s2      lref|href  point 2
normalized bool       should we project result to [0,1] interval?
type       enum       'num' or 'str'
=>        real[0>]   distance

```

F.3.174 PMLIB::Metric::Correlation

Functions Reference

distance_correlation (named)

```

l1      lref  first list
l2      lref  second list
s1      href  first histogram
s2      href  second histogram
=>     real  distance

```

Either `l1`, `l2` (lists) or `s1`, `s2` (histograms) must be given. Returns Correlation distance between histograms (bigger priority) or between lists.

For specifics about Correlation distance see section about `P_dmp` in PMSE manual.

F.3.175 PMLIB::Metric::Cosine

Functions Reference

distance_cosine (named)

```

l1      lref  first list
l2      lref  second list
s1      href  first histogram
s2      href  second histogram
=>     real  distance

```

Either l_1 , l_2 (lists) or s_1 , s_2 (histograms) must be given. Returns Cosine similarity between histograms (bigger priority) or between lists.

For specifics about Cosine distance see section about P_{dmp} in PMSE manual.

F.3.176 PMLIB::Metric::Czekanowski

Functions Reference

distance_czekanowski (named)

$l_1 s_1$	<code>lref href</code>	point 1
$l_2 s_2$	<code>lref href</code>	point 2
<code>normalized</code>	<code>bool</code>	should we project result to $[0,1]$ interval?
<code>=></code>	<code>real[0>]</code>	distance

F.3.177 PMLIB::Metric::DamerauLevenshtein

Functions Reference

distance_damerau_levenshtein (named)

$l_1 s_1$	<code>lref href</code>	point 1
$l_2 s_2$	<code>lref href</code>	point 2
<code>normalized</code>	<code>bool</code>	should we project result to $[0,1]$ interval?
<code>=></code>	<code>real[0>]</code>	distance

F.3.178 PMLIB::Metric::Dice

Functions Reference

distance_dice (named)

$l_1 s_1$	<code>lref href</code>	point 1
$l_2 s_2$	<code>lref href</code>	point 2
<code>normalized</code>	<code>bool</code>	should we project result to $[0,1]$ interval?
<code>=></code>	<code>real[0>]</code>	distance

This is a special case for PMLIB::Metric::Tversky distance_tversky for:

```
coef12 => 0.5
coef21 => 0.5
```

F.3.179 PMLIB::Metric::Discrete

Functions Reference

distance_discrete (named)

$l_1 s_1$	<code>lref href</code>	point 1
$l_2 s_2$	<code>lref href</code>	point 2
<code>normalized</code>	<code>bool</code>	should we project result to $[0,1]$ interval?
<code>=></code>	<code>real[0>]</code>	distance

F.3.180 PMLIB::Metric::Euclid**Functions Reference****distance_euclid (named)**

l1 s1	lref href	point 1
l2 s2	lref href	point 2
type	enum	'num' or 'str'
=>	real[0>]	distance

F.3.181 PMLIB::Metric::Hamming**Functions Reference****distance_hamming (named)**

l1 s1	lref href	point 1
l2 s2	lref href	point 2
normalized	bool	should we project result to [0,1] interval?
=>	real[0>]	distance

F.3.182 PMLIB::Metric::Jaccard**Functions Reference****distance_jaccard (named)**

l1 s1	lref href	point 1
l2 s2	lref href	point 2
normalized	bool	should we project result to [0,1] interval?
=>	real[0>]	distance

F.3.183 PMLIB::Metric::Jaro**Functions Reference****distance_jaro (named)**

l1 s1	lref href	point 1
l2 s2	lref href	point 2
normalized	bool	should we project result to [0,1] interval?
=>	real[0>]	distance

F.3.184 PMLIB::Metric::JaroWinkler**Functions Reference****distance_jaro_winkler (named)**

l1 s1	lref href	point 1
l2 s2	lref href	point 2
normalized	bool	project result to [0,1] interval?

prefix_weight	real	prefix weight (default: 0.1)
prefix_length_min	int	(minimal prefix length default: 3)
=>	real[0>]	distance

F.3.185 PMLIB::Metric::Kulezinski

Functions Reference

distance_kulezinski (named)

l1 s1	lref href	point 1
l2 s2	lref href	point 2
=>	real[0>]	distance

F.3.186 PMLIB::Metric::Levenshtein

Functions Reference

distance_levenshtein (named)

l1 s1	lref href	point 1
l2 s2	lref href	point 2
normalized	bool	should we project result to [0,1] interval?
=>	real[0>]	distance

F.3.187 PMLIB::Metric::Mahalanobis

Functions Reference

distance_mahalanobis (named)

l1	lref	first list
l2	lref	second list
s1	href	first histogram
s2	href	second histogram
matrix	matrix	regular matrix in the form [[row1], [row2] ...]
type	enum	'num' or 'str'
=>	real	distance

Either `l1`, `l2` (lists) or `s1`, `s2` (histograms) must be given. Returns Mahalanobis distance between histograms (bigger priority) or between lists.

`matrix` defines measure of flexure.

For specifics about Mahalanobis distance see section about `P_dmp` in PMSE manual or http://en.wikipedia.org/wiki/Mahalanobis_distance

F.3.188 PMLIB::Metric::Manhattan

Functions Reference

distance_manhattan (named)

<code>l1</code>	<code>lref</code>	first list
<code>l2</code>	<code>lref</code>	second list
<code>s1</code>	<code>href</code>	first histogram
<code>s2</code>	<code>href</code>	second histogram
<code>tokenizer</code>	<code>cref</code>	token's tokenizer
<code>subdistance</code>	<code>str</code>	subdistance name
<code>type</code>	<code>enum</code>	'num' or 'str'
<code>=></code>	<code>real</code>	distance

Either `l1`, `l2` (lists) or `s1`, `s2` (histograms) must be given. Returns Manhattan distance between histograms (bigger priority) or between lists.

Manhattan distance is special case of Minkowski distance for $p=1$.

`tokenizer` and `subdistance` options has same meaning as in `x2s_distance_euclid`

For specifics about Manhattan distance see section about `P_dmp` in PMSE manual.

F.3.189 PMLIB::Metric::Minkowski

Functions Reference

`distance_minkowski (named)`

<code>l1</code>	<code>lref</code>	first list
<code>l2</code>	<code>lref</code>	second list
<code>s1</code>	<code>href</code>	first histogram
<code>s2</code>	<code>href</code>	second histogram
<code>p</code>	<code>real</code>	positive Minkowski's coefficient
<code>type</code>	<code>enum</code>	'num' or 'str'
<code>=></code>	<code>real</code>	distance

Either `l1`, `l2` (lists) or `s1`, `s2` (histograms) must be given. Returns Minkowski distance between histograms (bigger priority) or between lists.

For specifics about Minkowski distance see section about `P_dmp` in PMSE manual.

F.3.190 PMLIB::Metric::NeedlemanWunsch

Functions Reference

`distance_needleman_wunsch (named)`

<code>l1</code>	<code>lref</code>	first list
<code>l2</code>	<code>lref</code>	second list
<code>s1</code>	<code>href</code>	first histogram
<code>s2</code>	<code>href</code>	second histogram
<code>d</code>	<code>real</code>	sanction for insertion of zero
<code>score</code>	<code>href</code>	structure of the form <code>\$score->{from}{to} = score</code>
<code>=></code>	<code>real</code>	distance

Either `l1`, `l2` (lists) or `s1`, `s2` (histograms) must be given. Returns Needleman-Wunsch score between vectors.

For specifics about Needleman-Wunsch score see section about `P_dmp` in PMSE manual.

F.3.191 PMLIB::Metric::Overlap

Functions Reference

distance_overlap (named)

```

l1|s1      lref|href  point 1
l2|s2      lref|href  point 2
normalized bool        should we project result to [0,1] interval?
=>         real[0>]   distance

```

F.3.192 PMLIB::Metric::Renkonen

Functions Reference

distance_renkonen (named)

```

l1|s1      lref|href  point 1
l2|s2      lref|href  point 2
normalized bool        should we project result to [0,1] interval?
=>         real[0>]   distance

```

F.3.193 PMLIB::Metric::RusselRao

Functions Reference

distance_russel_rao (named)

```

l1      lref  first list
l2      lref  second list
s1      href  first histogram
s2      href  second histogram
=>     real  distance

```

Either `l1`, `l2` (lists) or `s1`, `s2` (histograms) must be given. Returns Russel-Rao dissimilarity between vectors (usually boolean).

For specifics about Russel-Rao distance see section about `P_dmp` in PMSE manual.

F.3.194 PMLIB::Metric::SmithWaterman

Functions Reference

distance_smith_waterman (named)

```

l1      lref  first list
l2      lref  second list
s1      href  first histogram
s2      href  second histogram
d       real  sanction for insertion of zero
score  href  structure of the form $score->{from}{to} = score

```

```
=>    real    distance
```

Either `l1`, `l2` (lists) or `s1`, `s2` (histograms) must be given. Returns Smith-Waterman score between vectors.

For specifics about Smith-Waterman score0 see section about `P_dmp` in PMSE manual.

F.3.195 PMLIB::Metric::SokalSneath

Functions Reference

distance_sokal_sneath (named)

```
l1      lref    first list
l2      lref    second list
s1      href    first histogram
s2      href    second histogram
=>     real    distance
```

Either `l1`, `l2` (lists) or `s1`, `s2` (histograms) must be given. Returns Sokal-Sneath dissimilarity between vectors (usually boolean).

For specifics about Sokal-Sneath distance see section about `P_dmp` in PMSE manual.

F.3.196 PMLIB::Metric::Tanimoto

Functions Reference

distance_tanimoto (named)

```
l1|s1      lref|href    point 1
l2|s2      lref|href    point 2
normalized bool          should we project result to [0,1] interval?
=>         real[0>]     distance
```

This is same as `PMLIB::Metric::Tversky` `distance_tversky` for params:

```
coef12 => 1
coef21 => 1
```

F.3.197 PMLIB::Metric::Tversky

Functions Reference

distance_tversky (named)

```
l1|s1      lref|href    point 1
l2|s2      lref|href    point 2
coef21     real[0>]     l2-l1 coef
coef12     real[0>]     l1-l2 coef
normalized bool          should we project result to [0,1] interval?
=>         real[0>]     distance
```

There is a condition for `distance_tversky`:

coef21 + coef12 must be positive

F.3.198 PMLIB::Metric::Typo

Functions Reference

find_min_value (named)

i	int	position i in matrix
j	int	position j in matrix
matrix	lref	list of the form [matrix,matrix]
sets	lref	sets
=>	href	{levenshtein => real, geometric => real}

distance_typo (named)

l1 s1	lref href	point 1
l2 s2	lref href	point 2
=>	real[0>]	distance

Computes geometrical distance between two characters. The base of the algorithm is derived from the levenshtein distance computation.

F.3.199 PMLIB::Metric::Yule

Functions Reference

distance_yule (named)

l1	lref	first list
l2	lref	second list
s1	href	first histogram
s2	href	second histogram
=>	real	distance

Either l1, l2 (lists) or s1, s2 (histograms) must be given. Returns Yule dissimilarity between vectors (usually boolean).

For specifics about Yule distance see section about p_dmp in PMSE manual.

F.3.200 PMLIB::PMLIB

PetaMem Library: PMLIB specific tools PMLIB metafunctionality.

Functions Reference

get_subs (named)

1	str	path
=>	href	hash of the form file => [functions list]

Walks through PMLIB code and extract subs names (methods and functions).

F.3.201 PMLIB::Queue

PetaMem Library: Implementation of queues (priorized and otherwise) This will implement a queue or (priorized) multiqueue.

Methods Reference

BUILD (named)

```

    amnesia    enum    type of forgetting mode (def: 'old' - oldest first)
    maxstor    int     maximum length of all subqueues (def: 0 - unlimited)
    storage    lref    list of lists to initialize queue with data
    =>         queue   queue

```

Object build method.

If `maxstor` is given, and the initial data structure should be bigger than `maxstor` indicates, this will NOT truncate the initial data structure. Unlike the `store` method, `BUILD` does not check here for consistency but expects sane parameters in the first place. Of course further storage will be impossible unless the Queue size drops below the `maxstor` threshold.

store (named)

```

    elem      elem    element to store
    prio      int     int priority
    =>        int     queue length

```

Store an element in queue. Optionally a priority can be given if this is a prioritized queue (multiqueue). Returns length of this queue after the store.

If `maxstor` feature is enabled (parameter has value bigger than 0), before a storage occurs, the size of the queue is compared with the `maxstor` parameter.

Then, depending on the value of the `amnesia` parameter (old versus new), the maximum storage is handled like follows:

Only if size is smaller than `maxstor`, a storage actually happens.

fetch_next (positional)

```

    1      int     priority offset
    =>    elem    next element in queue

```

Fetch an element from queue. If this is a prioritized queue, then the element from the subqueue holding the highest priorities is fetched. If `offset` is given, this offset is subtracted from the highest priority.

fetch_last (void)

```

    =>    elem    removed one

```

Removes the oldest element in queue, which is by definition the element that would be fetched last. (I.e. last element in the lowest-prio subqueue).

get_storage (void)

```
=>   lref   storage
```

Returns the internal data structure that is used for storing and representing the (prioritized) queues. Basically this is a list of lists.

serialize (void)

```
=>   lref   serialized
```

This will grab all subqueues and return a list in sorted order with the next-in-line element of the subqueue with the highest priority as the first element and the last element of the subqueue with the lowest priority as the last element.

compare (positional)

```
1   queue   queue
=>  bool    comparation
```

Compare this queue object with another. Returns true if they are identical, false otherwise.

peek_next (positional)

```
1   int     offset  priority offser
=>  elem    next   elem
```

This will identify the next-in-line element and return it, but without removing it from the queue as `fetch_next([offset])` would do. The optional argument `<offset>` will subtract the offset from highest priority.

peek_last (void)

```
=>  elem    next   elem
```

This will identify the last-in-line element and return it, but without removing it from the queue as `fetch_last([offset])` would do. The optional argument `<offset>` will subtract the offset from highest priority.

sizeall (void)

```
=>  int     size
```

This will return the total size of the queue, i.e. adding up all elements in all subqueues. Also, this method is STRINGIFIED and NUMERIFIED so the number is returned by simply putting the object in the respective context.

size (positional)

```
1   int     priority
=>  int     size
```

This will return the size of the total queue (by using the `sizeall()` method), or just the size of a specific subqueue if the priority of that subqueue is given.

cs_freeze (void)

```
=>  freezed   serialized object
```

Own "Moose" Freeze. Serialization for our Object.

cs_thaw (positional)

```

1    freezed    freezed data
=>   queue     restored object

```

See `cs_freeze`. This is the corresponding thaw method.

F.3.202 PMLIB::Regex**PetaMem Library: Regular Expression Functions****Functions Reference****build_rx (named)**

```

op      enum    type of logical operation, default: 'OR'
pre     str     what to replace the start-boundary with (def: \b)
post    str     what to replace the end-boundary with (def: \b)
qr      bool    flag: do we want a qr-escaped return value (def: yes)
tokens  lref    list of regexes to concatenate
=>     str     regex unless C<qr>
=>     rx      regex if C<qr>

```

Build a regular expression from list of tokens.

Supported operations are OR, AND, NOT, NAND, NOR, XOR, NXOR.

contains_lookaround (positional)

```

1    list    regexes|strings
=>   bool    any of them contains lookaround?

```

is_qr (positional)

```

1    str     str
=>   bool    is regex?

```

GUESS if the string is a stringified regex

em (positional)

```

1    str     str
=>   rx      regex

```

Build a regular expression from string using `re 'eval'`.

es (positional)

```

1    str     str
2    str     str
=>   rx      return value of s{C<1>}{C<2>}

```

Substitute from string using `re 'eval'`.

test_rx (positional)

```

1   lref    list of tokens to test against a regular expression
2   rx      regular expression to test against
=>  bool    true if all are matched, false else

```

Test a regular expression against a list of tokens.

rx_len (positional)

```

1   rx      qr-ed string
=>  int     size of regular expression

```

Size means a number of length of the regex including qr{} chars.

rx_unqr (positional)

```

1   rx      qr-ed string
=>  str     un-qr-ed string

```

F.3.203 PMLIB::Scalar

PetaMem Library: scalar tools

Functions Reference

activity_ascii_animation (positional)

```

1   list    animation (default: rotation bar)
=>  iterator current animation string

```

Will provide different ASCII animation sequences to show activity. This subroutine contains a closure that will memoize the status last time it was called. The default ascii animation is a clockwise rotating bar -\|/, but other animations can be given in initialization.

```

# initialize with default ASCII animation (rotating bar)
my $act = activity_ascii_animation();

# now for some actions
for (@some_activity) {

    # call the inner closure to return current animation string
    $act->();
}

# now we want to initialize a counter clockwise rotating bar
my $act = activity_ascii_animation(qw(- / | \));

# Multi-char animations are also possible,
# as is a list reference as init argument

# init "ping-pong dot"
my $act = activity_ascii_animation(['. ', ' . ', ' .', ' . ']);

```

activity_progress_bar (named)

```

maxlen    int        maximum integer (goal/full indicator (mandatory!))
char      char       chars to fill the progress bar (default: '#')
showlen   int        length of progressbar (without brackets) (def: 40)
currilen  int        position/index where progressbar starts (def: 0)
=>        iterator   current animation string

```

Initializes and constructs a progress bar for subsequent output.

activity_progress_counter (named)

```

maxcount   int        maximum (goal/full indicator (mandatory!))
direction  enum       'up' (default) or 'down'
currilen   int        position/index of progressbar's start (def: 0)
=>         iterator   progressbar

```

Initializes and constructs a progress counter for subsequent output.

F.3.204 PMLIB::Script

PetaMem Library: Script functions

Functions Reference

get_wiki_script_for_iso (positional)

```

1         str        iso
=>        str        wiki script name
=>        undef      undef if not found

```

get_unicode_class_for_iso (positional)

```

1         str        iso
=>        str        unicode class matching this lang
=>        undef      undef if not found

```

get_letter_regex_for_iso (positional)

```

1         str        iso
=>        str        regex matching any letter for given lang
=>        undef      undef if not found

```

F.3.205 PMLIB::Serialize

PetaMem Library: Various serializing functions

Functions Reference

deserialize (named)

```

format    enum       desired deserialization format (can be guessed)
data      str        serialized data structure
=>        any        data structure

```

guess_format (named)

```

data    str    serialized data structure
=>     str    format name

```

Recognizes following formats:

```

std (see pp_struct)
xml
yaml
json
plaincmd (PMLS commands)

```

serialize (named)

```

format  enum   desired serialization format (can be guessed)
data    any    perl data structure to be serialized
=>     str    serialized data structure

```

Possible formats are:

```

std (see pp_struct)
xml
yaml
json
plaincmd (PMLS commands)

```

pp_struct (named)

```

struct  href||ref  data structure
=>     str         serialized data structure

```

Pretty printing function, but returns string instead of printing. Supports hash, list and simple scalars.

F.3.206 PMLIB::Stack

PetaMem Library: Implementation of a stack (priorized and otherwise) Implementation of a (optionally prioritized) Stack, by inheriting most of its codebase from PMLIB::Queue and overriding methods to provide LIFO functionality instead of FIFO

You will get a fairly decent documentation for this module if you look at PMLIB::Queue and replace there the word queue with stack.

Methods Reference**fetch (void)**

```

=>     elem     elem
=>     undef    if no elem

```

Overrides PMLIB::Queue method fetch, to get the LIFO functionality instead of FIFO.

remove_oldest (void)

```
=> elem elem
=> undef if no elem
```

Removes the oldest element in queue.

peek (void)

```
=> elem elem
=> undef if no elem
```

overrides PMLIB::Queue method fetch, to get the LIFO functionality instead of FIFO

serialize (void)

```
=> lref serialize
```

overrides PMLIB::Queue method serialize, to account for the LIFO functionality instead of FIFO

F.3.207 PMLIB::Stochastic::P

PetaMem Library: operations over probabilities of independent phenomenon

Functions Reference

not_p (positional)

```
1 list probabilities of events p1...pn
=> list probabilities of events NOT(p1...pn)
```

Negated probabilities of given events.

and_p (positional)

```
1 list probabilities of events p1...pn
=> prob probability of event AND(p1...pn)
```

Probability that all independent events do happen.

nand_p (positional)

```
1 list probabilities of events p1...pn
=> prob probability of event NAND(p1...pn)
```

Probability at least one of independent events doesn't happen.

or_p (positional)

```
1 list probability of events p1...pn
=> prob probability of event OR(p1...pn)
```

Probability that at least one of independent events does happen.

nor_p (positional)

```
1 list probabilities of events p1...pn
=> prob probability of event NOR(p1...pn)
```

Probability that none of independent events happen.

nxor_p (positional)

```
1 list probabilities of events p1...pn
=> prob probability of event NXOR(p1...pn)
```

Probability that any but not one of independent events do happen.

xor_p (positional)

```
1 list probabilities of events p1...pn
=> prob probability of event XOR(p1...pn)
```

Probability that only one one of independent events does happen.

ded_p (positional)

```
1 prob probability of event1 p1
2 prob probability of event2 p2
=> prob probability of event DED(p1,p2)
```

Probability, that if event2 occurred, so occurred event1.

imp_p (positional)

```
1 prob probability of event1 p1
2 prob probability of event2 p2
=> prob probability of event IMP(p1,p2)
```

Probability, that if event1 occurs, so occurs event2.

F.3.208 PMLIB::Stochastic::Sample

Functions Reference

sample_mean (positional)

```
1 lref listref with numeric values
=> undef if no input vector given
=> undef if input listref is empty
=> real arithmetic mean
```

Returns the arithmetic mean of given numeric vector.

sample_correlation (positional)

```
1 lref vector1
2 lref vector2
=> undef if v1 or v2 does not exist
=> real correlation
```

Returns sample correlation coefficient between two vectors.

sample_covariance (positional)

```
1 lref vector1
2 lref vector2
```

```
=> undef   if v1 or v2 does not exist
=> real    covariance
```

Returns the covariance value of two vectors.

sample_variance (positional)

```
1  lref    vector
=> undef   if vevtor does not exist
=> real    variance
```

Returns the variance of vector.

F.3.209 PMLIB::System

PetaMem Library system tools

Functions Reference

myrun (positional)

