



Middle East Technical University Northern Cyprus Campus
Computer Engineering Program

CNG491 Computer Engineering Design I

Markopy Documentation

Ata Hakçıl - 2243467
Osman Ömer Yıldıztugay - 1921956
Celal Sahir Çetiner - 1755420
Yunus Emre Yılmaz - 2243723

Supervised by
Assoc. Prof. Dr. Okan Topçu

0.7.0 Documentation

1 Markopy	3
1.1 About The Project	4
1.1.1 Possible Use Cases	4
1.1.2 Getting Started	4
1.1.3 Releases	4
1.2 Building	5
1.3 Prerequisites	5
1.3.1 General prequisites	5
1.3.1.1 Linux	5
1.3.1.2 Windows	5
1.3.2 MarkovModel	5
1.3.3 MarkovAPI	5
1.3.4 MarkovAPICLI	5
1.3.5 Markopy	5
1.3.6 CudaMarkovAPI	6
1.3.7 CudaMarkopy	6
1.3.8 MarkovPasswordsGUI	6
1.3.9 CMake Configuration	6
1.3.9.1 Build everything	6
1.3.9.2 Build libraries only	6
1.3.9.3 Build CUDA-accelerated libraries	6
1.3.9.4 Build python module & libraries	6
1.3.9.5 Build CUDA accelerated python module	6
1.3.9.6 Build CUDA accelerated python module and the GUI	6
1.3.10 Installing Dependencies	7
1.4 Known Common issues	7
1.4.1 Linux	7
1.4.1.1 Markopy - Python.h - Not found	7
1.4.1.2 Markopy/MarkovAPI - *.so not found, or other library related issues when building	7
1.4.2 Windows	8
1.4.2.1 Boost - Bootstrap.bat "ctype.h" not found	8
1.4.2.2 Cannot open file "*.lib"	8
1.4.2.3 Python.h not found	8
1.5 Contributing	8
1.6 Contact	8
2 NVSight Report	9
2.0.1 Overview	9
2.0.2 Session	9
2.0.3 PTX	9
3 Deprecated List	11

4 Namespace Index	13
4.1 Namespace List	13
5 Hierarchical Index	15
5.1 Class Hierarchy	15
6 Class Index	19
6.1 Class List	19
7 File Index	23
7.1 File List	23
8 Namespace Documentation	25
8.1 base Namespace Reference	25
8.2 cudamarkopy Namespace Reference	25
8.2.1 Variable Documentation	25
8.2.1.1 markopy	25
8.2.1.2 mp	25
8.2.1.3 spec	25
8.3 cudammx Namespace Reference	25
8.3.1 Variable Documentation	25
8.3.1.1 cudamarkopy	25
8.3.1.2 ext	26
8.3.1.3 markopy	26
8.3.1.4 mp	26
8.3.1.5 spec	26
8.4 importer Namespace Reference	26
8.4.1 Function Documentation	26
8.4.1.1 import_markopy()	26
8.5 markopy Namespace Reference	27
8.5.1 Variable Documentation	27
8.5.1.1 ext	27
8.5.1.2 markopy	27
8.5.1.3 mp	27
8.5.1.4 spec	27
8.6 Markov Namespace Reference	27
8.6.1 Detailed Description	28
8.7 Markov::API Namespace Reference	28
8.7.1 Detailed Description	28
8.8 Markov::API::CLI Namespace Reference	28
8.8.1 Detailed Description	29
8.8.2 Typedef Documentation	29
8.8.2.1 ProgramOptions	29
8.8.3 Function Documentation	29

8.8.3.1 operator<<()	29
8.9 Markov::API::Concurrency Namespace Reference	29
8.9.1 Detailed Description	29
8.10 Markov::API::CUDA Namespace Reference	29
8.10.1 Detailed Description	30
8.10.2 Function Documentation	30
8.10.2.1 FastRandomWalkCUDAKernel()	30
8.10.2.2 strchr()	31
8.11 Markov::API::CUDA::Random Namespace Reference	32
8.11.1 Detailed Description	32
8.11.2 Function Documentation	32
8.11.2.1 devrandom()	32
8.12 Markov::GUI Namespace Reference	32
8.12.1 Detailed Description	33
8.13 Markov::Markopy Namespace Reference	33
8.13.1 Detailed Description	33
8.13.2 Function Documentation	33
8.13.2.1 BOOST_PYTHON_MODULE()	33
8.14 Markov::Markopy::CUDA Namespace Reference	34
8.14.1 Detailed Description	34
8.14.2 Function Documentation	34
8.14.2.1 BOOST_PYTHON_MODULE()	34
8.15 Markov::Random Namespace Reference	35
8.15.1 Detailed Description	35
8.16 mm Namespace Reference	35
8.16.1 Variable Documentation	35
8.16.1.1 markopy	35
8.17 mmx Namespace Reference	35
8.17.1 Variable Documentation	36
8.17.1.1 markopy	36
8.17.1.2 mp	36
8.18 model_2gram Namespace Reference	36
8.18.1 Detailed Description	36
8.18.2 Variable Documentation	36
8.18.2.1 alphabet	36
8.18.2.2 f	36
8.19 mp Namespace Reference	36
8.19.1 Variable Documentation	36
8.19.1.1 markopy	37
8.19.1.2 mp	37
8.20 Python.CudaMarkopy Namespace Reference	37
8.20.1 Detailed Description	37

8.21 Python.Markopy Namespace Reference	37
8.21.1 Detailed Description	37
8.22 random_model Namespace Reference	38
8.22.1 Detailed Description	38
8.22.2 Variable Documentation	38
8.22.2.1 alphabet	38
8.22.2.2 f	38
8.23 Testing Namespace Reference	38
8.23.1 Detailed Description	38
8.24 Testing::MarkovModel Namespace Reference	38
8.24.1 Detailed Description	39
8.24.2 Function Documentation	39
8.24.2.1 TEST_CLASS() [1/3]	39
8.24.2.2 TEST_CLASS() [2/3]	40
8.24.2.3 TEST_CLASS() [3/3]	41
8.25 Testing::MarkovPasswords Namespace Reference	42
8.25.1 Detailed Description	42
8.26 Testing::MVP Namespace Reference	42
8.26.1 Detailed Description	42
8.27 Testing::MVP::MarkovModel Namespace Reference	42
8.27.1 Detailed Description	43
8.27.2 Function Documentation	43
8.27.2.1 TEST_CLASS() [1/3]	43
8.27.2.2 TEST_CLASS() [2/3]	44
8.27.2.3 TEST_CLASS() [3/3]	45
8.28 Testing::MVP::MarkovPasswords Namespace Reference	48
8.28.1 Detailed Description	48
8.28.2 Function Documentation	48
8.28.2.1 TEST_CLASS()	48
9 Class Documentation	51
9.1 Markov::API::CLI::_programOptions Struct Reference	51
9.1.1 Detailed Description	52
9.1.2 Member Data Documentation	52
9.1.2.1 bExport	52
9.1.2.2 bFailure	52
9.1.2.3 bImport	52
9.1.2.4 datasetname	52
9.1.2.5 exportname	53
9.1.2.6 generateN	53
9.1.2.7 importname	53
9.1.2.8 outputfilename	53

9.1.2.9 separator	53
9.1.2.10 wordlistname	53
9.2 Markov::GUI::about Class Reference	54
9.2.1 Detailed Description	55
9.2.2 Constructor & Destructor Documentation	55
9.2.2.1 about()	55
9.2.3 Member Data Documentation	55
9.2.3.1 ui	55
9.3 Python.Markopy.AbstractGenerationModelCLI Class Reference	55
9.3.1 Detailed Description	58
9.3.2 Member Function Documentation	58
9.3.2.1 __generate()	58
9.3.2.2 add_arguments()	59
9.3.2.3 check_corpus_path()	60
9.3.2.4 check_export_path()	60
9.3.2.5 check_import_path()	61
9.3.2.6 export()	61
9.3.2.7 generate()	62
9.3.2.8 help()	63
9.3.2.9 import_model()	63
9.3.2.10 init_post_arguments()	64
9.3.2.11 parse()	65
9.3.2.12 parse_arguments()	65
9.3.2.13 process()	66
9.3.2.14 train()	67
9.3.3 Member Data Documentation	68
9.3.3.1 args	68
9.3.3.2 model	69
9.3.3.3 parser	69
9.3.3.4 print_help	69
9.4 Python.Markopy.AbstractTrainingModelCLI Class Reference	69
9.4.1 Detailed Description	73
9.4.2 Member Function Documentation	73
9.4.2.1 __generate()	73
9.4.2.2 add_arguments()	73
9.4.2.3 check_corpus_path() [1/2]	74
9.4.2.4 check_corpus_path() [2/2]	74
9.4.2.5 check_export_path() [1/2]	75
9.4.2.6 check_export_path() [2/2]	75
9.4.2.7 check_import_path() [1/2]	76
9.4.2.8 check_import_path() [2/2]	76
9.4.2.9 export() [1/2]	77