```
1  lref    cmd
2  fh      stdin
3  fh      stdout
4  fh      stderr
=> href    back href
```

Run a subprocess with input/ouput redirection

Back href contains:

```
stdout
stderr
retval
signal
coredump
```

perlv (positional)

```
1  int     requested perl version
=> bool    do we run in newer perl?
```

Given a version this function checks if we actually run a perl at least of this version. If no argument is given, the current perl version is returned.

F.3.210 PMLIB::System::HW

Functions Reference

get_num_cpus (void)

```
=> int     number of cpus
```

get_avg_load (void)

```
=> int avg load
```

get_realfree_mem (void)

```
=> int realfree mem in bytes
realfree = free + cached + inactive
```

get_proc_mem (positional)

```
1 int index
=> int
```

Return process memory by default, or in general value at index of proc/statm.

get_free_mem (void)

```
=> int free mem in bytes
```

get_cached_mem (void)

```
=> int cached mem in bytes
```

get_inactive_mem (void)

```
=> int inactive mem in bytes
```

F.3.211 PMLIB::System::HW::Keyboard

SYNOPSIS

```
my ($pos_hr, $mapped_hr) = get_kb_model('default', 'deu');
my $distance = get_chars_distance($pos_hr, $mapped_hr, 'q', '@');
```

Functions Reference

get_key_positions (positional)

```
1 str kb name
=> href positions
```

Evaluates a given keyboard geometry model and returns a href with <key> = key-code and <value> = [x, y] listref with the absolute coordinates (in mm) of the centers of the keys on the given keyboard. Keycodes are independent of the keyboard language mapping.

get_chars_distance (positional)

```
1 href positions
2 href mapping
3 str char1
4 str char2
=> real distance
```

Higher level function, which takes a positional href (that's what e.g. `get_key_positions` returns), a char-to-keycode mapping href (that's what e.g. `map_char_to_key` returns) and two characters. It returns a listref consisting of the euclidean distance of the two

characters on that given keyboard with the given mapping, as well as the difference in modifiers.

get_kb_model (positional)

```
1   str    name (default: 'default')
2   str    iso (default: 'eng')
=> href   positions
```

Given a keyname, returns keycode (or possibly keycode combination).

get_keys_distance (positional)

```
1   href   positions
2   str    key 1
3   str    key 2
=> real    distance
```

Given a key positional href (that's what e.g. `get_key_positions` returns) and two keycodes, returns the euclidean distance of the centers of the two keycodes.

get_max_distance (positional)

```
1   href   kb model
=> real    max_distance
```

This function takes only one input argument: keyboard model. It returns the maximal geometrical distance on the given keyboard.

get_neighbors (named)

```
kbmodel  href   kb model
kcode    int    key code
=>        href   neighbours
```

Input argument must be a hashreference with these keys:

```
kb_model
kcode
```

This function returns a hash with info about neighbors of the given character. If the input character has a modifier, returned characters will also be 'modified'.

map_char_to_key (positional)

```
1   char   char
2   lref   modifiers
=> key    key
```

Takes mapping data for a given language and transforms it into a href consisting of <key> = character and <value> = listref of keycode 1 and listref of modifiers 2.

F.3.212 PMLIB::System::OS

Functions Reference

get_binaries_paths (positional)

```

1   str   binaries name
2   lref  search path (default: $ENV{PATH})
=>  href  hash of the form binary => path to binary

```

get paths to a list of binaries given

get_binary_path (positional)

```

1   str   binary name
2   lref  search path (default: $ENV{PATH})
=>  str   path to binary

```

get full path to a binary (like shell command 'which')

local_port_is_free (positional)

```

1   int   port
=>  bool  is free?

```

get_free_port (positional)

```

1   int   from
1   int   to
=>  int   free port

```

Search ports from `from` to `to` and return the first free port found.

F.3.213 PMLIB::Tag

Operations with tags

Methods Reference

tag2verbatim (positional)

```

1   str   tag
2   str   language code (default: eng)
=>  str   syntactic category name

```

class2tag (positional)

```

1   str   class name
=>  str   tag

```

convert (positional)

```

1   lref  context
2   href  arguments
=>  lref  tokens / tags
      undef  undef if no args were specified

```

This is a generic engine for applying of transformation rules. Arguments require 2 named parameters:

```

lang  language of given tagset
conv  direction of conversion

```

iso2tag_hr (positional)

```
1   str    language code (default: eng)
=> href   hash of class name translations
```

F.3.214 PMLIB::Tag::CLAWS5**Operations with CLAWS5 tags****Methods Reference****BUILD (void)**

```
=> - private
```

Private Moose method.

class2tag (positional)

```
1   str    class name (in ENG)
=> str    tag
```

ENG l10n available only.

convert (positional)

```
1   lref   context: tokens/tags
=> lref   tokens/ PMTS tags
      undef undef if no args were specified
```

Input list reference should have this structure:

```
[
  [token1, tag1],
  [token2, tag2],
  ...
]
```

iso2tag_hr (positional)

```
1   str    language code (default: eng)
=> href   hash of class name translations
```

ENG l10n available only.

parse (positional)

```
1   str    tagged text string
2   str    format
=> href   tokens / tags
```

There are 3 possible formats for CLAWS data:

```
horizontal
vertical
xml
```

Returned data structure is a hash reference with tokens / tags arrays.

F.3.215 PMLIB::Tag::Multext

Operations with Multext east tags

Methods Reference

BUILD (void)

```
=> - private
```

Private Moose method.

class2tag (positional)

```
1   str   class name (in ENG)
=>  undef
```

Not implemented yet.

convert (positional)

```
1   lref   context
=>  lref   tokens / tags
      undef  undef if no args were specified
```

Input list reference should have this structure:

```
[
  [token1, tag1],
  [token2, tag2],
  ...
]
```

iso2tag_hr (positional)

```
1   str   language code (default: eng)
=>  href  hash of class name translations
```

Not implemented yet.

F.3.216 PMLIB::Tag::Penn

Operations with Penn Treebank tags

Methods Reference

BUILD (void)

```
=> - private
```

Private Moose method.

class2tag (positional)

```
1   str   class name (in ENG)
=>  str   tag
```

ENG l10n available only.

convert (positional)

```
1   lref   context
=>  lref   tokens / tags
      undef   undef if no args were specified
```

Input list reference should have this structure:

```
[
  [token1, tag1],
  [token2, tag2],
  ...
]
```

iso2tag_hr (positional)

```
1   str   language code (default: eng)
=>  href   hash of class name translations
```

ENG l10n available only.

parse (positional)

```
1   str   string encoded in Penn format
=>  lref   tokens / tags
```

A list reference with tags / tokens arrays is returned. The format is:

```
[
  [<token1>, <tag1>],
  [<token2>, <tag2>],
  ...
]
```

F.3.217 PMLIB::Tag::Pmts

Operations with Pmts

Methods Reference

BUILD (void)

```
=> - private
```

Private Moose method.

convert (positional)

```
1   href   data to convert
2   str   target tagset
=>  href   array ref. with tags
```

The input hash reference must have following structure:

```
{
  tags   => [<tag1>, <tag2>, ...],
  tokens => [<token1>, <token2>, ...],
}
```

Both tags and tokens arrays must have the same length. These tagsets are currently supported:

```
Penn
CLAWS5
```

iso2tag_hr (positional)

```
1   str      language code (default: eng)
=>  href     hash of class names translations
```

tag2verbatim (positional)

```
1   str      tag
2   str      language code (default: eng)
=>  str      l10n of tag
```

pmts2CLAWS7 (positional)

```
1   str      language code (default: eng)
=>  str      tag
```

NOT IMPLEMENTED YET!

F.3.218 PMLIB::Test

PetaMem Library: Test library functions

Functions Reference

basic_prolog (positional)

```
1   str      module name
=>  (str,int) (module name, tests done)
```

Checks following entire module characteristics:

```
use_ok
require_ok
POD syntax
POD coverage
```

POD tests are done unless PM_MASTERENV environment variable is set.

func_prolog (positional)

```
1   str      function name
=>  (str,cref) (name, code)
```

Checks if function of some name is present in main package and returns this name and coderef.

gen (positional)

```
1 list mixture of distributions
=> any generated value
```

Generates value from given distribution. Distribution is specified by a list containing various elements, each representing a distribution (part of the mixture). All distributions in the mixture have same weight.

Here we can see supported distributions (elements of the list):

```
something else than hashref: simply means degenerated distribution
hashref with one couple: depends on its key:

sub => [cref, lref]: lref is passed to cref
and returns value is the generated one
real => [from,to,count]: generates lref of C<count> values
from interval [C<from>,C<to>]
int => [from,to,count]: generates lref of C<count> values
from interval [C<from>,C<to>]
char_lr => count: generates C<count> letters lref
char_lr_uniq => count: same, but they are uniq (and C<count> <= 26)
```

get_tpath (positional)

```
1 str path to append
=> str path
```

Gets a current path of the calling test script.

method_prolog (positional)

```
1 str module name
2 str method name
=> (str,cref) (name, code)
```

Checks if method of some name is present and returns this name and coderef.

pmtloop (named)

```
name str tested function/method/cmd name
sub cref tested function (for method is not obligatory)
testdata lref testcases (see further)
object obj classname or object - for methods only
skip bool|cref skip all testcases?
=> int number of tests done
```

testdata is a list of testcases of the following form.

Input:

```
args lref|scalar args to be tested (interpolated if list)
cmd lref shell command
```

Tests:

cmp	lref	expected return value after possible hook
like	rx	regex expected to match return value
stdout	rx str	stdout output - regex or string
stderr	rx str	stderr output - regex or string
exception	rx	output after exception capture test
object	obj	object check test
other	cref lref	other test to perform or list of them
sbl	str	storable file test
fileeq	str	file content equality test

Others:

skip	bool cref	skip this testcase?
retval_hook	cref	sub to transform return value if needed
msg	str	test message
prehook	cref	code to run before the testcase
posthook	cref	code to run after the testcase

pmt_similarity (named)

str1	str	string1
str2	str	string2
precision	real	tolerated difference
name	str	sub name
msg	str	test msg
=>	int	number of tests done

Compare 1 and 2 based on normalized levenshtein distance. Test passes if both strings are similar.

pmtcmp (positional)

1	any	result
2	any	expected value
3	str	test name
=>	val	1 = number of tests done

Performs cmd test.

pmtlike (positional)

1	any	result
2	rx	expected value
3	str	test name
=>	val	1 = number of tests done

Performs like test.

pmtunlike (positional)

1	any	result
2	rx	expected value
3	str	test name

```
=> val 1 = number of tests done
```

Performs unlike test.

pmtok (positional)

```
1 bool test result
2 str test name
=> val 1 = number of tests done
```

Performs ok test.

pmtok_env (void)

```
=> val 3 = number of tests done
```

Performs environment variables testing. Tests BIN and INC environment variables for PMSE, PMLS, PMLIB projects.

pmtcmp_stderr (positional)

```
1 cref code
2 rx regex to match output of C<1>
3 str test name
=> val 1 = number of tests done
```

Testing captured STDERR.

pmtcmp_stdout (positional)

```
1 cref code
2 rx regex to match output of C<1>
3 str test name
=> val 1 = number of tests done
```

Testing captured STDOUT.

pmt_pod_coverage (positional)

```
1 str module name
2 str message
=> val 1 = number of tests done
```

Tests pod coverage of 1.

pmt_pod_format (positional)

```
1 str .pm or .pod file
=> val 1 = number of tests done
```

Extracts POD from the file and tests formatting of specific parts.

pmt_pod_struct (positional)

```
1 str .pm or .pod file
=> val 1 = number of tests done
```

Extracts POD from the file and tests for various PetaMem conventions.

pmt_pod_syntax (positional)

```

1   str    module path
2   str    message
=>  val    1 = number of tests done

```

Tests pod syntax of 1.

pmt_fuzzy (named)

```

name          str    test name
runs          int    how many tests to perform
sub_tested    cref   tested code unit
sub_validate  cref   validator
template      cref   sub to generate data for C<sub_tested>
=>           val    1 = number of tests done

```

Fuzzy testing interface. First, `template` is runned to generate a data. Usually `gen` function is used inside this sub. Then we run `sub_tested` and validate data with the `sub_validate`, which gets 2 parameters:

```

lref of result of C<sub_tested> call
lref to generated data

```

This is runned `runs` times. If all results are valid, test pass.

Do nothing if `PM_NO_FUZZY_TESTS` environment variable is set.

pmt_exception (positional)

```

1   cref   code
2   rx     regex to match output of C<1>
3   str    test name
=>  val    1 = number of tests done

```

Testing captured exceptions output.

ut8dump (positional)

```

1   any    data
2   str    description
3   bool   noprint
=>  undef  unless noprint
=>  str    else serialized C<1>

```

Wrapper for `Data::Dumper::Dumper`.

safe_ut8dump (positional)

```

1   any    data
2   str    description
3   bool   noprint
=>  undef  unless noprint
=>  str    else serialized C<1>

```

Same as `ut8dump` unless data are too big.

pm_profile (positional)

```

1   cref   test sub
2   bool   tests disabled?
3   str    test name
4   real   how many seconds or 0 for default
=>  int    number of tests done

```

If PM_PROFLOG environment variable is set, run tests for at least some time, computes number of calls and write into the file with the name stored in PM_PROFLOG.

Unless PM_PROFLOG is set, simply run 1.

pm_tempfile (positional)

```

1   str    file name start (optional)
=>  str    filename absolute path

```

Create temporary file and returns some information about it.

File is removed automatically unless PM_KEEP_TEMP_FILES environment variable is set (good for debugging reasons).

pm_tmpdir (positional)

```

1   str    dir name start (optional)
=>  str    tmp dir absolute path

```

Create temporary directory and returns some information about it.

Directory is removed automatically unless PM_KEEP_TEMP_FILES environment variable is set (good for debugging reasons).

tmp_location (positional)

```

1   str    new location (optional)
=>  str    location

```

Get/set location of temporary files.

F.3.219 PMLIB::Text::Filter**PetaMem Library: Text filtering****Functions Reference****strip_wiki_markup (named)**

```

txt   str    wiki text
=>    str    stripped wiki markup

```

strip_html_markup (named)

```

txt   str    html text
=>    str    stripped html markup

```

F.3.220 PMLIB::Time

PetaMem Library: Time-related functionalities

Functions Reference

iso_time (positional)

```

1   int   seconds
=>  str   time in iso format

```

get current ISO-time string

time2readable (positional)

```

1   int   seconds
2   int   decimals
=>  str   readable time

```

F.3.221 PMLIB::Timer

PetaMem Library: Time-related functionalities

Functions Reference

iso_time (positional)

```

1   int   seconds
=>  str   time in iso format

```

get current ISO-time string

time2readable (positional)

```

1   int   seconds
2   int   decimals
=>  str   readable time

```

F.3.222 PMLIB::Tree::Nary

basic N-ary Tree implementation Basic implementation of a N-ary tree. This implements base functionality which is agnostic of any data being carried by the trees nodes. Specific handling of such data is handled by classes inheriting this base class.

NOMENCLATURE

node

The central unit of a tree. It contains data (a Property, see Core::NLP::Property.pm) and may be the parent of other nodes (its children) and also may be itself a child of another node.

Tree/Node Semantics: Although the object reference in this module allows for trees and nodes being handled interchangeably, we will distinguish between these two in

our methods: We will consider a meaning-object reference a tree, if the method that is applied on it, will consider it (potentially) as whole, and (potentially) touch also nodes below its childs hierarchy or the nodes ancestors.

tree

A single node or a group of nodes linked together as parents and their children. All nodes of that group must be linked in some way, or it is not a single tree, but several trees.

Tree/Node Semantics: Although the object reference in this module allows for trees and nodes being handled interchangeably, we will distinguish between these two in our methods: We will consider a meaning-object reference a tree, if the method that is applied on it, will consider it (potentially) as whole, and (potentially) touch also nodes below its childs hierarchy or the nodes ancestors.

parent

A node that has children. So "parent" and "leaf" are mutually exclusive.

child

A node that has a parent. Can itself be either parent or leaf.

leaf

A node that has no children. So "parent" and "leaf" are mutually exclusive, but a child can be a leaf.

sibling

A node that has the same parent as another node.

ancestor

A node that is either parent of a node or an ancestor if the nodes parent. (recursive definition ;-). See "root", which is ancestor of all nodes in a tree.

root

The ancestor of all nodes in a given tree.

descendant

Either a child of a node, or one of its descendants. (recursive definition ;-)

Methods Reference

new (positional)

```
1   str  data to fill into the new object
=>  obj  new object
```

Object constructor.

clone (void)

```
=>  obj  cloned object
```

Creates a clone of the current tree.

objectify (positional)

```
1  obj|str    object of data
=>  obj       construct object
```

Function, not method. Will return a `Tree::Nary` object if input wasn't already one, in which case it'll simply return the input.

is_ancestor (positional)

```
1  node      node
=>  bool      is C<1> my ancestor?
```

1 contains reference to a node object (`$node`). Tests whether this `$node` is ancestor of the object that called the method.

is_child (positional)

```
1  node      node
=>  bool      is C<1> child of the self objref?
```

is_descendant (positional)

```
1  node      node
=>  bool      is C<1> descendant of the self objref?
```

Argument contains reference to a node object (`$node`). Tests whether this `$node` is descendant of the object that called the method.

is_leaf (positional)

```
1  node      node
=>  bool      is C<1> leaf?
```

Basically inverse function of `is_parent`, except that it is a real bool context, and does return true if node is leaf and false if node is parent. (`is_parent` can return the number of childs of a node).

is_parent (positional)

```
1  node      node
=>  bool      is C<1> parent of self objref?
```

return whether 1 is parent (number of childs) or leaf (0)

are_siblings (positional)

```
1  lref      nodes
=>  bool      siblings?
```

Are nodes in a listref all siblings (true) or is some of them not?

stringify (void)

```
=>  str      stringified object
```

Data Queries and Handling

is_empty (void)

```
=> bool    is empty?
```

Will return true if data section is empty, false otherwise

is_filled (void)

Inverse function of is_empty. See there.

chg_data (positional)

```
1   cref   code
=>  str    old data
```

1 is a function reference. This function will receive the data of the calling node object and its return value will be written to the nodes data section. So this method can provide arbitrary modifications of the data section.

cpy_data (positional)

```
1   node   node with data
=>  str    old data
```

Copy the data section from the given node 1 to this node.

get_data (void)

```
=>  str    data from node
```

Get data from the node data section.

mov_data (positional)

```
1   node   node with data
=>  str    old data
```

Move data from the node 1 to this node. The data section in the 1-node is then set to the empty string.

set_data (positional)

```
1   str    new data
=>  str    old data
```

Set data section of node with new data. If data given is not defined, an empty string is set. This method will return the old/previous content of the data section.

swp_data (positional)

```
1   node   node with data
=>  str    old data from C<1>
```

Swap the data sections of this node and the node given 1. The return value is the OLD data of the 1-node (that is, the current content of this node).

peek_byidx (positional)

```
1   lref   indices
=>  node   found node
```

Give path to access directly a node within a tree. Indices may be given positive (counting from start, offset = 0) or negative (counting from end, -1 = last element). Returns objref of node found - if found, undef if no such node exists.

If no path is given, or path is empty, the objref of the root node is returned.

Tree Queries and Handling

get_canon (positional)

```
1  str    leaf char
2  str    left char
3  str    right char
=> str    tree structure as string
```

Obtain tree canon. Symbolic representation of the tree structure. Required e.g. for tree comparison. By default, a leaf is denoted by the '*' char, and siblings are grouped by round braces '()'. So a root node with three childs looks like '(***)'.

This method can take 3 optional positional parameters 1. leaf - string to represent a leaf 2. right brace - string to represent the right brace 3. left brace - string to represent the left brace

Which can substantially change the looks of a tree canon - should you need it.

add_childs (positional)

```
1  lref/oref  nodes
=> node      node
```

Adds a node/nodes (arg1 objref/listref) to self as child. The added nodes will be appended after evtl. already present child nodes.

cmp_childs (positional)

NYI: Compare childs of two nodes and determine their relation (set relations)

cpy_childs (positional)

```
1  node    from
2  cref    filtering function
=> node    resulting node
```

Copies the children of a given node (arg1) to self. An optional function reference (arg2) may be given, which will act exactly as in get_nodes (put constraints on child nodes to-be-copied).

del_childs (positional)

```
1  any    polymorphic argument
=> lref   nodes
```

This method will delete one or several childs depending on the arguments given. 1 is a polymorphic argument that defines this nodes behaviour:

```
objref: will be considered object reference to the node, if this
        object reference matches one of the nodes childs, this child
```

will get deleted.

`listref`: will expect a listref of object references. All of those that match existing `chids`, will be deleted from the nodes `chids`.

`funcref`: this will invoke the `get_chids` method with this `funcref` as an argument and the resulting listref will be `chids` considered for deletion.

`hashref`: like `listref`, but the elements are already keys of hash

As all resulting lists get transformed to a hash, evtl. occuring duplicates will get considered only once in the deletion process. This is probably a good thing to do.

get_chids (positional)

```
1   cref   filter
=>  lref   chids
```

Return list of children, that satisfy conditions given in the function reference (`arg1`). If no function reference is given, this method returns all `chids` of the node.

mov_chids (positional)

```
1   node   node
=>  lref   moved nodes
```

Move `chids` from one node (`self`), to another node 1.

swp_chids (positional)

```
1   node   node
=>  node   self
```

Swap the children of (`self`) with those of another node 1.

get_parent (positional)

```
1   node   node
=>  node   parent
```

Get the parent of a given node (1 - `objref`). `Self` is considered the root of the search tree or at least an ancestor as high as the parent. If no parent has been found, `undef` is returned.

get_siblings (positional)

```
1   node   node
=>  lref   nodes
```

Get the siblings of a given node (1 - `objref`). First the parent is searched, then all `chids` that are not the given node are returned.

An optional second argument (`coderef`) may be given. In this case, only those siblings are returned that satisfy the constraints defined by this code. Matching via regular expression, comparisons etc. may be realized this way.

get_child_idx (positional)