9.4.2.10 <code>export()</code> [2/2]	78
9.4.2.11 <code>generate()</code> [1/2]	78
9.4.2.12 <code>generate()</code> [2/2]	79
9.4.2.13 <code>help()</code> [1/2]	80
9.4.2.14 <code>help()</code> [2/2]	81
9.4.2.15 <code>import_model()</code> [1/2]	81
9.4.2.16 <code>import_model()</code> [2/2]	82
9.4.2.17 <code>init_post_arguments()</code> [1/2]	83
9.4.2.18 <code>init_post_arguments()</code> [2/2]	83
9.4.2.19 <code>parse()</code> [1/2]	84
9.4.2.20 <code>parse()</code> [2/2]	85
9.4.2.21 <code>parse_arguments()</code> [1/2]	85
9.4.2.22 <code>parse_arguments()</code> [2/2]	86
9.4.2.23 <code>process()</code> [1/2]	86
9.4.2.24 <code>process()</code> [2/2]	87
9.4.2.25 <code>train()</code> [1/2]	89
9.4.2.26 <code>train()</code> [2/2]	90
9.4.3 Member Data Documentation	91
9.4.3.1 <code>args</code> [1/2]	91
9.4.3.2 <code>args</code> [2/2]	92
9.4.3.3 <code>model</code> [1/2]	92
9.4.3.4 <code>model</code> [2/2]	92
9.4.3.5 <code>parser</code> [1/2]	92
9.4.3.6 <code>parser</code> [2/2]	92
9.4.3.7 <code>print_help</code> [1/2]	92
9.4.3.8 <code>print_help</code> [2/2]	93
9.5 <code>Markov::API::CLI::Argparse</code> Class Reference	93
9.5.1 Detailed Description	95
9.5.2 Constructor & Destructor Documentation	95
9.5.2.1 <code>Argparse()</code> [1/2]	95
9.5.2.2 <code>Argparse()</code> [2/2]	95
9.5.3 Member Function Documentation	97
9.5.3.1 <code>getProgramOptions()</code>	97
9.5.3.2 <code>help()</code>	97
9.5.3.3 <code>parse()</code>	98
9.5.3.4 <code>setProgramOptions()</code>	98
9.5.4 Member Data Documentation	99
9.5.4.1 <code>po</code>	99
9.6 <code>Python.Markopy.BaseCLI</code> Class Reference	99
9.6.1 Detailed Description	102
9.6.2 Constructor & Destructor Documentation	102
9.6.2.1 <code>__init__()</code>	102

9.6.3 Member Function Documentation	103
9.6.3.1 <code>_generate()</code>	103
9.6.3.2 <code>add_arguments()</code>	104
9.6.3.3 <code>check_corpus_path()</code>	104
9.6.3.4 <code>check_export_path()</code>	105
9.6.3.5 <code>check_import_path()</code>	105
9.6.3.6 <code>export()</code>	106
9.6.3.7 <code>generate()</code>	106
9.6.3.8 <code>help()</code>	107
9.6.3.9 <code>import_model()</code>	108
9.6.3.10 <code>init_post_arguments()</code>	109
9.6.3.11 <code>parse()</code>	109
9.6.3.12 <code>parse_arguments()</code>	110
9.6.3.13 <code>process()</code>	110
9.6.3.14 <code>train()</code>	112
9.6.4 Member Data Documentation	113
9.6.4.1 <code>args</code>	113
9.6.4.2 <code>model</code>	113
9.6.4.3 <code>parser</code>	113
9.6.4.4 <code>print_help</code>	113
9.7 <code>Markov::GUI::CLI</code> Class Reference	114
9.7.1 Detailed Description	115
9.7.2 Constructor & Destructor Documentation	115
9.7.2.1 <code>CLI()</code>	115
9.7.3 Member Function Documentation	116
9.7.3.1 <code>about</code>	116
9.7.3.2 <code>start</code>	116
9.7.3.3 <code>statistics</code>	117
9.7.4 Member Data Documentation	117
9.7.4.1 <code>ui</code>	117
9.8 <code>Markov::API::CUDA::CUDADeviceController</code> Class Reference	117
9.8.1 Detailed Description	119
9.8.2 Member Function Documentation	120
9.8.2.1 <code>CudaCheckNotifyErr()</code>	120
9.8.2.2 <code>CudaMalloc2DToFlat()</code>	120
9.8.2.3 <code>CudaMemcpy2DToFlat()</code>	121
9.8.2.4 <code>CudaMigrate2DFlat()</code>	122
9.8.2.5 <code>ListCudaDevices()</code>	123
9.9 <code>Python.CudaMarkopy.CudaMarkopyCLI</code> Class Reference	124
9.9.1 Detailed Description	134
9.9.2 Constructor & Destructor Documentation	134
9.9.2.1 <code>__init__()</code>	134