```

1   int    child index (default: 0)
2   cref   filter function
=>  node   child

```

Gets the objref of the childnode at the given index (arg1) position. The index may be positive (0 being the first child) or negative (-1 being the last child). If no child at the given index position exists, undef is returned. If no index is given, 0 is assumed.

If an optional second argument (arg2 - coderef) may be given. If given, the children retrieval will use this coderef as filter and only those children that satisfy the filter will be returned. The index will THEN be applied and the respective child will be returned (or undef if no such child exists).

get_descendants (name)

```

func    cref    filter children
mode    enum    'all' (default), 'leafs'
=>      lref    all descendants

```

Returns nodes that are descendants of the object (including the object itself) that called this method.

If an optional argument (coderef) is given, this will filter only those descendants that match the defined constraint. ATTENTION: If a node does not match the given constraint, its children will not get considered even though they might match the constraint. If you want to get such cases, call get_descendants() without arguments and filter the whole list.

Traversal

traverse (named)

```

order    enum    the order in which the trees nodes are visited.
maxdepth int    maximum depth for tree traversal
trav_mask enum    traversal mask
funcref   cref   fnc or method performed upon each node visit
funcargs  href   named arguments to funcref
=>        node   node

```

Traverse a given tree in various ways.

Possible values for order are:

```

PRE_ORDER  (default), first visit the node, then its children
LEVEL_ORDER first the children of the node, then their children
POST_ORDER first visit children, then node itself
RAND_ORDER first visit the node, then its children in random order

```

These parameters are equipped with sensible default values (those necessary and undefined) and passed to `_traverse_generic` which implements the various traversal methods.

Tree traversal: 100 is default, -1 means unlimited. Traversal mask: whether to visit, parents, leafs or all (default)

F.3.223 PMLIB::Types**Types definitions****F.3.224 PMLIB::UpdateCheck**

Checking for various updates.

Functions Reference**get_candidates_for_update_generic (named)**

function	str	name of update function
args	href	parameters for chosen update function
=>	lref	list of possible candidates for update

check_updates_generic (named)

function	str	name of update function
args	href	parameters for chosen update function
=>	lref	detailed information about update

print_updates_generic (named)

function	str	name of update function
args	href	parameters for chosen update function
=>	real	prints results

update_generic (named)

function	str	name of update function
args	href	parameters for chosen update function
=>	undef	undef

Performs update automatically.

F.3.225 PMLIB::UpdateCheck::ISO_15924**Functions Reference****get_candidates_for_update_iso15924 (named)**

path	str	path to ISO 15924
=>	lref	list of all sources to update

check_updates_iso15924 (named)

list	lref	list of of files to check
=>	lref	those that could be updated

We have 2 sources of information: Wikipedia & CIA factbook. Wikipedia is crawled online, factbook will be fetched, unzipped and parsed.

print_updates_iso15924 (named)

```
updates  lref    result of check_updates_15924
=>      undef   undef
```

Prints formatted results of update check.

update_iso15924 (named)

```
updates  lref    result of check_updates_pmlib
args     href    hash as get_candidates_for_update_pmlib arguments
auto     bool    without asking?
=>      undef   undef
```

Performs update.

_get_codes_from_unicode (void)

```
=>  undef  undef
```

This functions will parse Unicode website and fetch symbols for scripts.

_get_iso_from_wiki (positional)

```
1  str  path where to store data
=>  undef  undef
```

This function will crawl through Wikipedia and fetch pages for given 15924 codes. We mine l10n from these pages.

_update_l10n (named)

```
source  str    web page in HTML
iso_hash href   container hash ref
code    str    iso code processed
=>      undef  undef
```

Fills container hash with l10n.

F.3.226 PMLIB::UpdateCheck::ISO_3166

Functions Reference

get_candidates_for_update_iso3166 (named)

```
path     str    path to ISO 3166
=>      lref    list of all sources to update
```

check_updates_iso3166 (named)

```
list     lref    list of of files to check
=>      lref    those that could be updated
```

We have 2 sources of information: Wikipedia & CIA factbook. Wikipedia is crawled online, factbook will be fetched, unzipped and parsed.

print_updates_iso3166 (named)

```
updates  lref    result of check_updates_3166
=>      undef   undef
```

Prints formatted results of update check.

update_iso3166 (named)

```
updates  lref  result of check_updates_pmlib
args     href  hash as get_candidates_for_update_pmlib arguments
auto     bool  without asking?
=>      undef undef
```

Performs update.

_get_info_from_factbook (positional)

```
1  str  path where to store data
=>  undef undef
```

This functions will parse CIA factbook and get info like STANAG, GEC, borders, areas, languages etc.

_get_iso_from_wiki (positional)

```
1  str  path where to store data
=>  undef undef
```

This function will crawl through Wikipedia and fetch pages for given 3166-1 alpha2 codes. We mine l10n from these pages.

_get_3166_2 (positional)

```
1  str  link to fetch
2  href hash reference - container
=>  undef undef
```

Crawls through wikipedia and mines l10n for 3166-2 codes.

_update_l10n (named)

```
source  str  web page in HTML
iso_hash href  container hash ref
3166    str  iso code processed
=>      undef undef
```

Fills container hash with l10n.

F.3.227 PMLIB::UpdateCheck::ISO_4217

Functions Reference

get_candidates_for_update_iso4217 (named)

```
path  str  path to ISO 4217
=>    lref  list of all sources to update
```

check_updates_iso4217 (named)

```
list  lref  list of of files to check
=>    lref  those that could be updated
```

We have 2 sources of information: Wikipedia & CIA factbook. Wikipedia is crawled online, factbook will be fetched, unzipped and parsed.

print_updates_iso4217 (named)

```
updates  lref  result of check_updates_4217
=>      undef  undef
```

Prints formatted results of update check.

update_iso4217 (named)

```
updates  lref  result of check_updates_pmlib
args     href  hash as get_candidates_for_update_pmlib arguments
auto     bool  without asking?
=>      undef  undef
```

Performs update.

_get_symbols_from_xe (void)

```
=>  undef  undef
```

This functions will parse XE website and fetch symbols for currencies.

_get_iso_from_wiki (positional)

```
1  str  path where to store data
=>  undef  undef
```

This function will crawl through Wikipedia and fetch pages for given 4217 codes. We mine l10n from these pages.

_update_l10n (named)

```
source  str  web page in HTML
iso_hash href  container hash ref
code    str  iso code processed
=>      undef  undef
```

Fills container hash with l10n.

F.3.228 PMLIB::UpdateCheck::ISO_639_3

Functions Reference

get_candidates_for_update_iso6393 (named)

```
path     str  tab files dir. (def.: $ENV{PMLIB_INC}/PMLIB/ISO/639)
=>      lref  list of all .tab files (except for links)
```

check_updates_iso6393 (named)

```
list     lref  list of .tab files to check
=>      lref  those that could be updated
```

First, we filter .tab files that aren't up to date. Checks installed and available versions. Returns a list of lists of the form [name, installed, available]. Each element is information about one package.

print_updates_iso6393 (named)

```
updates  lref  result of check_updates_iso6393
=>      undef undef
```

Prints formatted results of update check.

update_iso6393 (named)

```
updates  lref  result of check_updates_iso6393
args     href  hash as get_candidates_for_update_iso6393 arguments
auto     bool  without asking?
=>      undef undef
```

Performs update.

F.3.229 PMLIB::UpdateCheck::PMLIB

Functions Reference

get_candidates_for_update_pmlib (named)

```
path     str   path to PMLIB (default: $ENV{PMLIB_INC})
=>      lref  list of all packages defined in PMLIB
```

check_updates_pmlib (named)

```
list     lref  list of packages to check
=>      lref  those that could be updated
```

First, we filter packages that aren't up to date. Checks installed and available versions. Returns a list of lists of the form [name, installed, available]. Each element is information about one package.

print_updates_pmlib (named)

```
updates  lref  result of check_updates_pmlib
=>      undef undef
```

Prints formatted results of update check.

update_pmlib (named)

```
updates  lref  result of check_updates_pmlib
args     href  hash as get_candidates_for_update_pmlib arguments
auto     bool  without asking?
=>      undef undef
```

Performs update.

F.3.230 PMLIB::Wiki::Helper

PetaMem Library: Wiki Helpers. Functionality for handling Wikis.

Functions Reference**ensure_wiki_xml (positional)**

```
1  str  path to wikimedia archive (either xml or xml.bz2)
=> lref 0:success status 1:path to wikimedia archive (guaranteed xml)
```

Taken from the P_dm_wiki script, this simple routine makes sure that the file given will be available as uncompresses XML for further processing. It returns an array with index 0 being a success status (1= success, 0=fail) and index 1 being a string. In case of success, the string is the path. In case of failure, it is the error message.

If an XML has been given already, return just that. If a compressed XML has been given (bzip2 is standard), unpack it to some temporary file and return that path.

F.3.231 PMLIB::Wiki::Parser

PetaMem Library: Wikimedia Parser

Functions Reference**wm_parser (void)**

```
=> obj  P:RD object
```

Create a P::RD Object for parsing MediaWikis.

wm_wiktionary_header_parser (void)

```
=> obj  P:RD object
```

Create a P::RD Object for parsing MediaWikis.

add_attr (positional)

```
1  xref  data or data structure
2  list  list of attributes to add
=> href  data structure with data and attributes
```

flatten (positional)

```
1  lref  listref (of listrefs) to flatten
=> lref  flat listref
```

F.3.232 PMLIB::f2f

PetaMem Library: File -> File functionality This documentation covers only the reference for the respective methods. If you want more examples, HowTos and sample code, please see the PMLIB Cookbook POD.

Functions Reference

f2f_get_clean_abspath (positional)

```
1   str   path
=>  str   absolute path with cleaned '/'s
```

Get cleaned up absolute file path. The file may actually exist or not. If undef is given, the empty string is returned.

f2f_trie2dir (positional)

```
1   str   d/i/r/e/c/t/o/r/y as trie
=>  str   directory (without '/'s)
```

Trie directory hierarchy given, return string. E.g. input 'a/b/c' -> output 'abc'

f2f_get_n_files_age (named)

```
dir   str   path
glob  glob  glob to match files
num   int   number of files to perform operation on, default: 1
code  cref  coderef what to do with files, default: remove
=>    undef
```

Returns the <num> newest (if positive) or oldest (if negative) files in a given directory. Default operation is just returning the filename, but a coderef can be given.

F.3.233 PMLIB::f2h

PetaMem Library: File -> Hash functionality This documentation covers only the reference for the respective methods. If you want more examples, HowTos and sample code, please see the PMLIB Cookbook POD.

Functions Reference

f2h_dir2attributes_hr (named)

```
dir   str   directory to read from (default current directory)
glob  str   glob to match files to read in (def: all non-dot files)
=>    href  attributes for files in C<dir>
```

Reads in files in a given directory and stores their attributes as returned by `stat` as values in a hash (keys being the file names).

The return value is a reference to a hash of filenames (keys) and file attributes (values).

f2h_dir2contents_hr (named)

```
dir   str   directory to read from (default current directory)
glob  str   glob to match files to read in (def all non-dot files)
enc   str   encoding to assume for files (default :raw)
filter rx   which files should be omitted
code  cref  returns file content
=>    href  hash of the form file => content
```

Reads in files in a given directory and stores their contents as values in a hash (keys being the file names).

f2h_get_words (positional)

```

1  lref  globs
2  rx    delimiter
3  int   optional: size limit to read (then skip)
4  href  optional: args for tokenize
=> href  set of tokens

```

Parse all files matching all globs, tokenize it and return tokens set.

F.3.234 PMLIB::f2l

PetaMem Library: File -> List functionality

Functions Reference

f2l_get_names_from_dir (named)

```

path      str    directory name where the search begins
create    bool   create C<path> if not exists
glob      str    glob pattern
filter    rx     skip files matching regex
map       cref   map function specification
recurse   bool   recursive flag
stype     enum   sort type (see l2l_sort_filelist for details)
trim      bool   trim flag
=>        lref   matching file names

```

Read directory listing to list and return list of relative filenames. These names are shortened by the path. So, don't forget to concatenate them if you need e.g. to get the absolute path:

```

../test/test1/file_A    is the original path
path => 'test/'         we start the search at 'test' dir
[test1/file_A]          will be returned

```

Glob patterns tip:

```

'*'    match only regular files
'.*'   match also hidden files
'.* *' will match both

```

f2l_file2list (named)

```

file      str|fh  filename or filehandle
enc       str    encoding (default ':raw')
preread   cref   perform before read (default: no-op)
perline   cref   perform per line read (default: no-op)
postread  cref   perform after read (default: no-op)
=>        lref   built array

```

```
=>          val      [] if not
```

Save data from file into the list.

All coderefs are expected to return the reference to some list that will be further processed. `preread` and `postread` get a reference to the current list as 1st argument, `perline` gets 3 arguments:

```
1   lref   current list
2   int    current index in list
3   scalar list element at the current index
```

Be careful if you manipulate the list length, to maintain a correct index!

f2l_file2tokens (positional)

```
1   str    filename
=>  lref   tokens list
```

read file 1 and return list of tokens.

F.3.235 PMLIB::f2s

PetaMem Library: File -> Scalar functionality

Functions Reference

f2s_file2scalar (positional)

```
1   str    filename
2   str    mode (default: ':utf8')
=>  sref   content of the file
```

Reads file.

f2s_files2scalar (positional)

```
1   lref   filenames
2   str    mode (default: ':utf8')
=>  sref   content of the file
```

Reads multiple files.

F.3.236 PMLIB::f2x

File -> (M)any functionality

Functions Reference

f2x_file2eval (named)

```
enc      str    encoding (default ':utf8')
file     str    filename
preeval  cref   code on content before eval (default no-op)
```

```

posteval cref code on return value before return (default no-op)
=>         lref processed data

```

Read in a file and perl-evaluate its contents (i.e. interpret it as Perl code).

preeval gets content of file as parameter.

If specified, `posteval` gets `listref` to result of `eval` and returned value rewrites return value of the whole `f2s_file2eval`.

If evaluation went ok, the result is returned. In case of an error, the subroutine croaks with the contents of `$@`. All other documentation from perl-builtin 'eval' applies.

F.3.237 PMLIB::h2f

PetaMem Library: Hash -> File functionality

Functions Reference

h2f_hash2file (named)

```

hash   lref   hash to be stored
path   path   filename
force  bool   to save even empty hash
out    enum   keys, values, both (default)
sort   enum   keys, values, none (default)
=>     str    filename

```

Takes `hash` and add content of all elements into specified file

h2f_process_files (named)

```

proc      href      hash of the form rx => cref
<special> <special> + args as f2l_get_names_from_dir
=>        lref      matched files

```

Process directory by a set of regex => coderef rules. For selecting files and its named args see `f2l_get_names_from_dir`.

F.3.238 PMLIB::h2h

PetaMem Library: Hash -> Hash functionality

Functions Reference

h2h_copy_hash (positional)

```

1      href   hash
=>     href   copy of C<1>

```

Copy a hash `1` into another hash and return reference to the copy. We need this in case `dclone/clone` fails on the hash in question, which seems to be the case for tied hashes, especially `%+`. The latter being the reason for implementing this.

h2h_cut_hash (named)

hash	href	hash of the form key => number
upper	int	num of upper values to return (default: C<lower> or 1)
lower	int	num of lower values to return (default: C<upper> or 1)
force	bool	allow to return less hash if true
=>	href	hash splitted into 2 sets

Takes `hash` and returns `href` with keys `upper` and `lower`. Value of `upper` is hashref of `upper` pairs from `hash` with biggest value. Analogically, value of `lower` is hashref of `lower` pairs from `hash` with the smallest value.

If `hash` hasn't enough keys, `undef` or whole `hash` is returned depending on `force`.

h2h_cut_hash_simple (named)

hash	href	hash of the form key => number
from	int	from which index
to	int	to which index
from_inc	str	include start pos.? (def: see <code>s2s_get_positions</code>)
to_inc	str	include end pos.? (def: see <code>s2s_get_positions</code>)
=>	href	hash

Sorts `hash` according to values and cut a hash slice.

`from` and `to` could be of the form like in `s2s_get_positions`.

h2h_cut_histogram (named)

histogram	href	hash of the form key => number
upper	real[0>1]	what upper fraction will be cut (default: 1)
lower	real[0>1]	what lower fraction will be cut (default: 0)
brief	bool	should return just cut?
=>	href	only rest of the C<histogram> if C<brief>
=>	href	all parts separated to groups unless C<brief>

Takes a `histogram` (which is a hash of the form `key => number`) and two bounds (`lower` and `upper`, decimals between 0 and 1 whose sum is ≤ 1). Sorts list of keys of the histogram (according to values from histogram) and separates it into 3 lists:

upper	- last keys in the list whose sum of values (from histogram) closely exceeds 'upper bound' fraction of the histogram sum
lower	- first keys in the list whose sum of values (from histogram) closely exceeds 'lower bound' fraction of the histogram sum
mid	- resting keys

It also returns 'sums' of the each sublist as following keys:

upper_count
lower_count
mid_count

h2h_get_keywords (named)

corpus	href	corpus frequencies
--------	------	--------------------

```
reference href reference corpus frequencies
=> href keywords
```

reference is usually some big and balanced corpus. corpus could be smaller and we get information about relevance of individual keys for this corpus. Values are kind of weights. Highest and lowest values for token/ngram/etc. are characterizing this corpus.

h2h_get_rank (positional)

```
1 href scores
=> href ranks
```

Sorts hash according to values (scores).

h2h_get_ranks (positional)

```
1 href scores
=> href ranks
```

Sorts hash according to multiple scores. Values are hashes, whose keys are score names.

h2h_merge_hashes (named)

```
h1 href hash
h2 href hash
=> href merged hash
```

Returns merged hash. Values of first hash are preferred.

h2h_sum_hashes (named)

```
h1 href hash of the form key => number
h2 href hash of the form key => number
=> href hash sum
```

Resulting hash has keys from both given h1 and h2 with values as sums.

F.3.239 PMLIB::h2l

PetaMem Library: Hash -> List functionality This documentation covers only the reference for the respective methods. If you want more examples, HowTos and sample code, please see the PMLIB Cookbook POD.

Functions Reference

hr2lr_ify (positional)

```
1 href hash
=> lref list
```

Is considered to be the "inverse" operation of x2hr_ify (x2h.pm) and assumes, value of the given href is a number. If it is not a number, the element is written just once to the listref.

Remark: We could add more semantics to this operation, e.g. `value=0 => skip key`, `value<0 => write undef x abs(value) into the list`.

F.3.240 PMLIB::h2s

PetaMem Library: Hash -> Scalar functionality

Functions Reference

h2s_arrange_kv (named)

```

hash    href    data to format
fill    char    to pad
kalign  enum    key alignment (default: l)
kfill   char    to pad key
vfill   char    to pad value
sort    enum    sorting flag (default: none)
gap     str     gap string
out     enum    what to output (default: both)
prefill str    prefill string
filter  rx     filter keys
=>     str     formatted hash

```

Format the key/values of a hash for subsequent printing.

Key alignment - possible values: see `s2s_pad align`. Sorting flag - possible values: none keys. What to output - possible values: keys values both.

h2s_synthetic_text (named)

```

tokens  int     tokens in the text (default: 100)
tknlen  int     length of tokens (default: 4)
chars   lref    allowed characters for a token (default: [a-z])
delim   str     delimiter to use between the tokens (default: ' ')
=>     str     synthetic text

```

Create a synthetic text according to given number of tokens `tokens`, token length `tknlen` and allowed token characters `chars`. Words are separated by `delim`.

h2s_write_synthetic_text (named)

```

out     str|-|undef filename|STDOUT|return value
delim   str          how to delimit the texts themself (default: ' ')
txts    lref         arg. hashes for low-level synthetic text creation
=>     str          created text if C<out>=undef
=>     undef        else

```

Assemble combined synthetic text from various simple synthetic texts and write to `file/stdout/return`.

if `'-'` is given as parameter for the `'out'` argument, the created text is printed to `STDOUT`. If any other string is given, it is interpreted as path/filename where to store created

synthetic text. If this argument is left undef (default), the created text is the return value.

F.3.241 PMLIB::h2x

PetaMem Library: Hash -> (M)any functionality

Functions Reference

h2x_get_contingency_table (named)

ngrams	href	set of ngrams
tokens	href	tokens occurrences
ngram	lref	for which ngrams to create contingency table
separ	str	separator of tokens in n-gram
=>	lref	contingency table for C<ngram>

Position in contingency table is binary index of the resulting list.

h2x_build_true_marginals_data_struct (named)

ngrams	href	set of ngrams
n	int	size of 'n'grams
tokens	href	tokens occurrences
marginal_char	str	character representing 'any token' (default: *)
=>	lref	true-marginals table

Position of true-marginal is binary index of the list where 1 means *.

F.3.242 PMLIB::l2f

PetaMem Library: List -> File functionality; with enrichment from various CPAN modules.

Functions Reference

l2f_list2file (named)

enc	str	encoding flag (default: ':encoding(UTF-8)')
list	lref	list to process
file	path	filename to write content
mode	str	open mode, (default: '>>')
=>	bool	indicate success

Takes list and add content of all elements into specified file

F.3.243 PMLIB::l2h

PetaMem Library: List -> Hash functionality

Functions Reference

l2h_do_set (positional)

```
1  lref  list
2  href  variable to be constructed (default: {})
=> href  set
```

Transforms a list 1 into a set 2, thus uniquifying the elements of the former list. A possible application is to get term counts of a document: Tokenize text, feed resulting list to `do_set`. You get a hash with term counts for this document. To obtain term frequencies see function `term_frequency`.

l2h_build_nested_hash (positional)

```
1  lref    some list
2  str     fill-value (default: undef)
=> href    nested hash
```

Will create a nested hash from a list reference. `(a b c d)` will become `{ a = { b => { c => d }}}`. If an empty list is given an empty hashref is returned. In the special case of just one element, depending on the value of 2 argument either just this element is returned (if `undef`) or a hashref with the element being the key and fill-value being the value is returned.

l2h_get_ngrams (named)

```
delim  str    token separator
size   int    ngram size
tokens lref   tokenized string
window int    window size (default: C<size>)
=>     href   set of ngrams
```

Constructs hash of ngrams from tokens list. Tokens in ngram are separated with `delim`.

Example. Params:

```
tokens = A B C D E F
size   = 2
window = 4
```

Processing:

```
first window: ABCD
ngrams: AB AC AD BC BD CD

second window: BCDE
ngrams: BC BD BE CD CE DE -> BC BD dupes

third window: CDEF
ngrams: CD CE CF DE DF EF -> CD CE dupes
```

Result: all uniq ngrams

l2h_get_sublists_by_type (positional)

```

1   lref   list of various types elements
=>  href   hash of the form type => [elems of C<1>]

```

This will examine all elements of the given list for their type (Scalar, ARRAY, HASH, Regexp, ...) and add them to the appropriate sublist in a hash, where the key is the type and the value is a listref of all elements of that type.

l2h_substrings_common_all (named)

```

strings lref   list of strings to evaluate
=>      href   hash of substrings

```

Given a list reference with strings to evaluate, this subroutine returns a hash reference with keys being the common substrings found and corresponding values the number of their occurrence. The named arguments given are

l2h_substrings_common_longest (named)

```

strings lref   list of strings to evaluate
=>      href   longest substrings

```

Given a list reference with strings to evaluate, this subroutine returns a hash reference with keys being the LONGEST common substrings and corresponding values the number of their occurrence. The named arguments given are.

This subroutine uses `l2h_substrings_common_all`.

l2h_strings2trie (positional)

```

1   lref   list of strings
=>  href   trie containing all given strings

```

l2h_tablist2hr (named)

```

delim rx   delimiter (see s2l_tab2kv)
func  cref mapping function (see s2l_tab2kv)
lines lref  list of lines
=>  href   hash of the form word1 => [word2 word3 ...]

```

Given a listref of lines with tab/delimiter-separated values, create a hashref. `s2l_tab2kv` is used to process each line. By default, keys are first substring, values are listrefs containing rest of the line, but this behaviour can be changed by the coderef given.

F.3.244 PMLIB::l2l

PetaMem Library: List -> List functionality

Functions Reference

l2l_after (positional)

```

1   cref   coderef (BLOCK) to be applied
2   lref   values
=>  list   results

```

Returns a list of the values of LIST after (and including) the point where BLOCK returns a true value. Sets \$_ for each element in LIST in turn.