9.9.3 Member Function Documentation	134
9.9.3.1 <code>_generate()</code>	134
9.9.3.2 <code>add_arguments() [1/2]</code>	135
9.9.3.3 <code>add_arguments() [2/2]</code>	136
9.9.3.4 <code>AdjustEdge() [1/4]</code>	136
9.9.3.5 <code>AdjustEdge() [2/4]</code>	138
9.9.3.6 <code>AdjustEdge() [3/4]</code>	139
9.9.3.7 <code>AdjustEdge() [4/4]</code>	139
9.9.3.8 <code>AllocVRAMOutputBuffer()</code>	140
9.9.3.9 <code>Buff() [1/4]</code>	140
9.9.3.10 <code>Buff() [2/4]</code>	141
9.9.3.11 <code>Buff() [3/4]</code>	143
9.9.3.12 <code>Buff() [4/4]</code>	144
9.9.3.13 <code>check_corpus_path() [1/6]</code>	145
9.9.3.14 <code>check_corpus_path() [2/6]</code>	146
9.9.3.15 <code>check_corpus_path() [3/6]</code>	146
9.9.3.16 <code>check_corpus_path() [4/6]</code>	147
9.9.3.17 <code>check_corpus_path() [5/6]</code>	147
9.9.3.18 <code>check_corpus_path() [6/6]</code>	148
9.9.3.19 <code>check_export_path() [1/6]</code>	148
9.9.3.20 <code>check_export_path() [2/6]</code>	149
9.9.3.21 <code>check_export_path() [3/6]</code>	149
9.9.3.22 <code>check_export_path() [4/6]</code>	150
9.9.3.23 <code>check_export_path() [5/6]</code>	150
9.9.3.24 <code>check_export_path() [6/6]</code>	151
9.9.3.25 <code>check_import_path() [1/6]</code>	151
9.9.3.26 <code>check_import_path() [2/6]</code>	152
9.9.3.27 <code>check_import_path() [3/6]</code>	152
9.9.3.28 <code>check_import_path() [4/6]</code>	153
9.9.3.29 <code>check_import_path() [5/6]</code>	153
9.9.3.30 <code>check_import_path() [6/6]</code>	154
9.9.3.31 <code>ConstructMatrix() [1/3]</code>	154
9.9.3.32 <code>ConstructMatrix() [2/3]</code>	156
9.9.3.33 <code>ConstructMatrix() [3/3]</code>	158
9.9.3.34 <code>CudaCheckNotifyErr()</code>	159
9.9.3.35 <code>CudaMalloc2DToFlat()</code>	160
9.9.3.36 <code>CudaMemcpy2DToFlat()</code>	161
9.9.3.37 <code>CudaMigrate2DFlat()</code>	162
9.9.3.38 <code>DeallocateMatrix() [1/3]</code>	163
9.9.3.39 <code>DeallocateMatrix() [2/3]</code>	164
9.9.3.40 <code>DeallocateMatrix() [3/3]</code>	164
9.9.3.41 <code>DumpJSON() [1/3]</code>	165

9.9.3.42 DumpJSON() [2/3]	166
9.9.3.43 DumpJSON() [3/3]	167
9.9.3.44 Edges() [1/4]	168
9.9.3.45 Edges() [2/4]	168
9.9.3.46 Edges() [3/4]	169
9.9.3.47 Edges() [4/4]	169
9.9.3.48 Export() [1/9]	169
9.9.3.49 Export() [2/9]	169
9.9.3.50 Export() [3/9]	170
9.9.3.51 Export() [4/9]	170
9.9.3.52 export() [1/6]	170
9.9.3.53 export() [2/6]	171
9.9.3.54 export() [3/6]	172
9.9.3.55 export() [4/6]	172
9.9.3.56 export() [5/6]	173
9.9.3.57 export() [6/6]	174
9.9.3.58 Export() [5/9]	174
9.9.3.59 Export() [6/9]	175
9.9.3.60 Export() [7/9]	175
9.9.3.61 Export() [8/9]	176
9.9.3.62 Export() [9/9]	176
9.9.3.63 FastRandomWalk() [1/9]	176
9.9.3.64 FastRandomWalk() [2/9]	177
9.9.3.65 FastRandomWalk() [3/9]	177
9.9.3.66 FastRandomWalk() [4/9]	179
9.9.3.67 FastRandomWalk() [5/9]	180
9.9.3.68 FastRandomWalk() [6/9]	181
9.9.3.69 FastRandomWalk() [7/9]	182
9.9.3.70 FastRandomWalk() [8/9]	183
9.9.3.71 FastRandomWalk() [9/9]	184
9.9.3.72 FastRandomWalkPartition() [1/3]	185
9.9.3.73 FastRandomWalkPartition() [2/3]	186
9.9.3.74 FastRandomWalkPartition() [3/3]	187
9.9.3.75 FastRandomWalkThread() [1/3]	189
9.9.3.76 FastRandomWalkThread() [2/3]	190
9.9.3.77 FastRandomWalkThread() [3/3]	191
9.9.3.78 FlattenMatrix()	193
9.9.3.79 GatherAsyncKernelOutput()	193
9.9.3.80 Generate() [1/5]	194
9.9.3.81 generate() [1/6]	194
9.9.3.82 generate() [2/6]	195
9.9.3.83 generate() [3/6]	196