```
l2l_after(sub { $_ % 5 == 0 }, (1 .. 9));
returns (5,6,7,8,9)
```

l2l_align (named)

```
l1   lref   first listref to compare
l2   lref   second listref to compare
fill str   filler of the missing elements in the list ???
=>   lref   empty aref if no input given
```

!!! THIS FUNCTION IS NOT IMPLEMENTED

l2l_before (positional)

```
1   cref   coderef (BLOCK) to be applied
2   lref   values
=>  lref   results
```

Returns a list of values of LIST upto (and including) the point where BLOCK returns a true value. Sets \$_ for each element in LIST in turn.

```
l2l_before(sub { $_ % 5 == 0 }, (1 .. 9));
returns (1,2,3,4,5)
```

l2l_block_pass_filter (named)

```
block_filter str   a regex, defines non-wanted tokens
inlist       lref   complete list of tokens
pass_filter  str   a regex, defines wanted tokens
=>           undef  if no input arguments given
=>           lref   original list if no block/pass filter
=>           lref   filtered list
```

block_filter is a regex, that defines elements you want to filter out from the original list

inlist is a list of tokens

pass_filter is a regex, that defines elements you want to extract from the original list

If both block_filter and pass_filter are defined and if both regular expressions define the same token, then the block filter has higher priority and the common element won't be present in the resulting list.

l2l_cmap (positional)

```
1   cref   coderef (BLOCK) to be applied
2   lref   values
=>  list   results within a list context
=>  scalar a single value within a scalar context
```

Applies BLOCK to each item in LIST and returns a list of the values after BLOCK has been applied. In scalar context, the last element is returned. This function is similar

to map but will not modify the elements of the input list:

```
my @list = (1 .. 4);
my @mult = l2l_cmap( sub { $_ *= 2 }, @list);
print "\@list = \@list\n";
print "\@mult = \@mult\n";
```

Will print:

```
@list = 1 2 3 4
@mult = 2 4 6 8
```

Think of it as syntactic sugar for

```
for (my @mult = @list) { $_ *= 2 }
```

l2l_compress_whitespaces_lr (positional)

```
1   lref  listref containing strings to format
=>  val   [] if no input argument given
=>  lref  formatted strings
```

Replaces all inline multi-spaces with just one space. This function uses **s2s_compress_whitespaces**. See its documentation for more details.

l2l_extract_from_parallel (named)

```
l1   lref  first list reference
l2   lref  second list reference
l3   lref  third list reference
=>  val   [] if no arguments given
=>  val   [] if one of input parameters doesn't exist
=>  lref  with matching elements
```

Extracts from a list(L2) those elements of list(L3) that match elements in L1 and return as a list reference.

l2l_filter_substrings (named)

```
return  str    string defines type of output
strings lref   list of strings
=>      val    [] if no 'strings' argument is given
=>      lref   superstrings/substrings
```

return 'super' (default) for superstrings, anything else for substrings

strings listref to list of strings

Given a list of strings, remove the substrings. That is, the strings that are contained in some other of the list elements. The return value can be set to give either the superstrings (default) or the substrings.

```
strings: ['ABC', 'QX', 'BC', 'X']
=> ['ABC', 'QX'] if return = 'super'
=> ['BC', 'X']   if return != 'super'
```

l2l_igrep (positional)

```

1  cref  coderef (BLOCK) to be applied
2  list  values
=> list  results

```

Evaluates BLOCK for each element in LIST (assigned to \$_) and returns a list of the indices of those elements for which BLOCK returned a true value. This is just like `grep` only that it returns indices instead of values:

```
my @indices = igrep(sub { $_ eq 'cure' }, qw(cure queen));
```

returns

```
@indices = (0)
```

l2l_insert_after (positional)

```

1  cref      coderef (BLOCK) to be applied
2  scalar    value to be inserted
3  lref      list in which you want to insert the value
=> val      0

```

Inserts VALUE after the first item in LIST for which the criterion in BLOCK is true. Sets \$_ for each item in LIST in turn.

```
my @list = qw(cure queen);
l2l_insert_after(sub { $_ eq "cure" }, "led zeppelin", \@list);
```

will modify the original list to:

```
@list = ('cure', 'led zeppelin', 'queen')
```

l2l_is_alignable (named)

```

l1      lref  1st listref to compare
l2      lref  2nd listref to compare
cmp      cref  code reference that would be applied on both lists
restyp  str   'full' (default) or 'simple'
=>      str   if restyp 'simple' is in use
=>      lref  if restyp 'full' is in use

```

`cmp` the default is `c<sub { shift() eq shift() }>`. It compares the members of the arrays references.

Results for 'simple' restyp:

```

'L1!L2' L1 and L2 are unalignable           (e.g.  A.B <-> B.A)
'L1=L2' L1 and L2 are identical             (e.g.  A.B <-> A.B)
'L1<L2' L1 is alignable within L2          (e.g.  A.B <-> A.B.C)
'L1>L2' L2 is alignable within L1          (e.g.  A.B.C <-> A.B)
'L1/L2' L1 and L2 are out of bounds alignable (e.g.  A.B <-> X.Y)
'?c1'   there was an element in list2 that has no place in list1
'?c2'   there was an element in list1 that has no place in list2
'???'   shouldn't occur

```

Result for 'full' restyp are 2 hashes; values ore orders of keys in other array. So, if you have two lists: `['good','afternoon','sir']` `['good','afternoon']`

Then you will get:

```
[
  {'good' => [0], 'afternoon' => [1], 'sir' => []},
  {'good' => [0], 'afternoon' => [1]}
]
```

What indicates, that 'sir' is not a member of the opposite arrayref. 'Good' and 'afternoon' have the same position in both arrayrefs, thus they have the same indices.

Note: If the restyp would be 'simple', the returned value should be 'L1>L2' but it is '???' - why?

l2l_mesh (positional)

```
1 list list of listrefs with element
=> list meshed list
```

Returns a list consisting of the first elements of each array, then the second, then the third, etc, until all arrays are exhausted.

Examples:

```
my $v1 = ['a', 'b', 'c'];
my $v2 = [1,2,3];

my @mesh = l2l_mesh($v1,$v2);
```

Will return:

```
@mesh = ('a',1,'b',2,'c',3);
```

If input listrefefs are not the same length:

```
$v1 = ['x'];
$v2 = ['1', '2'];
$v3 = [qw(zip zap zot)];
@mesh = l2l_mesh($v1,$v2,$v3);
```

Will return:

```
@mesh = (x, 1, zip, undef, 2, zap, undef, undef, zot);
```

l2l_min_index (positional)

```
1 list numeric values
=> lref indices
```

This function returns a listref with index of the lowest numeric value for an element in the input array. If exist multiple elements with the same (lowest) value in the input array, then multiple indices are returned.

```
input array: (5, 50, 20, 1)
=> [3]

input array: (5, 50, 1, 20, 54, 1)
=> [2,5]
```

l2l_rfilter (named)

```

code cref  code to apply
list lref  values to filter
=> href    empty href if no input arguments given
=> lref     empty listref if no 'list' argument
=> lref     original listref if no 'code' argument given
=> lref     filtered listref

```

Works similar like the builtin "grep", but can process a list recursively (that is, list-of-lists...). Only elements that match the criteria defined in the do-block will be passed to the return list.

Simply data structure:

```

input list: ['my', 'yours', 'our', 'Mike']
code:       sub { shift =~ m{\bm.+\\b}xmsi }
=>         ['my', 'Mike']

```

List of lists:

```

input list: ['my', 'yours', 'our', 'Mike', ['Marina', 'Cvetajevova'],]
code:       sub { shift =~ m{\bm.+\\b}xmsi }
=>         ['my', 'Mike', ['Marina']]

```

l2l_rflatten (positional)

```

1 lref given data structure
=> lref flattened listref

```

1 Input list reference can be anything from an empty list, to a hierarchical list of lists. The elements can be mixed normal values and list references.

Recursively flatten a list (of lists). Result is a reference to a simple (flat) list.

```

input listref: ['a', ['b', 'c', [1..3]], 'd']
=>            ['a', 'b', 'c', 1, 2, 3, 'd']

```

l2l_shuffle (positional)

```

1 list elements to shuffle
=> undef if the input array has no elements
=> list list with shuffled elements

```

Returns randomly shuffled list.

l2l_sort_filelist (named)

```

files lref  list of files
stype enum  sort type
=> val     [] if no input arguments given
=> lref    sorted list

```

sort list of files according to given sort type:

```

+alpha
+size

```

```
+time
-alpha
-size
-time
```

Sort type may be given case insensitive, + means ascending order, - means descending order. If a '?' is given the function returns a listref of accepted/known sort types. If an unknown sort type is given (including no sort type given) +alpha is assumed.

l2l_sum_lrs (positional)

```
1  lref  first 2d list of lists
2  lref  second 2d list of lists
=> undef if C<1> or C<2> not given
=> lref  results
```

Sums 2dimensional arrays:

```
listref1: [[1,2,]]
listref2: [[3,4,]]
=>        [[4,6,]]
```

l2l_trim_whitespaces_lr (positional)

```
1  lref  strings to format
=> val  [] if no input arguments given
=> lref  formatted strings
```

Removes whitespace characters (space, tab) at the beginning and at the end of a given string. Whitespaces - even multiple - within the string are ignored.

l2l_uniq (positional)

```
1  list  list of values
=> list  list of unique values
```

Returns a new list by stripping duplicate values in LIST. The order of elements in the returned list is the same as in LIST. In scalar context, returns the number of unique elements in LIST.

```
my @x = l2l_uniq( qw(1 1 2 2 3 5 3 4) );
```

returns

```
@x = (1,2,3,5,4,);
```

if

```
my $x = l2l_uniq( qw(1 1 2 2 3 5 3 4) );
```

returns

```
$x = 5;
```

l2l_uniq_generic (positional)

```
1  cref  stringification
2  list  list of values
```

```
=> list list of unique values
```

Works like `l2l_uniq` but applies stringification before the `uniq` process.

Former elements have higher priority.

F.3.245 PMLIB::l2s

PetaMem Library: List -> Scalar functionality; with enrichment from various CPAN modules.

Functions Reference

l2s_any (positional)

```
1 block true/false sub
2 list list
=> bool C<block> true for any value
```

Checks if `block` returns true for any value from `list`.

l2s_all (positional)

```
1 block true/false sub
2 list list
=> bool C<block> true for all values
```

Checks if `block` returns true for all values from `list`.

l2s_false (positional)

```
1 block true/false sub
2 list list
=> int number of falses
```

Counts the number of elements in `list` for which the criterion in `block` is false. Sets `$_` for each item in `list` in turn.

l2s_firstidx (positional)

```
1 block true/false sub
2 list list
=> int index with value satisfying the criterion
=> val -1 if no such index found
```

Returns the index of the first element in `list` for which the criterion in `block` is true. Sets `$_` for each item in `list` in turn.

l2s_firstval (positional)

```
1 block true/false sub
2 list list
=> int index with value satisfying the criterion
=> undef if no such element found
```

Returns the first element in `list` for which `block` evaluates to true. Each element of `list` is set to `$_` in turn.

`l2s_lastidx` (positional)

```
1  block true/false sub
2  list  list
=> int   index with value satisfying the criterion
=> val   -1 if no such index found
```

Returns the index of the last element in `list` for which the criterion in `block` is true. Sets `$_` for each item in `list` in turn.

`l2s_lastval` (positional)

```
1  block true/false sub
2  list  list
=> int   index with value satisfying the criterion
=> undef if no such element found
```

Returns the last element in `list` for which `block` evaluates to true. Each element of `list` is set to `$_` in turn.

`l2s_minmax` (positional)

```
1  list  list
=> list  min, max
=> val   () if C<list> is empty
```

Calculates the minimum and maximum of `list` and returns a two element list with the first element being the minimum and the second the maximum. Returns the empty list if `LIST` was empty.

The `l2s_minmax` algorithm differs from a naive iteration over the list where each element is compared to two values being the so far calculated min and max value in that it only requires $3n/2 - 2$ comparisons. Thus it is the most efficient possible algorithm.

However, the Perl implementation of it has some overhead simply due to the fact that there are more lines of Perl code involved. Therefore, `list` needs to be fairly big in order for `l2s_minmax` to win over a naive implementation. This limitation does not apply to the XS version (which is not implemented here).

`l2s_none` (positional)

```
1  block true/false sub
2  list  list
=> bool  false for all?
```

Checks if `block` returns false for all values from `list`.

`l2s_notall` (positional)

```
1  block true/false sub
2  list  list
=> bool  false for some?
```

Checks if `block` returns false for some value from `list`.

l2s_reduce (positional)

```
1  block  sub
2  list  list
=> other  result
```

Reduces `list` by calling `block`, in a scalar context, multiple times, setting `$a` and `$b` each time. The first call will be with `$a` and `$b` set to the first two elements of the list, subsequent calls will be done by setting `$a` to the result of the previous call and `$b` to the next element in the list.

Returns the result of the last call to `block`. If `list` is empty then `undef` is returned. If `list` only contains one element then that element is returned and `block` is not executed.

```
$foo = l2s_reduce { $_[0] < $_[1] ?
                  $_[0] : $_[1] } 1..10      # min

$foo = l2s_reduce { $_[0] lt $_[1] ?
                  $_[0] : $_[1] } 'aa'..'zz' # minstr

$foo = l2s_reduce { $_[0] + $_[1] } 1 .. 10  # sum

$foo = l2s_reduce { $_[0] . $_[1] } @bar    # concat
```

l2s_sum (positional)

```
1  list  list of strings
=> real  sum of all elements in C<list>
```

l2s_product (positional)

```
1  list  list of strings
=> real  product of all elements in C<list>
```

l2s_true (positional)

```
1  block  true/false sub
2  list  list
=> int   number of falses
```

Counts the number of elements in `list` for which the criterion in `block` is true. Sets `$_` for each item in `list` in turn.

l2s_max (positional)

```
1  list  list of numbers
=> real  numeric maximum of the list
```

l2s_maxstr (positional)

```
1  list  list of strings
=> string string maximum of the list
```

l2s_min (positional)

```

1 list list of numbers
=> real numeric minimum of the list

```

l2s_minstr (positional)

```

1 list list of strings
=> str string minimum of the list

```

l2s_min2d (positional)

```

1 lref list of lists
2 lref list of active indices
=> list min_value, min_i_coord, min_j_coord

```

Looks at all values in 2dimensional list 1 whose coordinates are from values in <2> and find the minimum and its coordinates.

l2s_rmax (positional)

```

1 lref list
2 href options
=> other maximum recursively

```

In 2 there could be following options:

```

mode      enum  mode of operation
copy      bool  copy mode flag
recurse   bool  recurse flag
defval    other default value for noncompatible values

```

This routine takes a list reference (can be list of lists) traverses it recursively and returns the "maximum value" in this list. As the list can contain various elements, the semantic "maximum" will differ from type to type. Therefore the named arguments hash contains a parameter `mode` which determines the behaviour of this subroutine. If no `c<mode>` is given (or no args hash whatsoever), the default behaviour is 'val' which means to return the numeric maximum value. See the following table for the various modes of operation and related return values:

```

val  element with maximum (positive) integer value - DEFAULT
str  the alphabetically last element
len  the longest element
aval autovalue: tries to determine list type and returns val/str

```

l2s_rmaxlen (positional)

```

1 lref list
2 href options
=> int maximum length recursively

```

In 2 there could be following options:

```

recurse bool  recurse flag
defval  other default value for noncompatible values

```

l2s_rmaxstr (positional)

```

1  lref    list
2  href    options
=> int     string maximum recursively

```

In 2 there could be following options:

```

recurse bool    recurse flag
defval  other   default value for noncompatible values

```

l2s_rmaxval (positional)

```

1  lref    list
2  href    options
=> int     value maximum recursively

```

In 2 there could be following options:

```

recurse bool    recurse flag
defval  other   default value for noncompatible values

```

l2s_rsum (positional)

```

1  lref    list (multidimensional)
=> int     recursive sum

```

Computes recursively the sum of all elements in the given list (of lists). Uncomputable elements will be added with a default value.

l2s_rtypecheck (positional)

```

1  list    list (multidimensional)
=> int     balance

```

Checks recursively (list of lists) the type of the elements encountered. A `balance` value is returned which is 0 if there were none elements or as much numbers as strings, a negative value if there were more numbers than strings and a positive value if there were more strings than numbers.

l2s_in (positional)

```

1  str     whatever string
2  list    list
=> bool    C<1> in C<2>?

```

Checks if `string` is contained in `list`.

l2s_inref (positional)

```

1  str     whatever string
2  lref    list
=> bool    C<1> in C<2>?

```

Checks if `string` is contained in `list`.

l2s_is_set (positional)

```

1  lref    list
=> bool    is C<1> set?

```

Checks if param is set (each element at most once)

l2s_list_choose (named)

```
list      lref   list
ch_mode  enum   mode
=>       other  pick an element from a list
```

Chooses element from a given list according to given criterion:

```
all      get string of all elements separated by newline
N        get element with given index N
cycleN   get element with N added to stored value
rand     get random
randX    get weighted random, X is of the form 'weight1,weight2...'
unknown  whatever else -> returns first element
```

l2s_lists_identic (positional)

```
1  lref   list 1
2  lref   list 2
=> bool   C<1> and C<2> identic?
```

Checks if lists are identic (same positions - same elements).

l2s_lists_equal (positional)

```
1  lref   list 1
2  lref   list 2
=> bool   C<1> and C<2> equal?
```

Checks if sorted lists are identic. Here we don't care about positions (compare to `l2s_lists_identic`)

l2s_occurrences_count (positional)

```
1  other  some element
2  lref   list
=> int    number of occurrences of C<1> in C<2>
```

l2s_visualize_contingency (positional)

```
1  lref      binary contingency table
2  lref      column names
3  str|undef delimiter
=> str|lol   csv text or data struct
```

Returns human readable representation of contingency table.

F.3.246 PMLIB::l2x

PetaMem Library: List -> (M)any functionality

Functions Reference

l2x_reduce_loh (positional)

```
1   lref      list of hashes
=>  lref|href list of merged hashes which were able to merge
```

Given a list of hashes, this method tries to reduce the list to a minimum set of hashes in a way, so unique keys get merged within a hash as soon as possible and duplicate keys are stored in the next possible hash.

F.3.247 PMLIB::s2f

PetaMem Library: Scalar -> File functionality This documentation covers only the reference for the respective methods. If you want more examples, HowTos and sample code, please see the PMLIB Cookbook POD.

Functions Reference

s2f_scalar2file (positional)

```
1   str      file name
2   str      content
=>  bool     indicate success
```

Takes scalar or scalar reference 2 and writes its content into specified file 1.

s2f_append2file (positional)

```
1   str      file name
2   str      content (default: empty string)
3   str      encoding (default: none - thus :raw input layer)
=>  bool     indicate success
```

Takes scalar or scalar reference 2 and appends its content into specified file 1 with the encoding specified in 3 (default :raw only).

F.3.248 PMLIB::s2h

PetaMem Library: Scalar -> Hash functionality This documentation covers only the reference for the respective methods. If you want more examples, HowTos and sample code, please see the PMLIB Cookbook POD.

Functions Reference

F.3.249 PMLIB::s2l

PetaMem Library: Scalar -> List functionality Complementary functions to those found in Perl as builtins like e.g. `split`, `unpack` etc.

Functions Reference

s2l_get_substrings (named)

```

str      str      input string
minl    int      minimum substring length (default: 2)
maxl    int      maximum substring length (default: length of str)
=>      lref     list of substring from a given string

```

Produces a list of substring from a given string. E. g. the string 'perl' with minl => 3 and maxl => 5 will produce these substrings: per, erl, perl.

s2l_split_by_length (positional)

```

1      str      string to be split
2      int      number of characters per slice (default: 1)
=>    lref     slices

```

This function will take a string as 1 and an optional slice length as 2. It chops 1 to slices of length 2 and returns a list reference to these slices.

s2l_tab2kv (named)

```

line    str      input line
func    cref     mapping function (default: [a,b,c] -> (a => [b,c]))
delim   rx       delimiter (default: qr{\t})
=>      lref     mapped/processed input string

```

This function will take a scalar (or scalarref) and interpret it as a visualization of one line of a table. The string will be splitted assuming `delim` being the delimiter. The resulting listref is mapped/processed by the coderef given (or the default code) and returned. Delimiters do not appear in the resulting pair.

s2l_tokenize (named)

```

str      str      input string
pass_filter  rx      matching tokens will pass
block_filter rx      matching tokens will be blocked
=>      lref     tokens from a given string

```

Returns a list reference of tokens from a given string. This list will only contain tokens that satisfied blocking/passing conditions (if such were given).

s2l_wrap (named)

```

str      str      input string to be split
len      int      max row length
separator str|undef rows separator/undef
=>      lref     rows unless separator defined
=>      str      rows joined by separator

```

Split a given string by length taking words length into account. Words are not split nor hyphenated.

s2l_parse_range (positional)

```
1  str    range string (e.g. "1-16", "1,3,6-8", "1,2-5,7")
=> list  sorted list of numbers
```

Parse page-range string syntax into a sorted list of numbers. Supports individual numbers (1,3,7) and ranges (2-5). Comma-separated combinations are allowed (1,3,6-8).

F.3.250 PMLIB::s2s

PetaMem Library: Scalar -> Scalar functionality

Functions Reference

s2s_bin2dec (positional)

```
1  int  binary number to be converted
=> int  decimal number
```

Takes the input (binary) number and returns its decimal value.

s2s_compress_whitespace (positional)

```
1  str  string to get formatted
=> str  formatted string (empty, if no input)
```

Removes superfluous whitespace characters in text. That is, several consecutive whitespace characters are replaced with one single whitespace character and leading and trailing whitespace characters are removed entirely. `s2s_trim_whitespace` is used for the latter.

s2s_dec2bin (positional)

```
1  int  decimal number to be converted
=> int  binary number
```

Takes the input (decimal) number and returns its binary value.

s2s_str2hex (positional)

```
1  oct  octets to be converted into hexadecimal string
=> str  hexadecimal string
```

Takes the input (octets) and unpacks it into a hexadecimal string

s2s_define (positional)

```
1  str  scalar value to be tested
=> str  original string if the input is defined
     str  empty string if the input is undef
```

Takes a single value as input. If the value is defined, it is returned unchanged. If it is not defined, an empty string is returned.

This subroutine is useful for printing when an undef should simply be represented as an empty string. Granted, Perl already treats undefs as empty strings in string context, but this sub makes `-w` happy. And you **ARE** using `-w`, right?

s2s_deformat (positional)

```

1  str    string to get deformatted
=> undef  if no input given
    str    formatted string

```

If the input string does not exist, an undef value is returned.

Removes all formatting information from the string. That is, removes all newlines and multiple spaces and makes a continuous text of it. Uses mainly `s2s_compress_whitespaces` to achieve its job.

s2s_dehyphen (positional)

```

1  str    hyphenated text
=> undef  if no text given
    str    dehyphenated text

```

All hyphenated words are dehyphenated and placed before the line end.

s2s_enclose_folding_marks (named)

```

str          str    enclosed text
fmt_close   str    closing sequence
fmt_open    str    opening sequence
key         str    a remark
=>          undef  if no arguments given
            str    formatted string

```

`str` a text you want to enclose, if not specified, default value `' '` is taken

`fmt_close` the closing sequence, if not specified, default string `# }}}` is used, what corresponds with the closing mark in Emacs 'folding mode' for Perl

`fmt_open` the opening sequence, if not specified, default string `# {{{` is used, what corresponds with opening mark in Emacs 'folding mode' for Perl

`key` is the annotation placed on the line with opening sequence, the default value is `' '`

`s2s_enclose_folding_marks` will return a text string in a following format:

```

<opening sequence> <key>
<text>
<closing sequence>

```

A remark is placed on the line next to the opening sequence.

s2s_equndef (positional)

```

1  str    1st string to be compared
2  str    2nd string to be compared
=> bool   are both strings equal?

```

Returns true if the two given strings are equal. Also returns true if both are undef. If only one is undef, or if they are both defined but different, returns false.

s2s_format_column (positional)

```

1  str  text to format
2  int  width (in characters) of the column
=> undef if no input string is given
    str  deformatted text, if no width specification is given
    str  formatted text (string)

```

If 1 not given, undef value is returned If 2 not given, deformatted text is returned (no line breaks, white-space reduction)

Formats a string by a given max-column. No line in the returned (formatted) string is longer than the given max-column.

s2s_fullchomp (positional)

```

1  str  string to be chomped
=> str  chomped string

```

Works like chomp, but is a little more thorough about removing \n's and \r's even if they aren't part of the OS's standard end-of-line.

Undefs are returned as undefs.

s2s_get_positions (named)

```

from      str      starting position
to        str      end position
size      int      list size
from_inc  str      include starting position (default: 1)
to_inc    str      include end position? (default: 1)
=>        (int,int) absolute (from, to)

```

from and to could be of the form of

```

absolute value (e. g. 40)
percent of the list size (e. g. 40%)
C<to> could be specified as difference to C<from> (e. g. +40, +40%)

```

They are converted into number values.

s2s_hascontent (positional)

```

1  str  string to test
=> val  0 if no input given or the string contains only white-spaces
=> val  1 if string has non-space content

```

Returns true if the given argument contains something besides whitespace.

This function tests if the given value is defined and, if it is, if that defined value contains something besides whitespace.

An undefined value returns false. An empty string returns false. A value containing nothing but whitespace (spaces, tabs, carriage returns, newlines, backspace) returns false. A string containing any other characters (including zero) returns true.

s2s_htmllesc (positional)

```

1  str  string to format
=> str  empty string, if no input (or undef) is given
    str  string contains escaped characters

```

Formats a string for literal output in HTML. An undefined value is returned as an empty string.

`s2s_htmllesc` is very similar to `CGI.pm`'s `escapeHTML`. If your script already loads `CGI.pm`, you may not need `s2s_htmllesc`. However, there are a few differences. `s2s_htmllesc` changes an undefined value to an empty string, whereas `escapeHTML` returns undefs as undefs.

s2s_iec_60027_2 (positional)

```

1  int    size for which we want to get the binary prefix
2  str    sprintf format definition
=> undef  if no input size is given
    str    string with number and prefix

```

Returns a string containing number and binary prefix for a given size. The number is formatted according to the specified sprintf format definition.

1 must exist or undef value is returned 2 if not given, default format definition `'%.1f'` is used

s2s_neundef (positional)

```

1  str    1st string to be compared
2  str    2nd string to be compared
=> bool   are both strings equal?

```

The opposite of `s2s_equndef`, returns true if the two strings are **not** the same.

s2s_nospace (positional)

```

1  str  str to remove the whitespace characters from
=> str  the formatted string

```

If is the input string not defined, returns the original value. Removes all whitespace characters from the given string.

s2s_pad (named)

```

str      str    string to be padded
align    char   alignment indicator
padlen   int    length of pad
padfill  char   character to fill pad with
=>       undef  if no arguments or no C<str> given
=>       str    padded string according to arguments

```

Performs padding on a given string.

`str` is the string that is to be padded. It must exist, or the function returns an undefined value.

`align` is an optional character that indicates the alignment of the string. Valid values for are: 'r' -> right alignment 'l' -> left alignment 'c' -> centered alignment undef -> 'r' is assumed other -> 'c' is assumed

`padlen` the length in characters of the resulting padded string. If not given, the function returns the original string given.

`padfill` is the character to fill the resulting gap (pad) with. It must be of char length 1, or bad things will happen. The function does not check if the argument given has length 1. If none is given, ' ' (whitespace) is assumed default.

`s2s_pad` will prune the string if a shorter length than the current string length is given. Therefore this subroutine can act as "prune_string" (see there). Actually it is used as backend for `s2s_prune`.

s2s_process (named)

```

str      str      string to be processed
deformat int      Boolean value indicating the operation
repair_i int      Boolean value indicating the operation
=>       undef    if no args or input string are given
=>       str      formatted string

```

Will process given string, if given indicator(s) exist(s).

`str` must exists, else undef value is returned

`c<deformat>` indicates whether to perform `s2s_deformat` operation, the default value is 1

`repair_i` indicates whether to perform `s2s_repair_interpunction` operation, the default value is 1

This subroutine dispatches several string processings on a formatting/low-level scope. Currently processed subs are `s2s_deformat` (see docs in this POD) `s2s_repair_interpunction` (see docs in this POD)

s2s_prune (named)

```

str  str      string to be pruned
len  int      desired absolute length | neg. int: relative strip
pos  str      position of prune: 'start' | 'end'
=>   undef    if no args, undef is returned
      str      formatted string

```

`str` string to prune; if not given, undef is returned

`len` resulting length of the string; if not given, undef is returned

`pos` position of the prune: two positions implemented: 'start' and 'end'; if not given, 'start' is the default position

Will prune a given string either to a given absolute length (if `len` is a positive integer), or relatively shorten by `-len` chars if `len` is given a negative integer.

To determine where to shorten the string, the `pos` parameter can be given. If none given, 'end' is assumed default as position. Any other string is assumed 'start'.

This subroutine uses `s2s_pad` (see there) as backend.

s2s_remove_headline (positional)

```
1  str    string to normalize
=> undef  if no input string given
    str    normalized string
```

Removes a header - a substring consisting of first upper case letter and no punctuation before linebreak. In the text:

```
Super header
```

```
Completeley new type of a header was invented,
designed and implemented in TeXLive Core...
```

Will be removed the substring 'Super header'. The side effect: If the text contains bad formatting, e.g. superfluous linebreaks like:

```
Super header
```

```
was invented on 22th of July 2025.
```

This function will repair it on:

```
Super header was invented on 22th of July 2025.
```

This is necessary to remove the real header.

s2s_repair_interpunction (positional)

```
1  str    string to normalize
=> undef  if no input string given
    str    normalized string
```

Performs a normalization on interpunction within the given scalar string. Space between comma, question/exclamation mark etc. and preceding text is eliminated. e.g.:

```
...text , other text ?
```

is transformed to

```
...text, other text?
```

s2s_text_repair_generic (named)

```
function str    name of repair function
args      href   parameters for chosen repair function
=>        real   distance
```

This function is a switch for different repair functions. Here is a list of possible text repair functions:

```
dehyphen           (see s2s_dehyphen)
compress_whitespaces (see s2s_compress_whitespaces)
deformat           (see s2s_deformat)
```

```

remove_headline      (see s2s_remove_headline)
repair_interpunction (see s2s_repair_interpunction)
trim_whitespacees   (see s2s_trim_whitespacees)

```

args depends on chosen distance function.

s2s_trim_whitespacees (positional)

```

1  str  string to format
=> str  empty string, if input string is not given
    str  formatted string

```

Removes leading and trailing whitespace characters from a given string.

s2s_unquote (positional)

```

1  str  string to format
=> str  the formatted string

```

If is the input string not defined, returns the original value.

If the given string starts and ends with quotes, removes them. Recognizes single quotes and double quotes. The value must begin and end with same type of quotes or nothing is done to the value. Undef input results in undef output.

s2s_var_replace (named)

```

str  str  string to replace
vars  href  hash reference containing variables
=>   str  empty string, if no input string is given
      str  original str., if no replacement given
      str  formatted string

```

str is a string in that the replacement should be done; if not given an empty string is returned

vars is a hash reference that contains variables for replacement; if not given original string is returned

Takes a string and makes replacement of given variable with a specified value. The variable in original string must be surrounded by '%' sign. E.g.:

```

str:  '%text%-%name%-%year%'
vars: {text => 'Poetica', name => 'GreekPhilosophy', year => '2025',}

```

Will return:

```
Poetica-GreekPhilosophy-2025
```

It is also possible to define default value for variables. In that case variable must be of the form %VARNAME default_value%.

F.3.251 PMLIB::s2x

PetaMem Library: Scalar -> (M)any functionality; with enrichment from various CPAN modules.

Functions Reference

s2x_get_ngrams (named)

```

txt      str|sref  input string
max      int[1>]  maximum number of ngrams
minl     int[1>]  minimum ngram length
maxl     int[1>]  maximum ngram length
tknzzr   cref    tokenizer
=>       lref     iso639-3 codes
=>       -        if max=0

```

The argument `txt` can be either a scalar or scalarref and denotes the text that will be processed (where ngrams get extracted).

If `max` is not given, the value of `$MAX_NGRAM_COUNT` is taken as default, if `max` is 0, a hashref instead of a listref is returned with the raw ngram hashref (key=ngram, value=number of occurrences).

The `minl` parameter is the minimum length of a ngram. If not given, `$MIN_NGRAM_LENGTH` is taken as default.

The `maxl` parameter is the maximum length of a ngram. If not given, `$MAX_NGRAM_LENGTH` is taken as default.

The `tknzzr` parameter is a code reference to a subroutine to provide the tokenization. If not given/by default, `PMLIB::s2l::tokenize` is taken as tokenizer. The sub itself should expect a hashref for named arguments, namely `str => <input string, copy of the original txt(scalar)>` `lang => <iso639-3 code, copy of the original lang(scalar)>`

The `lang` parameter is a hint for the tokenizer of the language of the given text.

F.3.252 PMLIB::x2f

PetaMem Library: (M)any -> File functionality

Functions Reference

x2f_acquire_data (named)

```

protocol      str  mech (default) or svn
threads       int  mech: number of threads
url           str  mech: url to get before parsing
get           str  mech: template for a link (use $vars & %VARS%)
match         str  mech: regex to capture strings from source
posthook_final str  mech: code to process after downloads finished
prehook_final str  mech: code to process before download
clean         str  mech: how old data to clean up
posthook      str  mech, svn: code to process after each download
prehook       str  mech, svn: code to process before each download
store         str  mech, svn: where to store downloaded content
source        str  svn: url path to check out

```

```

user          str  svn: username
auth          str  svn: password
<special>    str  all: %VARS% variables (names are not defined)
=>           lref  list of downloaded links

```

Build a path from a given string or list reference. By default, the input is.

x2f_any2path (named)

```

input  xref      input data as str (scalar) or already split (lref)
len    int [1>]  length of chunks to split string int
clean  bool      flag: clean the input data ? YES (default) NO
=>     str       path

```

Build a path from a given string or list reference. By default, the input is scanned cleaned malformed data, but that can be switched off.

The input can be string or list reference. In case of a string, the function `split_by_length` is used to obtain a listref with the chunks. In case the input is already a listref, the `len` argument is ignored. If the input is a string and `len` is not given, the default value 1 is assumed for `len`.

`clean` is a flag that controls whether the input shall be cleaned of dubious characters that constitute malformed file and path names, namely `'.'`, `'..'` and `'/'`. This flag is on by default.

x2f_load_struct (positional)

```

1  filename  storable file
=> struct    struct retrieved from the file

```

Retrieve perl data structure in Storable format from file 1.

x2f_remove_tree (positional)

```

1  list  dirs
=> int   number of files removed

```

Removes directory recursively.

x2f_save_struct (positional)

```

1  filename  storable file (filename)
2  xref      data struct to store
=> bool      indicate success

```

Save perl data structure 2 into specified file 1.

x2f_make_dir (positional)

```

1  dirname  storable file
=> bool      indicate success

```

Creates dir hierarchy.

x2f_safe_save (positional)

```

1  str  filename
2  bool flag, create file?
3  str  content
4  bool consider extension?
=> str  new filename

```

Saves content to file or/and returns suggested file name. If specified file exists, volume it.

x2f_save_robust (named)

```

file   str  filename
create bool flag, create? (default: 0)
data   sref what to store
=>     str  new filename

```

Create file with specified content. If specified file exists, volume it. Uses `x2f_write_file`.

x2f_copy_file (positional)

```

1  str  from filename
2  str  to filename
=> bool indicate success

```

Copy file 1 into file 2.

x2f_move_file (positional)

```

1  str  from filename
2  str  to filename
=> bool indicate success

```

Move file 1 into file 2.

x2f_write_file (named)

```

name  str  filename
cont  sref content (scalar ref)
mode  str  mode of writing (default 'raw')
=>    bool indicates success

```

Write content `cont` into file `name`. If `cont` is not given, empty string is used. If `mode` is not given, `raw` is used. Correct syntax for UTF-8 mode is:

```
encoding(UTF-8)
```

but the programm will complete the construction, even if you submit just `UTF-8`, `utf8` etc. This feature is supported just for UTF-8.

If no arguments given, function bails out.

F.3.253 PMLIB::x2h

PetaMem Library: (M)any -> Hash functionality This documentation covers only the reference for the respective methods. If you want more examples, HowTos and sample code, please see the PMLIB Cookbook POD.

Functions Reference

x2hr_ify (positional)

```
1   any    some structure
=>  href   hash
```

Transform any data structure to hrefref (considered a set) with the following semantics:

- * if a hrefref is given, it is returned unchanged
- * if a listref is given, all elements are stored in a hash, where the keys are the elements of the given list, and the values are the number of occurrence of that element (because it may have occurred multiple times), that way not much information is lost (the order is) and hr2lr_ify from h2l.pm can transform it back.
- * undef or simple scalar => empty set

x2h_compute_token_frequencies (positional)

```
1   xref      tokens as lref/href
=>  href      frequencies
```

Get the term frequencies for strings in a list or hash. This could be tokens of a text. Uses do_set to obtain the term counts and normalize them dividing by the number of all terms (sum of all counts).

x2h_compute_token_probabilities (positional)

```
1   xref      tokens as lref/href
=>  href      probabilities
```

Get the frequencies for strings in a list of hash and divide number of all tokens.

F.3.254 PMLIB::x2l

PetaMem Library: (M)any -> List functionality This documentation covers only the reference for the respective methods. If you want more examples, HowTos and sample code, please see the PMLIB Cookbook POD.

Functions Reference

x2l_split2lr (positional)

```
1   xref      delimited list of values (str) or listref
2   rx        delimiter (default: qr{,})
=>  lref      list
```

Expects as 1 a scalar that is a delimited list of values. Returns a listref of these values. If input was already listref, it is passed through unmodified. If an optional 2 is given, it is considered a specific regex to delimit the values given.

x2lr_ify (positional)

```
1   any    some structure
=>  lref   list
```

Convert any data structure to listref with the following semantics:

```
* if argument is undef, return empty listref
* if argument given is already listref, return unchanged
* if argument is a simple scalar, return listref containing
  this scalar as single element
* if argument is hashref, return listref with hashref element
```

xr2lr_ify (positional)

```
1   any    some structure
=>  lref   list
```

Convert any data structure to listref with the following semantics (same as x2lr_ify but last point):

```
* if argument is undef, return empty listref
* if argument given is already listref, return unchanged
* if argument is a simple scalar, return listref containing
  this scalar as single element
* if argument is hashref, call hr2lr_ify and return its return value
```

F.3.255 PMLIB::x2s

Functions Reference

x2s_get_same_size (positional)

```
1   xref   hash/list
2   xref   hash/list
=>  int    size of both hashes/lists
=>  undef  sizes differs
```

Size of both hashes/lists is returned in case it is same and elements disjoint.

x2s_is_sublexica (positional)

```
1   xref   sublexica extended with regexes
2   xref   lexica
=>  bool   is given sublexica less specific than lexica?
```

See PMLS manual to get more information about lexica structure. In short: returns true if 2 is substructure of 1.

You can use a regex anywhere instead of string in sublexica.

F.3.256 PMLIB::x2x

PetaMem Library: (M)any -> (M)any functionality This documentation covers only the reference for the respective methods. If you want more examples, HowTos and sample code, please see the PMLIB Cookbook POD.

Functions Reference

x2x_merge_structures (named)

```
data  lref    list of data structures
hook  cref    hook that can modify result
=>    struct  merger of given structure
```

A hook sub gets structures and suggested result of the merger. It can be modified. The original result is replaced by the return value of this sub.

x2x_isect_structures (named)

```
data  lref    list of data structures
hook  cref    hook that can modify result
=>    struct  isect of given structure
```

A hook sub gets structures and suggested result of the intersection. It can be modified. The original result is replaced by the return value of this sub.

x2x_subtract_structures (named)

```
data  lref    list of two data structures
hook  cref    hook that can modify result
=>    struct  isect of given structure
```

A hook sub gets two structures and suggested result of the difference. It can be modified. The original result is replaced by the return value of this sub.

x2x_complete_dependences (named)

```
size      int    size of ngram
dependence lref   list of disjoint lists of dependent tokens
=>        lref   extended C<dependence>
```

Example. We have 6-grams and dependences specified as `[[0, 5], [1, 2, 3]]`. Token 4 is not expressed there so result is `[[0, 5], [1, 2, 3], [4]]`.

dependence is extended with independent tokens as 1-element lists.

x2x_dep_plain2struct (positional)

```
1    str    dependence string
=>   lref   dependence list
```

Example. Converts `'0 5|1 2 3'` into `[[0, 5], [1, 2, 3]]`.

x2x_dep_struct2plain (positional)

```
1    lref   dependence list
=>   str    dependence string
```

Example. Converts `[[0, 5], [1, 2, 3]]` into `'0 5|1 2 3'`.

x2x_histogram2numeric (named)

```
<special> <special> args like in x2x_validate_args_for_metric and
tokenizer  cref    string parser (def: all word = 1 token)
```

```

subdistance  enum      second metric (def: discrete)
=>          href      numeric args for metric

```

Various distances in PMLIB::Metric allow to measure lists and histograms. This function preprocess args for them to be used in their numeric version.

In case `tokenizer` and `subdistance` params are passed, comparing tokens can be generalized. By default, the distance of different tokens is 1 and distance of same tokens is 0 (discrete distance). These two options make it scalable.

Second metric is intended for concatenated measures.

x2x_validate_args_for_metric (named)

```

type  enum  list/hash
args  href  see above
=>    lref  [l/href,l/href] which is [group1, group2]
=>    undef something wrong

```

Validate args for use in metric from PMLIB::Metric.

args looks following:

```

l1  lref  group1
l2  lref  group2
s1  href  group1
s2  href  group2

```

You should specify either `l1` and `l2` or `s1` `s2`.

x2x_walk_struct (named)

```

rdepth  int      current recursion depth (default: 0)
rlimit  int      recursion depth limit (default: 0)
struct  any      data structure to walk
cargs   href      arguments for coderef
result  lref      result storage container: (default: [])
stack   lref      stack (default: [])
code    cref      code
noinit  bool      perform coderef initialization call
nofinal bool      perform coderef finalization call?
=>      bool      indicate success

```

Offers a "recursive structure walker", which can be given a `coderef` and thus various operations on a complex structure (composed of native perl structures) can be performed.

F.4 APIs for PMSE:: Namespace

F.4.1 PMSE

This module exports common functions for PMSE project

OPTIONS**-cpu <n>**

set the number of CPUs manually. Actually this is recognized automatically and influences the number of parallel started processes (if possible). With this you can override the autodetect.

-dry

dry run facility. Do not actually execute the commands

-fpmode [auto|mem|iter]

Determines the file processing mode. Depending on size, some files may be too big to process in memory and have to be processed piece by piece. Normally, a PMSE script detects such a situation and chooses the right file processing mode.