9.9.3.84 generate() [4/6]	197
9.9.3.85 generate() [5/6]	198
9.9.3.86 generate() [6/6]	199
9.9.3.87 Generate() [2/5]	200
9.9.3.88 Generate() [3/5]	201
9.9.3.89 Generate() [4/5]	202
9.9.3.90 Generate() [5/5]	203
9.9.3.91 GenerateThread()	205
9.9.3.92 help()	206
9.9.3.93 Import() [1/8]	206
9.9.3.94 Import() [2/8]	207
9.9.3.95 Import() [3/8]	208
9.9.3.96 Import() [4/8]	209
9.9.3.97 Import() [5/8]	210
9.9.3.98 Import() [6/8]	211
9.9.3.99 Import() [7/8]	212
9.9.3.100 Import() [8/8]	213
9.9.3.101 import_model() [1/6]	213
9.9.3.102 import_model() [2/6]	214
9.9.3.103 import_model() [3/6]	215
9.9.3.104 import_model() [4/6]	216
9.9.3.105 import_model() [5/6]	217
9.9.3.106 import_model() [6/6]	218
9.9.3.107 init_post_arguments() [1/2]	219
9.9.3.108 init_post_arguments() [2/2]	219
9.9.3.109 LaunchAsyncKernel()	220
9.9.3.110 ListCudaDevices()	220
9.9.3.111 MigrateMatrix()	220
9.9.3.112 Nodes() [1/4]	221
9.9.3.113 Nodes() [2/4]	221
9.9.3.114 Nodes() [3/4]	221
9.9.3.115 Nodes() [4/4]	222
9.9.3.116 OpenDatasetFile() [1/4]	222
9.9.3.117 OpenDatasetFile() [2/4]	222
9.9.3.118 OpenDatasetFile() [3/4]	223
9.9.3.119 OpenDatasetFile() [4/4]	224
9.9.3.120 OptimizeEdgeOrder() [1/4]	224
9.9.3.121 OptimizeEdgeOrder() [2/4]	224
9.9.3.122 OptimizeEdgeOrder() [3/4]	225
9.9.3.123 OptimizeEdgeOrder() [4/4]	225
9.9.3.124 parse()	225
9.9.3.125 parse_arguments() [1/6]	225

9.9.3.126 parse_arguments() [2/6]	226
9.9.3.127 parse_arguments() [3/6]	226
9.9.3.128 parse_arguments() [4/6]	227
9.9.3.129 parse_arguments() [5/6]	227
9.9.3.130 parse_arguments() [6/6]	228
9.9.3.131 parse_fail()	228
9.9.3.132 prepKernelMemoryChannel()	228
9.9.3.133 process()	229
9.9.3.134 RandomWalk() [1/4]	229
9.9.3.135 RandomWalk() [2/4]	230
9.9.3.136 RandomWalk() [3/4]	232
9.9.3.137 RandomWalk() [4/4]	234
9.9.3.138 Save() [1/4]	236
9.9.3.139 Save() [2/4]	237
9.9.3.140 Save() [3/4]	237
9.9.3.141 Save() [4/4]	238
9.9.3.142 StarterNode() [1/4]	239
9.9.3.143 StarterNode() [2/4]	239
9.9.3.144 StarterNode() [3/4]	239
9.9.3.145 StarterNode() [4/4]	239
9.9.3.146 stub()	239
9.9.3.147 Train() [1/4]	240
9.9.3.148 Train() [2/4]	241
9.9.3.149 Train() [3/4]	242
9.9.3.150 train() [1/6]	243
9.9.3.151 train() [2/6]	244
9.9.3.152 train() [3/6]	245
9.9.3.153 train() [4/6]	247
9.9.3.154 train() [5/6]	248
9.9.3.155 train() [6/6]	251
9.9.3.156 Train() [4/4]	252
9.9.3.157 TrainThread()	253
9.9.4 Member Data Documentation	253
9.9.4.1 alternatingKernels	254
9.9.4.2 args	254
9.9.4.3 blInfinite	254
9.9.4.4 cli	254
9.9.4.5 cudaBlocks	254
9.9.4.6 cudaGridSize	254
9.9.4.7 cudaMemPerGrid	254
9.9.4.8 cudaPerKernelAllocationSize	254
9.9.4.9 cudastreams	255

9.9.4.10 cudaThreads	255
9.9.4.11 datasetFile	255
9.9.4.12 device_edgeMatrix	255
9.9.4.13 device_matrixIndex	255
9.9.4.14 device_outputBuffer	255
9.9.4.15 device_seeds	255
9.9.4.16 device_totalEdgeWeights	255
9.9.4.17 device_valueMatrix	255
9.9.4.18 edgeMatrix [1/3]	256
9.9.4.19 edgeMatrix [2/3]	256
9.9.4.20 edgeMatrix [3/3]	256
9.9.4.21 edges	256
9.9.4.22 fileIO [1/2]	256
9.9.4.23 fileIO [2/2]	256
9.9.4.24 flatEdgeMatrix	256
9.9.4.25 flatValueMatrix	257
9.9.4.26 iterationsPerKernelThread	257
9.9.4.27 matrixIndex [1/3]	257
9.9.4.28 matrixIndex [2/3]	257
9.9.4.29 matrixIndex [3/3]	257
9.9.4.30 matrixSize [1/3]	257
9.9.4.31 matrixSize [2/3]	257
9.9.4.32 matrixSize [3/3]	258
9.9.4.33 model [1/4]	258
9.9.4.34 model [2/4]	258
9.9.4.35 model [3/4]	258
9.9.4.36 model [4/4]	258
9.9.4.37 modelSavefile	258
9.9.4.38 nodes	258
9.9.4.39 numberOfPartitions	259
9.9.4.40 outputBuffer	259
9.9.4.41 outputFile	259
9.9.4.42 parser [1/6]	259
9.9.4.43 parser [2/6]	259
9.9.4.44 parser [3/6]	259
9.9.4.45 parser [4/6]	259
9.9.4.46 parser [5/6]	260
9.9.4.47 parser [6/6]	260
9.9.4.48 print_help [1/6]	260
9.9.4.49 print_help [2/6]	260
9.9.4.50 print_help [3/6]	260
9.9.4.51 print_help [4/6]	260

9.9.4.52 print_help [5/6]	260
9.9.4.53 print_help [6/6]	261
9.9.4.54 ready [1/3]	261
9.9.4.55 ready [2/3]	261
9.9.4.56 ready [3/3]	261
9.9.4.57 starterNode	261
9.9.4.58 totalEdgeWeights [1/3]	261
9.9.4.59 totalEdgeWeights [2/3]	261
9.9.4.60 totalEdgeWeights [3/3]	262
9.9.4.61 totalOutputPerKernel	262
9.9.4.62 totalOutputPerSync	262
9.9.4.63 valueMatrix [1/3]	262
9.9.4.64 valueMatrix [2/3]	262
9.9.4.65 valueMatrix [3/3]	262
9.10 Markov::API::CUDA::CUDAModelMatrix Class Reference	262
9.10.1 Detailed Description	267
9.10.2 Member Function Documentation	268
9.10.2.1 AdjustEdge()	268
9.10.2.2 AllocVRAMOutputBuffer()	268
9.10.2.3 Buff()	269
9.10.2.4 ConstructMatrix()	270
9.10.2.5 CudaCheckNotifyErr()	272
9.10.2.6 CudaMalloc2DToFlat()	272
9.10.2.7 CudaMemcpy2DToFlat()	273
9.10.2.8 CudaMigrate2DFlat()	274
9.10.2.9 DeallocateMatrix()	275
9.10.2.10 DumpJSON()	276
9.10.2.11 Edges()	277
9.10.2.12 Export() [1/2]	277
9.10.2.13 Export() [2/2]	278
9.10.2.14 FastRandomWalk() [1/3]	278
9.10.2.15 FastRandomWalk() [2/3]	280
9.10.2.16 FastRandomWalk() [3/3]	281
9.10.2.17 FastRandomWalkPartition()	282
9.10.2.18 FastRandomWalkThread()	283
9.10.2.19 FlattenMatrix()	284
9.10.2.20 GatherAsyncKernelOutput()	285
9.10.2.21 Generate()	285
9.10.2.22 GenerateThread()	286
9.10.2.23 Import() [1/2]	287
9.10.2.24 Import() [2/2]	288
9.10.2.25 LaunchAsyncKernel()	289

9.10.2.26 ListCudaDevices()	289
9.10.2.27 MigrateMatrix()	290
9.10.2.28 Nodes()	290
9.10.2.29 OpenDatasetFile()	290
9.10.2.30 OptimizeEdgeOrder()	291
9.10.2.31 prepKernelMemoryChannel()	291
9.10.2.32 RandomWalk()	292
9.10.2.33 Save()	293
9.10.2.34 StarterNode()	294
9.10.2.35 Train()	294
9.10.2.36 TrainThread()	295
9.10.3 Member Data Documentation	296
9.10.3.1 alternatingKernels	296
9.10.3.2 cudaBlocks	296
9.10.3.3 cudaGridSize	296
9.10.3.4 cudaMemPerGrid	297
9.10.3.5 cudaPerKernelAllocationSize	297
9.10.3.6 cudastreams	297
9.10.3.7 cudaThreads	297
9.10.3.8 datasetFile	297
9.10.3.9 device_edgeMatrix	297
9.10.3.10 device_matrixIndex	297
9.10.3.11 device_outputBuffer	297
9.10.3.12 device_seeds	297
9.10.3.13 device_totalEdgeWeights	298
9.10.3.14 device_valueMatrix	298
9.10.3.15 edgeMatrix	298
9.10.3.16 edges	298
9.10.3.17 flatEdgeMatrix	298
9.10.3.18 flatValueMatrix	298
9.10.3.19 iterationsPerKernelThread	298
9.10.3.20 matrixIndex	298
9.10.3.21 matrixSize	299
9.10.3.22 modelSavefile	299
9.10.3.23 nodes	299
9.10.3.24 numberOfPartitions	299
9.10.3.25 outputBuffer	299
9.10.3.26 outputFile	299
9.10.3.27 ready	299
9.10.3.28 starterNode	300
9.10.3.29 totalEdgeWeights	300
9.10.3.30 totalOutputPerKernel	300