```

auto    autodetection (default)
batch   force in-memory processing
iter    force iterative processing

```

-?**-help**

print the script specific help and the general PMSE help

-info

information about the current environment, such as arguments, detected values, etc. When this parameter is detected the information delivered is current i.e. parameters AFTER this parameter are not considered yet. If you want the final information for all parameters, place this parameter last on the commandline

-report [<n>]

Defines the reporting level. By default all scripts have no output. If you'd like to know more about what's going on, set this option. Setting <n> higher will produce more verbose output. If you want to silence the scripts set <n> to 0 or omit it by just giving '-report'.

-version

prints the version number of the script, and of PMSE and exits.

Functions Reference**ask_for (named)**

```

msg      str    a message
need     int    importance status
check    bool   control code
checkmsg str    err. message
default  bool   control value
type     str    type of argument

```

```

=>      bool   type of output depends on the input argument type
=>      undef  if input args are bad specified
=>      href   type of output depends on the input argument type
=>      lref   type of output depends on the input argument type

```

This function provides handling of input parameters coming from the interactive PMSE mode.

bexec (positional)

```

1  str   the command to be executed
=> undef no output

```

Bexec is a wrapper that executes external commands.

check_free_hd (positional)

```

1  str  location of free space check
=> int  returns the free space

```

A wrapper for df. Returns 0 if df returns no output.

check_relative_speed (void)

```

=> int  result of the time test

```

Performs a time test and returns the result in seconds.

cmd_call (positional)

```

1  str  cmd from PMSE bin
=> lref full cmd with binary paths, options etc.

```

cmp_file_mem (positional)

```

1  str  file name
=> bool indicates if file can be stored in memory

```

This fnc. indicates whether a file can be store in the memory or not.

dryrun (void)

```

=> bool indicates if dryrun is in use

```

This fnc. returns true if dryrun was specified on the CLI.

err (positional)

```

1  str  error message
2  bool flag (exit after err printout)
3  bool flag (print help)
=> undef if arg 1 is not specified
=> bool 1 - indicates that all is wrong

```

Error handling for PMSE.

err_filecmd (positional)

```
1 int    exit code
=> undef returns nothing at all
```

Substitution of file and CMD error handling.

err_filesize (positional)

```
1 int    exit code
=> undef returns nothing at all
```

Big file size error handling.

err_nofile (positional)

```
1 int    exit code
=> undef returns nothing at all
```

No input file error handling.

get_fpmode (positional)

```
1 str    path to file
=> str    type of processing
```

Provides batch processing functionality. Returns types of processing mode.

get_regex (named)

```
anchor str    denotes section in bulk file
bulk   str    path to the bulk file
code   cref   code to apply
hook   str    denotes specific part of section in the bulk file
=>     lref   array reference contains regexps to process
```

Read a regexp(s) from a bulk file. Anchor and hook are necessary to identify the correct section to process.

getval_from_file (positional)

```
1 str    path to a file
2 str    row
3 str    column
=> str    empty string or the appropriate column
```

Reads a list for a file. Returns a value from a table. 2 and 3 are row and column specifications.

help (positional)

```
1 str    specific help
=> undef returns nothing at all
```

Provides generic PMSE help.

histogram (named)

```
opt     href   href with options
result  href   href with histogram data
```

```
=> href histogram
```

Filtering of histogram data.

init (void)

```
=> int returns time of start
```

Initial check of PMSE infrastructure.

interactive (void)

```
=> bool indicates if iact mode is in use
```

Small wrapper function.

list_params (named)

```
exit    bool    want to exit the original script?
opt     str     name of option
params  lref    list (ref.) of parameters
script  str     name of script
=>      undef
```

This function will list all possible parameters for an option and exit, if wanted.

ofilter (named)

```
bulk    str     path to a bulk file
code    cref    to apply
hook    str     specific section of the bulk file
result  href    href with data to filter
=>      href    filtered data
```

Output filter for P_csp and P_gnp.

pmse_info (void)

```
=> undef returns nothing at all
```

Dumps PMSE info.

pmse_report (positional)

```
1  str    text to print out
2  int    level of verbosity
3  bool   flag if wait
=> undef  nothing at all
```

Provides extended information format for err and info messages inside PMSE.

pmse_version (void)

```
=> undef nothing at all
```

Prints version of the script / module.

process_file (positional)

```

1  str  path to a file
=> cref  processing line by line
=> str  content of a line / file

```

Reads specified file according to a mode.

process_option (named)

```

bulk    str  path to a bulk file
code    cref  code to apply
hook    str  specific section of bulk
tokens  lref  tokens to process
=>      lref  original / modified tokens

```

Provides P_csp's or P_gnp's 'process' functionality.

programe (void)

```

=> str  PMSE script invocation name

```

Returns name of the PMSE script.

store_data (named)

```

bottommark  xref  polymorphic data
data        xref  polymorphic data
format      str   format of output
out         str   dirname or STDOUT
outfile     str   outputfile
topmark     xref  polymorphic data
=>         undef  if outfile is not specified

```

Stores data according to specified format.

register_pmse (positional)

```

1  str  script data
=> href  info about the script

```

Returns a sum of available info for a script.

verify_output (positional)

```

1  href  data-hash
2  str   P_csp action
=> undef  nothing at all

```

Checks if the result of action is not empty.

vidattr (positional)

```

1  str  string to be escaped
2  str  comma-separated list of attributes
=> str  string with formatting

```

Provides string formatting for CLI output.

F.4.2 PMSE::Categorization

routines for running categorization process PMSE::Categorization provides a code frame for performance of the text categorization.

Functions Reference

add (named)

```
to      lref   target array
this    lref   array to be added
=>      undef  undef
```

This function sums 2 two-dimensional arrays. A target listref must be specified at least, if not, undef is returned.

analyze_clusters (positional)

```
1      href   hash reference holding the results of TextCat
=>     undef   if no input args given
=>     href   hash reference with info for each cluster
```

This fcn. takes each cluster - group which points to 2 other groups - and counts its statistical information:

```
mean      mean count of texts per the sub-clusters

variance  variance in texts distribution among sub-clusters

purity    is a clustering quality measure:
           it is counted as

           (1 / sum_of_all_texts) * sum_of_max_in_category

           e.g.: we have super-cluster with 2 sub-clusters,
                   20 texts, 2 categories
           cl1 => 5 texts from cat A
           cl2 => 15 texts - 10 from cat B, 5 from cat A
           then Purity = (1/20) * 15 = 0.75

entropy   is also clustering quality measure
           it is counted as:

           Entropy = - (1/ log num_of_categories) * SUM
           SUM      = sum (max_in_category / texts_per_subcluster)
                       * (log (max_in_category / texts_per_subcluster))

           from the example above:
           E = - (1/ log 2) * ( ( (5/5) * (log(5/5)) ) + ( (10/15)
                       * (log(10/15)) ) )

           E = 0,389975
```

This assumes you know the categories of texts BEFORE categorization. To get correct results of cluster evaluation, you have to name your input texts correctly. This function is designed for text-names like:

```
perl-1.txt
```

Where 'perl' is a name of the category. Only this string will be taken as a category name. The categories are recognized automatically from the names of the texts. The dash '-' and '.txt' suffix are mandatory.

categorize (named)

```
groups  lref    groups with texts/clusters
opt      href    input options for TextCat
procdir scalar  directory holding current run of TextCat
=>      undef   unless all 3 arguments are specified
        href    the results of TextCat
```

This function wraps code and sub calls for the TextCat process.

The resulting data-structure is a binary tree. It is returned as a hash reference, where each key is the group and its value is a sub-cluster or a 'leaf' - the input text. The highest group (key) is the root node of the graph.

categorized2dot (named)

```
opt      href    input options for TextCat
procdir  scalar  directory holding current run of TextCat
results  href    hash reference with the results of TextCat
=>      undef   unless all 3 parameters given
=>      scalar  name of the dot file
```

Categorized2dot converts the resulting data-structure into dot language. The dot code is convertible into "readable" format - e.g.: svg, jpg etc., which will let you display the binary tree in a "human readable form".

This fnc. returns the name of the dot file.

corpora_list (named)

```
root    scalar  directory containing 'derived' and 'original' dirs
=>      undef   if input argument(s) are not given
=>      lref    input texts
```

We consider each single input text as a 'corpus'. This fnc. checks all input files - the 'corpora' - and returns their list in an array reference.

count_cluster_statistics (named)

```
data      href    href holding info for each subcluster
categories int    automatically recognized from the names of texts
=>        href    hash-reference contains data for each super-cluster
```

The keys of the resulting hash are: sample, variance, purity and entropy. See the documentation for 'analyze_clusters' sub for detailed info.

create_groups (named)

```
corpora_list lref    list of input files
opt          href    input options fro TextCat
```

```
=>      undef  unless both params specified
=>      lref   list of groups and their properties
```

This fnc. creates a basic elements of TextCat - groups, which are in fact hashes filled with text names and lexicostatistical data.

(In the beginning, each text has one group. As the process goes on, primary groups are grouped together and create clusters.)

Arref containing hash refs is returned.

distance (named)

```
group1 href  data for group 1
group2 href  data for group 2
opt     href  input opts for TextCat
=>      undef  unless all 3 params given
=>      real   distance between groups
```

Counts distance between groups.

filter_corpora (named)

```
corpora_list lref  list of input texts
opt          href  input options for TextCat
procdir      scalar directory holding current run of TextCat
=>           undef  undef
```

Filters corpora contents according to some criteria (see PMSE::Categorization::FilterNgrams:: namespace).

filter_files (named)

```
corpora_list lref  list of input texts
opt          href  input options for TextCat
=>           undef  unless both params specified
=>           lref  original corpora_list if fi. opts aren't specified
=>           lref  filtered corpora
```

This fnc. wraps code for filtering of files. If no options for filtering specified, it returns original corpora list.

flatten_tree (positional)

```
1 href      results of TextCat
2 lref|scalar subcluster (arref) or text name (scalar)
3 lref      empty list or l. of text names
=> lref      list of text names for given tree
```

This function takes the result of TextCat, hash storing binary tree, and flattens the tree (or its branch) and extracts the end 'leafs'- text names.

get_categorization_name (named)

```
opt href  input options for TextCat
=> undef  if no options specified
```

```
=> int name of procdir (which is number)
```

This function checks the TextCat environment and set the name for current procdir.

get_vector (named)

```
corpus str text name
opt href input options for TextCat
=> undef unless all params specified
=> href vector info
```

This fnc. wraps code and sub calls for vector support. Vector is an abstract element of TextCat. It contains specifications for the process, e.g. the criteria of similarity measurement.

This sub is focused on the 'leaf' groups - groups, which contain a text name.

get_vector_for_cluster (named)

```
opt href input options for TextCat
groups lref groups with their properties
=> undef unless all params specified
```

The same as `get_vector`, but for cluster groups.

next_active_combination (named)

```
current lref current combination
n int data size
k int group size
active lref list of active data ids
=> lref combination
```

Returns n combinations after given one

get_couple_generator (named)

```
first lref current value
n int data size
k int group size
active lref list of active data ids (all by default)
new cref generator of next combination
=> iterator combination
```

Returns iterator for couple generation.

get_couples_generator (named)

```
first lref current value
n int data size
k int group size
active lref list of active data ids (all by default)
new cref generator of next combination
count int count of couples in each step (default: 5000)
=> iterator combination
```

Returns iterator for couple generation.

parse_opt (positional)

```
1 href hash reference with input options
=> undef if no options given
=> href processed input options
```

This sub parses input option called `vector`. It returns the option as detailed data-structure stored as a `href`.

precompute_distances (named)

```
cpus int number of available cpus
distances lref computed distances
groups lref groups with their properties
new int unless defined, all distances will be computed
opt href input options for TextCat
procdir scalar current procdir
=> undef unless opt,groups,procdir or input hash defined
=> lref distances - 2dimensional array
```

This fnc. computes distances between existing groups.

preprocess_corpora (named)

```
corpora_list lref list of texts
opt href input options for TextCat
=> undef unless all params defined
```

This fnc. wraps code and sub calls for preprocessing of texts. Preprocessing means obtaining data as frequency lists or ngrams, which are needed for the computation of distances among texts.

preprocess_corpus (named)

```
opt href input options for TextCat
file scalar text name of text to process
=> undef unless all params specified
=> href lexicostatistical data
```

This fnc. provides tokenisation for given text file. It also provides n-grams creation. It returns `href` with lexicostatistical data.

update_active (named)

```
active lref list of active groups
activate lref list of groups to activate
deactivate lref list of groups to deactivate
=> undef if not input or active specified
=> lref list of active groups
```

This fnc. handles lists of groups during the process of categorization.

F.4.3 PMSE::Categorization::FilterFiles

Wrappers for routines for files filtering in categorization process.

Functions Reference

filter_files_generic (named)

```
function str    name of filter_files function
args         href parameters for chosen filter_files function
=>          lref filtered corpora
```

F.4.4 PMSE::Categorization::FilterFiles::Pareto

Functions Reference

pareto (named)

```
root         path  directory of given language
corpora_list lref  relative paths to corpora
=>          lref  filtered corpora list
```

Makes a pareto cut of file list. It means that too big (20% of size) and too small (20% of size) of corpora will be cut off.

F.4.5 PMSE::Categorization::FilterFiles::Raw

Functions Reference

raw (named)

```
corpora_list lref  relative paths to corpora
=>          lref  corpora_list
```

Filter nothing. Return a list which get.

F.4.6 PMSE::Categorization::FilterNgrams

Functions Reference

filter_ngrams_generic (named)

```
function str    name of filter_ngrams function
args         href parameters for chosen filter_ngrams function
=>          undef undef
```

Make ngrams filtering.

F.4.7 PMSE::Categorization::FilterNgrams::Pareto

Functions Reference

filter_ngrams_pareto (named)

vector	href	parameters for all vectors
file	path	relative path to corpora
root	path	directory of given language
corpora_list	href	list of corpora
=>	undef	

Makes a pareto cut of file ngrams. It means that too frequent (20% of occurrences) and too small (20% of occurrences) ngrams will be cut off.

F.4.8 PMSE::Categorization::Vector

Wrappers for various vector plugins for categorization.

Functions Reference

distance_key_generic (named)

function	str	name of key_distance function
args	href	parameters for chosen key_distance function
=>	struct	distance

Wrapper for various distance_key functions. Each computes distance between 2 groups according to its definition.

get_vector_generic (named)

function	str	name of get_vector function
args	href	parameters for chosen get_vector function
=>	struct	vector

Wrapper for various get_vector functions. Each computes vector from preprocessed corpus according to its definition.

get_vector_for_cluster_generic (named)

function	str	name of get_vector_for_cluster function
args	href	parameters for get_vector_for_cluster function
=>	struct	vector

Wrapper for various get_vector_for_cluster functions. Each computes vector from two groups according to its definition.

F.4.9 PMSE::Categorization::Vector::Frequent

Functions Reference

distance_key_frequent (named)

group1	href	precomputed group1 vector
group2	href	precomputed group2 vector
vector	href	parameters of frequent method
=>	real	distance

Computes frequent-distance between **group1** and **group2**. Parameters of this computation are in **vector**.

It accepts following items in **vector**:

```
distance  str    name of distance metric
weight    num    number to multiply the result
```

get_vector_frequent (named)

```
corpus      path    relative path to the corpus
vector      href    parameters of frequent method
root        path    directory of given language
filter_tokens  enum  which preprocessed tokens should we use
=>         href    vector for given corpus
```

Computes vector of the given group for vector frequent.

get_vector_for_cluster_frequent (named)

```
vector      href    parameters of frequent method
group1      path    vector for group1
group2      enum    vector for group2
=>         href    vector for given cluster
```

Computes vector of the cluster for vector frequent.

F.4.10 PMSE::Corpus

This is a parent class for corpora manipulation.

Methods Reference

pdump (void)

```
=>  undef
```

Dump data to STDOUT

store (named)

```
out  str    directory to dump
=>  undef
```

Store data to 'out'. If not defined, use CWD.

F.4.11 PMSE::Corpus::BNC

This is a class for BNC manipulation.

DESCRIPTION The specification and documentation for BNC can be found here: <http://www.natcorp.ox.ac.uk/>

The XML construction of BNC is described here: <http://www.natcorp.ox.ac.uk/XMLedition/>

Tagset - POS annotation is described here: <http://www.natcorp.ox.ac.uk/docs/URG/posguide.html>
 BNC consists of english written (and spoken) texts only.

Methods Reference

BUILD (void)

```
=> bool returns 1 if succeeded
```

Internal method.

convert (named)

```
metainfo bool flag
=> xref listref with converted data
```

Convert data to internal data-structure format. Argument `metainfo` denotes if we want meta information or not. If yes, a hashreference will be returned, where keys are IDs of sentences. If not, data structure will have form of list reference.

new (named)

```
source str path to source
=> obj new object
```

New object has to be invoked with `source` parameter.

pdump (void)

```
=> undef
```

Dump data to STDOUT.

F.4.12 PMSE::Corpus::CNK

Handle vertical corpora with CNK attributes.

DESCRIPTION Supports written language corpora so far: <https://ucnk.ff.cuni.cz/struktura.php>

Methods Reference

BUILD (void)

```
=> bool returns 1 if succeeded
```

Internal method.

convert (named)

```
direction str direction of conversion
=> href href with converted data
```

Convert data to internal data-structure format. Direction may be:

```
cnk2pmts
pmts2cnk
```

new (named)

```
source str path to source
=>      obj new object
```

New object has to be invoked with `source` parameter.

pdump (void)

```
=>      undef
```

Dump data to STDOUT.

F.4.13 PMSE::Corpus::OANC

This is class for OANC corpora manipulation.

DESCRIPTION OANC stands for "The Open American National Corpus". Info about construction and composition of OANC can be found here: <http://www.americannationalcorpus.org/>

OANC consists of english texts.

Methods Reference**BUILD (void)**

```
=>      bool returns 1 if succeeded
```

Internal method.

convert (void)

```
=>      lref listref with converted data
```

Convert data to internal data-structure format. Converted data will also replace 'data' section of the object.

new (named)

```
source str data source
=>      obj new object
```

This class must be invoked with 'source' parameter. Source is a path to file (source) of the corpus.

pdump (void)

```
=>      undef
```

Dump data to STDOUT.

F.4.14 PMSE::Corpus::PDT

Import PDT xml trees into PMSE.

DESCRIPTION Supports PDT 3.0 so far. <http://ufal.mff.cuni.cz/pdt3.0>

Note on annotation levels in PDT. PDT has 3 annotation level (m,a,t) + one non-annotation layer (w). These are:

```
m morphological layer
a analytical layer
t tectogrammatical layer

w word layer (words, paragraphs, sentences)
```

In tectogrammtical layer, every sentence is represented as a rooted tree with labeled nodes and edges. Each level has several attributes.