9.10.3.31 totalOutputPerSync	300
9.10.3.32 valueMatrix	300
9.11 Python.CudaMarkopy.CudaModelMatrixCLI Class Reference	300
9.11.1 Detailed Description	308
9.11.2 Constructor & Destructor Documentation	308
9.11.2.1 __init__().	308
9.11.3 Member Function Documentation	308
9.11.3.1 _generate().	308
9.11.3.2 add_arguments().	309
9.11.3.3 AdjustEdge() [1/2]	309
9.11.3.4 AdjustEdge() [2/2]	310
9.11.3.5 AllocVRAMOutputBuffer().	311
9.11.3.6 Buff() [1/2]	311
9.11.3.7 Buff() [2/2]	312
9.11.3.8 check_corpus_path() [1/2]	314
9.11.3.9 check_corpus_path() [2/2]	314
9.11.3.10 check_export_path() [1/2]	315
9.11.3.11 check_export_path() [2/2]	315
9.11.3.12 check_import_path() [1/2]	316
9.11.3.13 check_import_path() [2/2]	316
9.11.3.14 ConstructMatrix() [1/2]	317
9.11.3.15 ConstructMatrix() [2/2]	318
9.11.3.16 CudaCheckNotifyErr().	320
9.11.3.17 CudaMalloc2DToFlat().	320
9.11.3.18 CudaMemcpy2DToFlat().	321
9.11.3.19 CudaMigrate2DFlat().	322
9.11.3.20 DeallocateMatrix() [1/2]	323
9.11.3.21 DeallocateMatrix() [2/2]	324
9.11.3.22 DumpJSON() [1/2]	325
9.11.3.23 DumpJSON() [2/2]	326
9.11.3.24 Edges() [1/2]	327
9.11.3.25 Edges() [2/2]	327
9.11.3.26 Export() [1/4]	327
9.11.3.27 Export() [2/4]	328
9.11.3.28 export() [1/2]	328
9.11.3.29 export() [2/2]	329
9.11.3.30 Export() [3/4]	329
9.11.3.31 Export() [4/4]	330
9.11.3.32 FastRandomWalk() [1/6]	330
9.11.3.33 FastRandomWalk() [2/6]	330
9.11.3.34 FastRandomWalk() [3/6]	332
9.11.3.35 FastRandomWalk() [4/6]	333

9.11.3.36 FastRandomWalk()	[5/6]	334
9.11.3.37 FastRandomWalk()	[6/6]	335
9.11.3.38 FastRandomWalkPartition()	[1/2]	336
9.11.3.39 FastRandomWalkPartition()	[2/2]	338
9.11.3.40 FastRandomWalkThread()	[1/2]	339
9.11.3.41 FastRandomWalkThread()	[2/2]	340
9.11.3.42 FlattenMatrix()		342
9.11.3.43 GatherAsyncKernelOutput()		342
9.11.3.44 generate()	[1/2]	342
9.11.3.45 generate()	[2/2]	343
9.11.3.46 Generate()	[1/2]	344
9.11.3.47 Generate()	[2/2]	345
9.11.3.48 GenerateThread()		346
9.11.3.49 help()	[1/2]	347
9.11.3.50 help()	[2/2]	348
9.11.3.51 Import()	[1/4]	348
9.11.3.52 Import()	[2/4]	349
9.11.3.53 Import()	[3/4]	350
9.11.3.54 Import()	[4/4]	351
9.11.3.55 import_model()	[1/2]	352
9.11.3.56 import_model()	[2/2]	353
9.11.3.57 init_post_arguments()		354
9.11.3.58 LaunchAsyncKernel()		354
9.11.3.59 ListCudaDevices()		354
9.11.3.60 MigrateMatrix()		355
9.11.3.61 Nodes()	[1/2]	355
9.11.3.62 Nodes()	[2/2]	356
9.11.3.63 OpenDatasetFile()	[1/2]	356
9.11.3.64 OpenDatasetFile()	[2/2]	356
9.11.3.65 OptimizeEdgeOrder()	[1/2]	357
9.11.3.66 OptimizeEdgeOrder()	[2/2]	357
9.11.3.67 parse()	[1/2]	358
9.11.3.68 parse()	[2/2]	358
9.11.3.69 parse_arguments()	[1/2]	359
9.11.3.70 parse_arguments()	[2/2]	359
9.11.3.71 prepKernelMemoryChannel()		360
9.11.3.72 process()	[1/2]	360
9.11.3.73 process()	[2/2]	362
9.11.3.74 RandomWalk()	[1/2]	363
9.11.3.75 RandomWalk()	[2/2]	364
9.11.3.76 Save()	[1/2]	366
9.11.3.77 Save()	[2/2]	367