```
morphological: (4)
  lemma form representing whole paradigm of a word
  tag    morphological information (15 positions)
  id     unique identifier ("a" layer pointer)
  w.rf   reference to the word level

analytical: (6)
  id     unique identifier
  ord    linear order
  afun   analytical function
  is_member (syntactical) coordination / apposition
  is_parenthesis_root (syntactical) parenthesis
  m.rf   links node to the morphological layer

tectogrammatical: (39)
  id     unique identifier
  nodetype type of the node
  functor describes the type of the edge leading
         from the node to its governor
  t_lemma tectogrammatical lemma of the node
  gram/*  grammatem: a group of 16 attributes
  ...
  tfa     topic-focus articulation
  deepord (numeric) underlying ordering of nodes
  coref_text.rf textual co-reference
  coref_gram.rf grammatical co-reference
  compl.rf special co-reference
```

Other attributes were not mentioned in the original documentation: <http://ufal.mff.cuni.cz/pdt2.0/documentation/en/html/ch02.html>

Methods Reference

BUILD (void)

```
=> bool returns 1 if succeeded
```

Internal method.

convert (named)

```
direction  str  direction of conversion
=>         href href with converted data
```

Convert morphological annotation of PDT corpus. Available directions are:

```
pdt2pmts
pmts2pdt
```

new (named)

```
source  str  path to source
=>      obj  new object
```

New object has to be invoked with `source` parameter.

pdump (void)

```
=>  undef
```

Dump data to STDOUT.

F.4.15 PMSE::Corpus::PM**Create and convert PetaMem (PMLS) corpora.**

DESCRIPTION PM corpus supports 2 structural levels: Files (opuses) and sentences. Both levels have `data` and `<meta>` sections.

Methods Reference**BUILD (void)**

```
=>  bool  returns 1 if succeeded
```

Internal method.

context2pmts (named)

```
context  lref  AoA with sentence
lang     str   ISO 639-3 code
=>       href  hash reference (PM format)
```

This is a helper function. It converts whole sentence (as is represented in `PMLIB::Tag`) to PM format.

convert (named)

```
direction  str  specify direction of tagset conversion
format     str  specify output format
=>         any  converted corpus
```

Gets input data and converts it to specified tagset and format. Format may be:

```

vertical
plain
xml

```

new (named)

```

source any path to source or data struct
=>     obj  new object

```

New object has to be invoked with `source` parameter. You may provide path to xml file or a href with data.

pdump (void)

```

=> undef

```

Dump data to STDOUT.

F.4.16 PMSE::Corpus::PM::Sentence**Manipulate PM corpus on the level of sentences.**

DESCRIPTION Make iteration of sentences elegantly.

Methods Reference**BUILD (void)**

```

=> bool returns 1 if succeeded

```

Internal method.

iterate (named)

```

back any data structure to modify
code cref subroutine to be executed
elem str element of meaning
=> any returns modified 'back'

```

This method will take single sentence and will iterate over all it's positions. A reference to an anonymous subroutine (`code`) will be called in each iteration, feeded with hash of options, where:

```

A annotation: element of meaning (on given position)
B back:       container that holds data
K token:     token for given position

```

new (named)

```

source href sentence
=>     obj  new object

```

New object has to be invoked with `source` parameter. Provide a hash reference with the data.

F.4.17 PMSE::Corpus::Penn

This is class for Penn-like corpora manipulation.

DESCRIPTION The specification and documentation for Penn treebank can be found here: <http://www.cis.upenn.edu/~treebank/>

The original corpus consists only of english texts.

Methods Reference

BUILD (void)

```
=> bool returns 1 if succeeded
```

Internal method.

convert (named)

```
direction str tagset conversion direction
lang      str language of the tagset
=>        lref listref with converted data
```

Convert data to internal data-structure format. Converted data will also replace 'data' section of the object.

new (named)

```
source str data source
=>      obj new object
```

This class must be invoked with 'source' parameter. Source is a path to file (source) of the corpus.

pdump (void)

```
=> undef
```

Dump data to STDOUT.

F.4.18 PMSE::Corpus::WikiCorpus

WikiCorpus manipulation.

DESCRIPTION You can read: * the original corpus * convert it into internal data-structure * convert Penn tags into Pmts tags * dump the result

The specification and documentation for WikiCorpus can be found here: <http://www.lsi.upc.edu/~nlp/wikicorpus>

The original corpus consists of thre languages: eng, spa & cat. Currently is supported only eng.

Methods Reference**BUILD (void)**

```
=> bool returns 1 if succeeded
```

Internal method.

convert (void)

```
=> lref listref with converted data
```

Convert data to internal data-structure format. Converted data will also replace 'data' section of the object.

new (named)

```
source str data source
=>      obj new object
```

This class must be invoked with 'source' parameter. Source is a path to file (source) of the corpus.

pdump (void)

```
=>      undef
```

Dump data to STDOUT.

F.4.19 PMSE::DM**Generic functions for Data Mining.****Functions Reference****list_cfg_files (positional)**

```
1  str    root of the library
=> undef  if no library root specified
=> lref   paths to config files
```

This function will return an array reference with absolute paths to .dmf.info files found in the location specified by the only input argument.

F.4.20 PMSE::DM::Aspell

This module provides a data mining functions for Aspell dictionaries.

Functions Reference**get_wordlist_from_source (named)**

```
base      str    path to the library root
dict      str    path to a packed (tar.bz2) dictionary
lang      str    aspell code of given language-dictionary
```

```

variety str    specify a variety to process
=>         undef undef

```

This function extracts a wordlists from aspell source aspell dictionary (<*tar.bz2> file). You need to specify name of given language to process. Aspell language name are similar to locale configuration, e.g.:

```
en_GB
```

The resulting file will be stored as <base>/i/s/o/original/aspell*.bz2

get_wordlist_from_system (named)

```

base      str    path to the library root
cpus      str    optional - num of cpus
lang      str    comma separated list of aspell-lang codes
variety   str    variety to process
=>        undef

```

This function extracts wordlists from aspell dictionaries installed on the system. The function checks automatically the available dictionaries installed on the system.

If the `lang` argument is given, only dictionaries for specified languages will be processed.

The `cpus` argument specifies available cpus. If `cpus > 1`, parallel processig is started.

This function awaits already installed aspell dictionaries.

Note: aspell-lang codes are similar to locale codes.

list_dictionaries (void)

```
=> href hash reference with available dictionaries
```

Print out available dictionaries with a brief characteristics.

F.4.21 PMSE::Library

Generic functions for Data Mining.

Functions Reference

list_cfg_files (positional)

```

1  str    root of the library
=> undef  if no library root specified
=> lref   paths to config files

```

This function will return an array reference with absolute paths to `.dmf.info` files found in the location specified by the only input argument.

F.4.22 PMSE::Visualize

This is a parent class for visualization

Methods Reference

document_self (named)

```
section str    name of section to document
=>        str    help for given section
```

Return formatted string with help.

is_public (named)

```
method str    method to check
=>          bool returns true / false value
```

This function checks, if given method may be used in P_dvf as public.

pdump (named)

```
out str|FH    name of the output file or filehandle
=>          undef
```

Use Data::Dumper to print out the input data structure.

storable (named)

```
out str    name of the output file
=>        undef
```

Store data as a Storable file.

text (named)

```
out str|FH    name of the output file or filehandle
=>          undef
```

Pretty printed data-structures.

yaml (named)

```
out str|FH    name of the output file or filehandle
=>          undef
```

Transform data into YAML format.

graphic (named)

```
out str|FH    name of the output file or filehandle
=>          undef
```

Pretty printed data-structures.

F.4.23 PMSE::Visualize::BinaryTree

conversion to GraphViz Language for binary tree data structure

Methods Reference

BUILD (void)

```
=> undef returns nothing
```

Internal method, sets attributes.

newick (named)

```
out str output file
=> undef undef
```

Will create a tree in the newick format. For details on newick format see:

http://en.wikipedia.org/wiki/Newick_format <http://evolution.genetics.washington.edu/phylip/newick-tree.html>

The advantage is, that the newick is commonly used, it is also a part of the NEXUS format.

Trees in newick format may be also loaded into R. See <http://www.r-phylo.org/wiki/HowTo/In-puttingTrees> for details.

spreadsheet (named)

```
out str output directory
=> undef return value is undef
```

Will create 2 files in a directory specified by `out` parameter. First file is called `nodes.csv` and contains specifications of nodes in CSV format. Second file is called `edges.csv` and contains specification for edges.

The format specification was taken from the Gephi project: <https://gephi.org/users/supported-graph-formats/csv-format/>

The `nodes.csv` file may look like this:

```
Id;Label;Time Interval
1;AA;"<[1,Infinity]>"
2;BB;"<[2,Infinity]>"
3;CC;"<[3,Infinity]>"
4;DD;"<[4,Infinity]>"
5;5;"<[5,Infinity]>"
6;6;"<[6,Infinity]>"
7;7;"<[7,Infinity]>"
```

The column called 'Time Interval' allows you to build a dynamic graph, where nodes and edges are appearing according to a time line. Example of edges:

```
Source;Target;Label;Time Interval
1;5;"<[5,Infinity]>"
2;5;"<[5,Infinity]>"
3;6;"<[6,Infinity]>"
4;6;"<[6,Infinity]>"
5;7;"<[7,Infinity]>"
```

```
6;7;"<[7,Infinity]>"
```

graphic (named)

```
out    str    define output file
otype  str    define output format
fmt    str    GV graph format - !!! NOT IMPLEMENTED YET !!!
=>      undef undef
```

Will visualize a binary tree in GraphViz. The output file is in the SVG format.

F.4.24 PMSE::Visualize::Contingency

Conversion to CSV / R for contingency data structure

Methods Reference

BUILD (void)

```
=>    undef  returns nothing
```

Internal method, sets attributes.

text (named)

```
out  str    name of outfile or STDOUT
=>    undef  returns undef
```

This method will write a CSV file, which can be loaded in spreadsheet editor or R.

filter (named)

```
bulk    str    path to a blk file
filter  cref   filtering code
=>      undef  returns undef
```

This method allows to filter the resulting set of lines in a contingency table. This means, that the counts (in rows of the table) are not affected.

Filtering serves only to decide which members of the contingency table to display and which not. Code may be:

```
$key =~ m{\d+}xms
$val == 60
```

the `$key` variable is the token itself. The `$value` variable is the total count of the token. Note: The term 'token' stands here for n-gram.

You may specify the filtering via a bulk file.

visualize (named)

```
out  str    name of outfile or STDOUT
=>    undef  returns undef
```

The graphic output of the contingency table data is a mosaic plot in SVG. This plot is rendered via the R and the 'vcd' library.

The readability of the graph is affected by the amount of the data you load in. If the data is too big, use filter option or an alternative method of visualization.

If STDOUT is chosen as output, the R matrix will be printed to STDOUT. (No svg file will be created.)

F.4.25 PMSE::Visualize::Cooccurrence

Conversion to SVG/dot for cooccurrences

Methods Reference

BUILD (void)

=> undef returns nothing

Internal method, sets attributes.

visualize (named)

out str output file
=> undef returns nothing

Build dot graph of cooccurrences from given input data structure. It takes two input arguments

otype str may be text or svg (svg is default)
out str name of the input file

The input array reference (self->{data}) should have this structure:

```
[
  { target-word(0) => { cooc1, cooc2, cooc3 },
  { cooc1(1)      => { cooc-of-cooc1, cooc-of-cooc1 ... },
]
```

Target-word is placed on level 0. Cooc1 is the cooccurrence of the 1st order. Then target-word and cooc-of-cooc1 are cooccurrences of the 2nd order etc.

F.4.26 PMSE::Visualize::Distance

Conversion of distance information into CSV / text

Methods Reference

BUILD (void)

=> undef returns nothing

Internal method, sets attributes.

spreadsheet (named)

```
out str    output fileP/filehandle
=> undef  returns nothing
```

Will display distances in CSV format.

text (named)

```
out str    output fileP/filehandle
=> undef  returns nothing
```

Will display distances in text format. Values are separated by tab.

F.4.27 PMSE::Visualize::Dot

Conversion to GraphViz Language

Methods Reference

add_C_components (named)

```
mode enum mode (edge (default), node)
FH    fh    filehandle
=>    undef undef
```

This is an auxiliary function that feeds the input filehandle variable with the data to print.

build_tree (named)

```
struct href results of TextCat
analyzed href results of clusters analysis
=>      undef if no struct specified
=>      scalar dot code
```

This function visualizes the results of textCat. It returns dot code of a binary tree.

build_cooccurrences (named)

```
occurrences lref occurrences
FH          fh    filehandle
=>          undef undef
```

Build dot graph of cooccurrences from given input data structure.

The input array reference should have this structure:

```
[
  { target-word(0) => { cooc1, cooc2, cooc3 },
    { cooc1(1)      => { cooc-of-cooc1, cooc-of-cooc1 ... },
  ]
```

Target-word is placed on level 0. Cooc1 is the cooccurrence of the 1st order. Then target-word and cooc-of-cooc1 are cooccurrences of the 2nd order etc.

F.4.28 PMSE::Visualize::FileStat

Conversion of file information to CSV / text format

Methods Reference

BUILD (void)

=> undef returns nothing

Internal method, sets attributes.

spreadsheet (named)

out str output fileP/filehandle
=> undef returns nothing

Will display file info in CSV format.

text (named)

out str output fileP/filehandle
=> undef returns nothing

Will display file info in text format. Values are separated by tab.

F.4.29 PMSE::Visualize::Histogram

Conversion to various formats from histogram-like data structure

Methods Reference

BUILD (void)

=> undef returns nothing

Internal method, sets attributes.

pdump (named)

out str define outfile
sort str sort order
=> undef returns undef

Use Data::Dumper to get printed data structure. Sort order is optional and it may be

+key
-key
+val
-val

text (named)

fmt str define format of one key/val pair
limit int specify limit of lines to print
out str name of outfile or STDOUT

```

sort    str    sort order
=>     undef  returns undef

```

This method will write a text file. Basically, it is a list which consists of key - value pairs. This is a basic data structure for all kinds of language data as

```
<token> <value>
```

This may be e.g. word - frequency pairs. The format of one pair is affected via `fmt` option:

```

%k \t %v
%v \t %k
%k***%v
%k,%v

```

which is a template. `%k` will be replaced by a key, `%v` with a value.

The `limit` is the maximal count of lines to print out. If `limit` set e.g. on 30, exactly 30 lines will be printed out (if the input file has at least 30 key - value pairs).

Sort order is similar to `pdump`.

yaml (named)

```

out    str    define outfile
sort  str    sort order
=>    undef  returns undef

```

Will write out the input in YAML format. Sort order is similar to `as_printed_data_structure`.

filter (named)

```

bulk   str    define an INI file
filter rx     define filtering
=>     undef  returns undef

```

Define a negative filter with Perl code `filter` or in a separate file `bulk`. Tokens / values which don't match will be deleted.

Filter may e.g. be:

```

$key =~ <regexp>
$value > <value>
$key !~ m{\d+}xmsg # delete all non-digits

```

graphic (named)

```

out  str    name of outfile or STDOUT
=>  undef  returns undef

```

The input data structure is visualized as a histogram (numerical histogram) or a word cloud (histogram-like data as frequency lists).

Both types of graphics are rendered via R.

The second factor of the word-cloud is the length of the token.

F.4.30 PMSE::Visualize::Neighbors

Conversion to YAML for neighbors-like data

Methods Reference

BUILD (void)

```
=> undef returns nothing
```

Internal method, sets attributes.

pdump (named)

```
out str define outfile
```

```
=> undef returns undef
```

Use Data::Dumper to get printed data structure.

yaml (named)

```
limit str limit output
```

```
out str define outfile
```

```
sort str sort order
```

```
=> undef returns undef
```

Will write out the input in YAML format. Sort order may be:

```
+key
```

```
-key
```

```
+val
```

```
-val
```

Limit may be simple integer or '<int1>|<int2>' string, where int1 is an offset applied to keys from data section, int2 is a length (just like in perl's `splice`).

```
1000
```

```
'1000|9000'
```

F.5 APIs for PMSE Tools APIs

See manual for individual tools in chapter 8.

Appendix (Developer Manual) G

Regular Expressions Reference

正規表現は簡単です！

Jeffrey E.F. Friedl - Mastering Regular Expressions

The regular expression (regex) engine used by PMLs is that of the underlying Perl interpreter. This reference is intended to give you the minimum overview needed for doing some real-world matching applications. If you have perl installed on your system, the command

```
bash> perldoc perlre
```

will give you a more concise reference overview. Also, the command

```
bash> perldoc perlretut
```

will launch a good tutorial on perl regular expressions (regexes). Also make sure you consult the Book *Mastering Regular Expressions* by Jeffrey E.F. Friedl¹.

G.1 Basics

A regex is a string of characters. It is also a definition in a way that the regex defines what strings can possibly be matched by it. You could have a simple name as regular expression, e.g. `Alfons` which would match all strings **containing** the string `Alfons`, e.g. “Alfons Huber” (match), “Alfonso” (match), “alfons” (no match - regexes are case sensitive if not stated otherwise).

Of course pattern matching requirements are much more complex than simple substring-matching capabilities. Therefore regexes offer metacharacters, escape sequences and the capability to quantify them, define alternations etc.

¹see <http://regex.info/>

The most powerful features of the regular expressions that are available within the PMLS/PMSE software come from the perl-specific extensions to this concept. Namely the possibility to execute perl code within a regular expression.

G.2 Metacharacters

The following metacharacters have specific meanings

meta	description
\	Quote the next metacharacter. Quoting is important especially if you want to match characters that are considered metacharacters. Assume you want to match an opening bracket, you have to use \ (in your regex to achieve that.
.	Match any character (except newline). If you would like to match the words “ast” and “axt”, you could have a regex <code>a.t</code> - that would of course also match “art”, “abt” etc.
	Alternation. You can provide several regexes and use the “vertical pipe symbol” as delimiter. This will be interpreted as a logical or and if one of the alternatives matches, the whole alternation matches. e.g. <code>boo bah fuz</code> would match any string containing “boo” and/or “bah” and/or “fuz”.
()	Grouping is a very important regex concept. It allows you to handle any regular expression enclosed in grouping brackets as single entity. Assume our alternation example above if we would like to match “xbooy” and/or “xbahy” and/or “xfuzy”. Instead of prepending and appending “x” and “y” to each of these strings, we group the variable parts and move the invariants out: <code>x(boo bah fuz)y</code>
[]	Character class. In our “match any character” example above we used <code>a.t</code> to match “axt” and/or “ast”, but as we have seen, this also would match many other strings. This overmatching is often undesired. Now we could have specified just the acceptable characters by alternation: <code>a(s x)t</code> A shorter way to do so is <code>a[sx]t</code> The <code>^</code> character can be used to define a negated class. <code>[b]at</code> means “match .at except bat”.

Table G.1: Metacharacters in regular expressions.

G.3 Escape Sequences

Escape sequences form a large class in perl regexes. Often these escape sequences are classified into “escape sequences”, “character classes” and “assertions”. We will refrain from this further classification in our tables and show only the most important sequences to keep things easy.

sequence	description
----------	-------------

<code>\w</code>	Match a "word" character (alphanumeric plus "_") (complement to <code>\W</code>)
<code>\W</code>	Match a non-"word" character (complement to <code>\w</code>)
<code>\s</code>	Match a whitespace character (complement to <code>\S</code>)
<code>\S</code>	Match a non-whitespace character (complement to <code>\s</code>)
<code>\d</code>	Match a digit character
<code>\D</code>	Match a non-digit character
<code>\t</code>	Match the tabulator (TAB), which is a special case of the whitespace character.
<code>\n</code>	Match newline, which is a special case of the whitespace character.
<code>\1</code>	Backreference to a specific group. '1' may actually be any positive integer. This allows to match balanced quotes or similar applications.
<code>\b</code>	Match a word boundary.
<code>\B</code>	Match except at a word boundary.
<code>\A</code>	Match only at beginning of string.
<code>\Z</code>	Match only at end of string.

Table G.2: Escape sequences in regular expressions.

G.4 Quantifiers

quantifier	description
<code>*</code>	Match 0 or more times
<code>+</code>	Match 1 or more times
<code>?</code>	Match 1 or 0 times
<code>{n}</code>	Match exactly n times
<code>{n,}</code>	Match at least n times
<code>{n,m}</code>	Match at least n but not more than m times

Table G.3: Quantifiers in regular expressions.

By default, a quantified subpattern is "greedy", that is, it will match as many times as possible (given a particular starting location) while still allowing the rest of the pattern to match. If you want it to match the minimum number of times possible, follow the quantifier with a "?".

Given the string `axxxb`, the regex `(ax+x)` will return 'axxx' as captured pattern, whereas `(ax+?x)` will return 'axx'.

G.5 Examples

We will discuss some examples of regular expressions using only the aforementioned features. The examples are tailored to the input matching needs of the Discourse Engine.

G.5.1 Catching common Typos

Assume your logs show evidence, that users keep typing certain words wrong. Instead of having your matches fail, you would like to accept these mistypings too. For better readability, we omit the word boundaries in the regular expressions where they can be implicitly assumed.

“typo” or “tpyo” (but nothing else) → `t(yp|py)o`
“abzugsteuer” or “abzugssteuer”? → `abzugss?teuer`
“dass” or “das” (with boundary) → `\bdass?\b`
“petamen” or “petamen”? → `petame[mn]`
“schön”, “schöner”, “schönste” → `schön(er|ste)?`

The last example would also match words like “wunderschön” etc. Word boundaries are here to prevent this behaviour if not desired.

Appendix (Developer Manual) H

ISO639 Reference

As PMLs now supports the full ISO639-3 code range (3-letter code), there are nearly 7700 languages to be listed. This full list, however, would take here more than 28 pages in 4 columns and tiny characters. Instead we will provide only the ISO639-1 codes, their ISO639-3 equivalent and the english language name in the following lists for reference:

1. ISO639-1 (sortkey) ISO639-3 Name
2. ISO639-1 ISO639-3 (sortkey) Name
3. ISO639-1 ISO639-3 Name (sortkey)

For a more thorough coverage of the iso639-3 code, see the iso639 command or visit the iso639-3 authority webpages¹.

H.1 Sort by ISO639-1

aa aar Afar	co cos Corsican	gd gla Scottish Gaelic	jv jav Javanese
ab abk Abkhazian	cr cre Cree	gl glg Galician	ka kat Georgian
ae ave Avestan	cs ces Czech	gn grn Guarani	kg kon Kongo
af afr Afrikaans	cu chu Church Slavic	gu guj Gujarati	ki kik Kikuyu
ak aka Akan	cv chv Chuvash	gv glv Manx	kj kua Kuanyama
am amh Amharic	cy cym Welsh	ha hau Hausa	kk kaz Kazakh
an arg Aragonese	da dan Danish	he heb Hebrew	kl kal Kalaallisut
ar ara Arabic	de deu German	hi hin Hindi	km khm Central Khmer
as asm Assamese	dv div Dhivehi	ho hmo Hiri Motu	kn kan Kannada
av ava Avaric	dz dzo Dzongkha	hr hrv Croatian	ko kor Korean
ay aym Aymara	ee ewe Ewe	ht hat Haitian	kr kau Kanuri
az aze Azerbaijani	el ell Modern Greek (1453-)	hu hun Hungarian	ks kas Kashmiri
ba bak Bashkir	en eng English	hy hye Armenian	ku kur Kurdish
be bel Belarusian	eo epo Esperanto	hz her Herero	kv kom Komi
bg bul Bulgarian	es spa Spanish	ia ina Interlingua	kw cor Cornish
bi bis Bislama	et est Estonian	id ind Indonesian	ky kir Kirghiz
bm bam Bambara	eu eus Basque	ie ile Interlingue	la lat Latin
bn ben Bengali	fa fas Persian	ig ibo Igbo	lb ltz Luxembourgish
bo bod Tibetan	ff ful Fulah	ii iii Sichuan Yi	lg lug Ganda
br bre Breton	fi fin Finnish	ik ipk Inupiaq	li lim Limburgan
bs bos Bosnian	fj fij Fijian	io ido Ido	ln lin Lingala
ca cat Catalan	fo fao Faroese	is isl Icelandic	lo lao Lao
ce che Chechen	fr fra French	it ita Italian	lt lit Lithuanian
ch cha Chamorro	fy fry Western Frisian	iu iku Inuktitut	lu lub Luba-Katanga
	ga gle Irish	ja jpn Japanese	lv lav Latvian

¹<http://www.sil.org/iso639-3/download.asp>

mg mlg Malagasy	oc oci Occitan (post 1500)	si sin Sinhala	tr tur Turkish
mh mah Marshallese	oj oji Ojibwa	sk slk Slovak	ts tso Tsonga
mi mri Maori	om orm Oromo	sl slv Slovenian	tt tat Tatar
mk mkd Macedonian	or ori Oriya	sm smo Samoan	tw twi Twi
ml mal Malayalam	os oss Ossetian	sn sna Shona	ty tah Tahitian
mn mon Mongolian	pa pan Panjabi	so som Somali	ug uig Uighur
mo ron Romanian	pi pli Pali	sq sqi Albanian	uk ukr Ukrainian
mr mar Marathi	pl pol Polish	sr srp Serbian	ur urd Urdu
ms msa Malay	ps pus Pushto	ss ssw Swati	uz uzb Uzbek
mt mlt Maltese	pt por Portuguese	st sot Southern Sotho	ve ven Venda
my mya Burmese	qu que Quechua	su sun Sundanese	vi vie Vietnamese
na nau Nauru	rm roh Romansh	sv swe Swedish	vo vol Volapük
nb nob Norwegian Bokmål	rn run Rundi	sw swa Swahili	wa wln Walloon
nd nde North Ndebele	ro ron Romanian	ta tam Tamil	wo wol Wolof
ne nep Nepali	ru rus Russian	te tel Telugu	xh xho Xhosa
ng ndo Ndonga	rw kin Kinyarwanda	tg tgk Tajik	yi yid Yiddish
nl nld Dutch	sa san Sanskrit	th tha Thai	yo yor Yoruba
nn nno Norwegian Nynorsk	sc srd Sardinian	ti tir Tigrinya	za zha Zhuang
no nor Norwegian	sd snd Sindhi	tk tuk Turkmen	zh zho Chinese
nr nbl South Ndebele	se sme Northern Sami	tl tgl Tagalog	zu zul Zulu
nv nav Navajo	sg sag Sango	tn tsn Tswana	
ny nya Nyanja	sh hbs Serbo-Croatian	to ton Tonga (Tonga Islands)	

H.2 Sort by ISO639-3

aar aa Afar	ewe ee Ewe	kaz kk Kazakh	oji oj Ojibwa
abk ab Abkhazian	fao fo Faroese	khm km Central Khmer	ori or Oriya
afr af Afrikaans	fas fa Persian	kik ki Kikuyu	orm om Oromo
aka ak Akan	fij fj Fijian	kin rw Kinyarwanda	oss os Ossetian
amh am Amharic	fin fi Finnish	kir ky Kirghiz	pan pa Panjabi
ara ar Arabic	fra fr French	kom kv Komi	pli pi Pali
arg an Aragonese	fry fy Western Frisian	kon kg Kongo	pol pl Polish
asm as Assamese	ful ff Fulah	kor ko Korean	por pt Portuguese
ava av Avaric	gla gd Scottish Gaelic	kua kj Kuanyama	pus ps Pushto
ave ae Avestan	gle ga Irish	kur ku Kurdish	que qu Quechua
aym ay Aymara	glg gl Galician	lao lo Lao	roh rm Romansh
aze az Azerbaijani	glv gv Manx	lat la Latin	ron ro Romanian
bak ba Bashkir	grn gn Guarani	lav lv Latvian	run rn Rundi
bam bm Bambara	guj gu Gujarati	lim li Limburgan	rus ru Russian
bel be Belarusian	hat ht Haitian	lin ln Lingala	sag sg Sango
ben bn Bengali	hau ha Hausa	lit lt Lithuanian	san sa Sanskrit
bis bi Bislama	hbs sh Serbo-Croatian	ltz lb Luxembourgish	sin si Sinhala
bod bo Tibetan	heb he Hebrew	lub lu Luba-Katanga	slk sk Slovak
bos bs Bosnian	her hz Herero	lug lg Ganda	slv sl Slovenian
bre br Breton	hin hi Hindi	mah mh Marshallese	sme se Northern Sami
bul bg Bulgarian	hmo ho Hiri Motu	mal ml Malayalam	smo sm Samoan
cat ca Catalan	hrv hr Croatian	mar mr Marathi	sna sn Shona
ces cs Czech	hun hu Hungarian	mkd mk Macedonian	snd sd Sindhi
cha ch Chamorro	hye hy Armenian	mlg mg Malagasy	som so Somali
che ce Chechen	ibo ig Igbo	mlt mt Maltese	sot st Southern Sotho
chu cu Church Slavic	ido io Ido	mon mn Mongolian	spa es Spanish
chv cv Chuvash	iii ii Sichuan Yi	mri mi Maori	sqi sq Albanian
cor kw Cornish	iku iu Inuktitut	msa ms Malay	srd sc Sardinian
cos co Corsican	ile ie Interlingue	mya my Burmese	srp sr Serbian
cre cr Cree	ina ia Interlingua	nau na Nauru	ssw ss Swati
cym cy Welsh	ind id Indonesian	nav nv Navajo	sun su Sundanese
dan da Danish	ipk ik Inupiaq	nde nd North Ndebele	swa sw Swahili
deu de German	isl is Icelandic	ndo ng Ndonga	swe sv Swedish
div dv Dhivehi	ita it Italian	nep ne Nepali	tah ty Tahitian
dzo dz Dzongkha	jav jv Javanese	nld nl Dutch	tam ta Tamil
ell el Modern Greek (1453-)	jpn ja Japanese	nno nn Norwegian Nynorsk	tat tt Tatar
eng en English	kal kl Kalaallisut	nob nb Norwegian Bokmål	tel te Telugu
epo eo Esperanto	kan kn Kannada	nor no Norwegian	tgk tg Tajik
est et Estonian	kas ks Kashmiri	nya ny Nyanja	tgl ti Tagalog
eus eu Basque	kat ka Georgian	oci oc Occitan (post 1500)	tha th Thai
	kau kr Kanuri		tir ti Tigrinya

ton to Tonga (Tonga Islands)	uig ug Uighur	vol vo Volapük	zha za Zhuang
tsn tn Tswana	ukr uk Ukrainian	wln wa Walloon	zho zh Chinese
tso ts Tsonga	urd ur Urdu	wol wo Wolof	zul zu Zulu
tuk tk Turkmen	uzb uz Uzbek	xho xh Xhosa	
tur tr Turkish	ven ve Venda	yid yi Yiddish	
twi tw Twi	vie vi Vietnamese	yor yo Yoruba	

H.3 Sort by ISO639-Name

abk ab Abkhazian	fin fi Finnish	lub lu Luba-Katanga	iii ii Sichuan Yi
aar aa Afar	fra fr French	ltz lb Luxembourgish	snd sd Sindhi
afr af Afrikaans	ful ff Fulah	mkd mk Macedonian	sin si Sinhala
aka ak Akan	glg gl Galician	mlg mg Malagasy	slk sk Slovak
sqi sq Albanian	lug lg Ganda	msa ms Malay	slv sl Slovenian
amh am Amharic	kat ka Georgian	mal ml Malayalam	som so Somali
ara ar Arabic	deu de German	mlt mt Maltese	nbl nr South Ndebele
arg an Aragonese	grn gn Guarani	glv gv Manx	sot st Southern Sotho
hye hy Armenian	guj gu Gujarati	mri mi Maori	spa es Spanish
asm as Assamese	hat ht Haitian	mar mr Marathi	sun su Sundanese
ava av Avaric	hau ha Hausa	mah mh Marshallese	swa sw Swahili
ave ae Avestan	heb he Hebrew	ell el Modern Greek (1453-)	ssw ss Swati
aym ay Aymara	her hz Herero	mon mn Mongolian	swe sv Swedish
aze az Azerbaijani	hin hi Hindi	nau na Nauru	tgl tl Tagalog
bam bm Bambara	hmo ho Hiri Motu	nav nv Navajo	tah ty Tahitian
bak ba Bashkir	hun hu Hungarian	ndo ng Ndonga	tgk tg Tajik
eus eu Basque	isl is Icelandic	nep ne Nepali	tam ta Tamil
bel be Belarusian	ido io Ido	nde nd North Ndebele	tat tt Tatar
ben bn Bengali	ibo ig Igbo	sme se Northern Sami	tel te Telugu
bis bi Bislama	ind id Indonesian	nor no Norwegian	tha th Thai
bos bs Bosnian	ina ia Interlingua	nob nb Norwegian Bokmål	bod bo Tibetan
bre br Breton	ile ie Interlingue	nno nn Norwegian Nynorsk	tir ti Tigrinya
bul bg Bulgarian	iku iu Inuktitut	nya ny Nyanja	ton to Tonga (Tonga Islands)
mya my Burmese	ipk ik Inupiaq	oci oc Occitan (post 1500)	tso ts Tsonga
cat ca Catalan	gle ga Irish	oji oj Ojibwa	tsn tn Tswana
khm km Central Khmer	ita it Italian	ori or Oriya	tur tr Turkish
cha ch Chamorro	jpn ja Japanese	orm om Oromo	tuk tk Turkmen
che ce Chechen	jav jv Javanese	oss os Ossetian	twi tw Twi
zho zh Chinese	kal kl Kalaallisut	pli pi Pali	uig ug Uighur
chu cu Church Slavic	kan kn Kannada	pan pa Panjabi	ukr uk Ukrainian
chv cv Chuvash	kau kr Kanuri	fas fa Persian	urd ur Urdu
cor kw Cornish	kas ks Kashmiri	pol pl Polish	uzb uz Uzbek
cos co Corsican	kaz kk Kazakh	por pt Portuguese	ven ve Venda
cre cr Cree	kik ki Kikuyu	pus ps Pushto	vie vi Vietnamese
hrv hr Croatian	kin rw Kinyarwanda	que qu Quechua	vol vo Volapük
ces cs Czech	kir ky Kirghiz	ron ro Romanian	wln wa Walloon
dan da Danish	kom kv Komi	roh rm Romansh	cym cy Welsh
div dv Dhivehi	kon kg Kongo	run rn Rundi	fry fy Western Frisian
nld nl Dutch	kor ko Korean	rus ru Russian	wol wo Wolof
dzo dz Dzongkha	kua kj Kuanyama	smo sm Samoan	xho xh Xhosa
eng en English	kur ku Kurdish	sag sg Sango	yid yi Yiddish
epo eo Esperanto	lao lo Lao	san sa Sanskrit	yor yo Yoruba
est et Estonian	lat la Latin	srd sc Sardinian	zha za Zhuang
ewe ee Ewe	lav lv Latvian	gla gd Scottish Gaelic	zul zu Zulu
fao fo Faroese	lim li Limburgan	srp sr Serbian	
fij fj Fijian	lin ln Lingala	hbs sh Serbo-Croatian	
	lit lt Lithuanian	sna sn Shona	

Index

- [*, 55](#)
- [-iact, 146](#)
- [.ont-file](#)
 - [interpolating, 74](#)
 - [non-interpolating, 74](#)
- [.ont-files, 71](#)
- [.user_auth, 54](#)
- [?, 55, 181](#)
- [PMLS](#)
 - [deactivate, 19](#)
 - [delete, 19](#)
- [PMLS server process, 9](#)
- [32bit, 11](#)
- [64bit, 11](#)
- [abbreviations, 158](#)
- [add](#)
 - [entries, 64](#)
- [Arch, 11](#)
- [archive decompression, 153](#)
- [ARM, 12](#)
- [association measures, 124](#)
- [asterisk, 55](#)
- [autofs, 18](#)
- [automatic summarization, 60](#)
- [avatar, 67](#)
- [bash, 6](#)
- [binary tree, 165](#)
- [Boot Sequence, 46](#)
- [bot, 181](#)
- [bot_add, 182](#)
- [bot_del, 183](#)
- [bots_list, 183](#)
- [cat, 183](#)
- [categorization.pl](#)
 - [OPTIONS, 166](#)
 - [SYNOPSIS, 166](#)
- [categorization.pl, 166, 168](#)
- [cfg-client, 183](#)
- [cfg-get, 184](#)
- [cfg-reload, 185](#)
- [cfg-set, 185](#)
- [chatbot, 67](#)
- [Chomsky-hierarchy, 68](#)
- [class logic, 211](#)
- [CLI client, 9](#)
- [client](#)
 - [source only, 82](#)
- [clone](#)
 - [lexicon, 64](#)
- [clustering, 168](#)
 - [entropy, 169](#)
 - [purity, 169](#)
- [co-occurrences, 93, 161, 162](#)
- [collocation, 162](#)
- [command, 48](#)
- [command iteration, 133](#)
- [compat, 185](#)
- [connected](#)
 - [users, 53](#)
- [contingency tables, 124](#)
- [conversion](#)
 - [data, 55](#)
- [convert, 186](#)
- [cookbook, 153](#)
- [copy](#)
 - [lexicon, 64](#)
- [corpus, 91](#)
- [cp, 186](#)
- [CPAN, 11](#)
- [crash course, 153](#)
- [create](#)
 - [lexicon, 63](#)
 - [user, 53](#)
- [csh, 6](#)
- [cwd, 54](#)

- data
 - conversion, 55
- data acquisition, 100
- data library, 84
- deactivate
 - PMLS, 19
- Debian, 11, 12
- debug, 209
- deinstall
 - PMLS, 19
- deinstallation, 17
- delete
 - PMLS, 19
 - lexicon, 64
- delimiter, 147
- diacritics processor, 3
- dict, 186
- difference, 64
- directory
 - installation, 19
- directory structure, 84
- disconnect
 - user, 54
- discourse engine, 3
 - answer selection, 73
 - constant, 73
 - cycle, 73
 - random, 73
 - weighted random, 73
- Discourse Engine ChatBot, 241
 - \$Rev: 560 \$, 241
 - Bot Knowledge Processing, 242
 - Bot State Processing, 242
 - Input Processing, 241
 - METHODS, 241
 - Top-Level Functionality, 241
- distance, 172
 - measures, 172
 - normalization, 172
- distance measures, 112
- doc, 187
- document classification, 60
- document summarization, 60
- downgrading, 17
- download, 54
 - file, 54
- DVD, 9
 - installation, 19
- e-cpy, 187
- e-del, 188
- e-describe, 188
- e-get_arg, 188
- e-get_common, 188
- e-get_thull, 189
- e-mov, 189
- e-new, 189
- e-show, 189
- e-translate, 190
- encode, 190
- ending
 - session, 51
- entry, 63
 - add, 64
 - key, 63
 - meaning, 63
- environment variables, 132
- estart, 9
- European Medicines Agency, 152
- Fedora, 12
- file
 - delete, 55
 - download, 54
 - listing, 55
 - manipulation, 55
 - rename, 55
 - show, 55
 - upload, 54
- file modifications, 153
- file statistics, 88
- file system, 54
- find, 65
 - approximate, 65
 - entries, 65
 - exact, 65
 - regular expression, 65
- format conversion, 104, 120
- g-cpan, 13
- Gentoo, 11–13
- get, 209
- get_context_probability, 161
- get_subgrams, 160
- globbing, 55
- GraphViz, 93, 168
- h, 190

- hardware
 - requirements, 11
- help, 87, 190
- help-syntax, 191
- heredocs
 - interpolating, 74
 - non-interpolating, 74
- histogram, 172
 - filtering, 176
 - output limit, 176
 - sorting, 176
- home
 - directory, 53, 54
- inference
 - semantic, 1
- info
 - output, 9
- information
 - installation, 17
- INI file, 100, 157
- INI-Files, 71
- INI-style, 27
- installation, 17
 - directory, 19
 - DVD, 19
- interactive, 146, 148
- interpolation, 74
- intersection, 64
- ISO 639-3, 178
- ISO-Image, 19
- key-words, 124
- keyword, 48
- l-cpy, 192
- l-del, 192
- l-info, 192
- l-list, 193
- l-load, 193
- l-mov, 193
- l-new, 194
- l-reversify, 194
- l-save, 194
- language identification, 3, 81
- lc-chart, 195
- lc-cpy, 195
- lc-del, 195
- lc-info, 195
- lc-list, 196
- lc-mov, 196
- lc-new, 196
- lcmp, 196
- lcover, 196
- lemmatizer, 3
- lexical operations, 3
- lexicon, 48, 63
 - clone, 64
 - copy, 64
 - create, 63
 - delete, 64
 - load, 63
 - move, 64
 - rename, 64
 - save, 63
 - set operations, 64
- li, 197
- li-llm, 198
- list
 - users, 53
- load
 - lexicon, 63
- loopback, 19
- ls, 198
- machine translation, 3
- macros, 84, 176
- Mak1s1l, 176
- make, 81
- makepp, 81
- management
 - user, 53
- Manual Startup, 45
- MI-score, 124, 150
- morphology, 68
- mosaic plot, 172
- move
 - lexicon, 64
- mt, 198
- MULTEX-east, 60
- multilingual library, 106
- mv, 199
- N-ary
 - tree, 67
- n-gram, 124
- n-grams, 153, 170, 173
- named parameters, 48

- natural language
 - analysis, 1
 - generation, 1
 - processing, 1
 - understanding, 1
- negative lookbehind, 156
- NFS, 19
- NLA, 1
- nla-list, 199
- nla-morph, 199
- nla-number, 200
- nla-tag, 200
- NLG, 1
- nlg-list, 201
- nlg-morph, 201
- nlg-number, 201
- NLP, 3
- nlp-m_test, 202
- NLU, 3
- Novell SLES, 12
- numeral, 69

- ontology, 71
- OpenBSD 4.1, 81
- openSUSE, 11–13
- options, 87

- P_bsd, 85, 88
- P_cct, 91
- P_cop, 85, 93, 163
- P_csp, 86, 146
- P_daf, 100, 152
- P_dmf, 86, 104, 152
- P_dmp, 86, 112
- P_dvf, 86, 120, 167
- P_fdt, 86
- P_gnp, 86, 124, 148, 160
- P_help, 86, 132
- P_ici, 86, 133
- P_rer, 86, 138, 155, 158, 160
- P_trt, 86, 141, 155, 178
- P_vls, 86, 143
- parallel, 84
- parallelisation, 133
- parameters
 - named, 48
 - positional, 48
- parsing, 1
- part-of-speech, 60

- Perl, 11
- perl, 12
- perl interpreter, 12
- perl modules, 12
- perm_add, 202
- perm_del, 202
- permissions
 - user, 54
- perms_list, 203
- PID file, 41
- PMSE, 83
- PMSE Cookbook, 153
- PMSE objects, 120, 167
- PMSE root, 84
- PMSE Tutorial, 145
- PMTS (tagset), 91
- POS
 - tagger, 60
- positional parameters, 48
- precondition, 76
- probability of neighbors, 161
- purge, 203
 - user, 54
- put, 209

- question mark, 55
- quit, 209

- rank, 124
- RedHat, 12, 13
 - Enterprise Server 4, 81
- regex
 - escape sequences, 432
 - metacharacters, 432
 - quantifiers, 433
- regexp replacement, 138
- regular expression, 138
 - escape sequences, 432
 - metacharacters, 432
 - quantifiers, 433
- regular expressions, 68
- remove
 - PMLS, 19
 - user, 53
- rename
 - lexicon, 64
- repository, 63
- requirements
 - hardware, 11

- software, 12
 - OS, 12
 - Perl, 12
- RHES, 12
- RHES4, 81
- riter, 203
- rm, 204
- RPM, 13
- runs, 165
- save
 - lexicon, 63
- say, 204
- scripting environment, 83
- segmenter, 155
- segmenter for Czech, 157
- Semantic Lexicon, 229
 - \$Rev: 560 \$, 229
 - Entry, 237
 - Glossary, 229
 - Helper Functions (no methods), 239
 - Inference, 239
 - Internal/Low-Level Access/Retrieve, 232
 - Lexicon Data, 233
 - Lexicon Meta, 235
 - Meaning, 237
 - Meanings/Meaning, 237
 - METHODS, 232
 - Relation, 237
 - Syntax, 238
- semicolon, 48
- sentence segmentation, 155, 176
- server
 - shutdown, 45
 - startup, 45
- session
 - ending, 51
- set operations, 64
 - lexicon, 64
- shutdown, 45, 205
- single-user, 51
- SLES, 11
- SMB, 19
- software
 - requirements, 12
 - OS, 12
 - Perl, 12
- Solaris 10, 81
- square bracket, 55
- startup, 45
- stemmer, 3
- STerm, 157
- sub word n-grams, 158
- Subversion, 221
- SUSE, 12
- SUSE 10.1, 81
- SUSE 9.2, 81
- SVG, 165, 170, 174
- SVN, 221
- SY, 67
- symmetric difference, 64
- syntactic
 - information, 67
- sys-info, 205
- system
 - control, 51
 - information, 51
- T-score, 124
- tagger, 3, 60
- tagset, 91
- tagset conversion, 91
- tape, 19
- tcsh, 6
- text
 - basic functions, 57
 - text categorization, 3, 60, 152, 154, 165
 - text classification, 60
 - text formatting, 141
 - text summarization, 3, 60
- Text::NSP, 162
- tokenization, 148
- tokenizer, 147
- TR, 67
- translation
 - information, 67
- tree
 - N-ary, 67
- trie, 23, 152
- turing-complete, 76
- tutorial, 145
- txt-info, 206
- Ubuntu, 12
- Ubuntu 7.04, 81
- union, 64
- UNIX, 83

- upgrading, 17
- upload, 54
 - file, 54
- USB-stick, 19
- user, 210
 - concurrent, 53
 - create, 53
 - disconnect, 54
 - management, 53
 - named, 53
 - permissions, 54
 - purge, 54
 - real, 53
 - remove, 53
 - virtual, 53
- user_add, 206
- user_del, 206
- users
 - connected, 53
 - list of, 53
- users_list, 206
- UTF-8, 26

- v-cchk, 206
- v-get_clique, 207
- v-info, 207
- v-query, 208
- variable interpolation, 74
- vars.ont, 74
- version control, 221
- virtual machine, 11
- visualization, 120, 167
 - binary tree, 168
 - contingency tables, 170
 - distances, 172
 - file statistics, 172
 - histogram, 172
- VM, 11
- Volume-ID, 18

- who, 208
- whoami, 208
- Wikimedia processing, 104
- wildcards, 55
- wordlist, 154
- working directory
 - current, 54

Go on, get out - last words are for fools who haven't said enough.
Karl Marx

End of PMLS Manual (as of October 12, 2025)