9.11.3.78 StarterNode() [1/2]	367
9.11.3.79 StarterNode() [2/2]	368
9.11.3.80 Train() [1/2]	368
9.11.3.81 Train() [2/2]	369
9.11.3.82 train() [1/2]	370
9.11.3.83 train() [2/2]	371
9.11.3.84 TrainThread()	373
9.11.4 Member Data Documentation	374
9.11.4.1 alternatingKernels	374
9.11.4.2 args [1/2]	374
9.11.4.3 args [2/2]	374
9.11.4.4 bInfinite	374
9.11.4.5 cudaBlocks	374
9.11.4.6 cudaGridSize	375
9.11.4.7 cudaMemPerGrid	375
9.11.4.8 cudaPerKernelAllocationSize	375
9.11.4.9 cudastreams	375
9.11.4.10 cudaThreads	375
9.11.4.11 datasetFile	375
9.11.4.12 device_edgeMatrix	375
9.11.4.13 device_matrixIndex	375
9.11.4.14 device_outputBuffer	375
9.11.4.15 device_seeds	376
9.11.4.16 device_totalEdgeWeights	376
9.11.4.17 device_valueMatrix	376
9.11.4.18 edgeMatrix [1/2]	376
9.11.4.19 edgeMatrix [2/2]	376
9.11.4.20 edges	376
9.11.4.21 fileIO	376
9.11.4.22 flatEdgeMatrix	377
9.11.4.23 flatValueMatrix	377
9.11.4.24 iterationsPerKernelThread	377
9.11.4.25 matrixIndex [1/2]	377
9.11.4.26 matrixIndex [2/2]	377
9.11.4.27 matrixSize [1/2]	377
9.11.4.28 matrixSize [2/2]	377
9.11.4.29 model	378
9.11.4.30 modelSavefile	378
9.11.4.31 nodes	378
9.11.4.32 numberOfPartitions	378
9.11.4.33 outputBuffer	378
9.11.4.34 outputFile	378

9.11.4.35 parser [1/2]	378
9.11.4.36 parser [2/2]	379
9.11.4.37 print_help [1/2]	379
9.11.4.38 print_help [2/2]	379
9.11.4.39 ready [1/2]	379
9.11.4.40 ready [2/2]	379
9.11.4.41 starterNode	379
9.11.4.42 totalEdgeWeights [1/2]	379
9.11.4.43 totalEdgeWeights [2/2]	380
9.11.4.44 totalOutputPerKernel	380
9.11.4.45 totalOutputPerSync	380
9.11.4.46 valueMatrix [1/2]	380
9.11.4.47 valueMatrix [2/2]	380
9.12 Markov::Random::DefaultRandomEngine Class Reference	380
9.12.1 Detailed Description	382
9.12.2 Member Function Documentation	383
9.12.2.1 distribution()	383
9.12.2.2 generator()	383
9.12.2.3 random()	384
9.12.2.4 rd()	384
9.13 Markov::Edge< NodeStorageType > Class Template Reference	385
9.13.1 Detailed Description	386
9.13.2 Constructor & Destructor Documentation	386
9.13.2.1 Edge() [1/2]	386
9.13.2.2 Edge() [2/2]	386
9.13.3 Member Function Documentation	387
9.13.3.1 AdjustEdge()	387
9.13.3.2 EdgeWeight()	387
9.13.3.3 LeftNode()	388
9.13.3.4 RightNode()	388
9.13.3.5 SetLeftEdge()	388
9.13.3.6 SetRightEdge()	389
9.13.3.7 TraverseNode()	389
9.13.4 Member Data Documentation	389
9.13.4.1 _left	389
9.13.4.2 _right	390
9.13.4.3 _weight	390
9.14 Markov::GUI::Generate Class Reference	390
9.14.1 Detailed Description	391
9.14.2 Constructor & Destructor Documentation	391
9.14.2.1 Generate()	392
9.14.3 Member Function Documentation	392

9.14.3.1 generation	392
9.14.3.2 home	393
9.14.3.3 train	393
9.14.3.4 vis	394
9.14.4 Member Data Documentation	394
9.14.4.1 ui	394
9.15 Python.Markopy.MarkopyCLI Class Reference	394
9.15.1 Detailed Description	401
9.15.2 Constructor & Destructor Documentation	401
9.15.2.1 __init__().	401
9.15.3 Member Function Documentation	401
9.15.3.1 _generate().	401
9.15.3.2 add_arguments().	402
9.15.3.3 AdjustEdge() [1/2]	403
9.15.3.4 AdjustEdge() [2/2]	403
9.15.3.5 Buff() [1/2]	404
9.15.3.6 Buff() [2/2]	405
9.15.3.7 check_corpus_path() [1/4]	407
9.15.3.8 check_corpus_path() [2/4]	407
9.15.3.9 check_corpus_path() [3/4]	408
9.15.3.10 check_corpus_path() [4/4]	408
9.15.3.11 check_export_path() [1/4]	409
9.15.3.12 check_export_path() [2/4]	409
9.15.3.13 check_export_path() [3/4]	410
9.15.3.14 check_export_path() [4/4]	410
9.15.3.15 check_import_path() [1/4]	411
9.15.3.16 check_import_path() [2/4]	411
9.15.3.17 check_import_path() [3/4]	412
9.15.3.18 check_import_path() [4/4]	412
9.15.3.19 ConstructMatrix().	413
9.15.3.20 DeallocateMatrix().	414
9.15.3.21 DumpJSON().	415
9.15.3.22 Edges() [1/2]	416
9.15.3.23 Edges() [2/2]	416
9.15.3.24 Export().	417
9.15.3.25 Export() [2/5]	417
9.15.3.26 export() [1/4]	417
9.15.3.27 export() [2/4]	418
9.15.3.28 export() [3/4]	419
9.15.3.29 export() [4/4]	419
9.15.3.30 Export() [3/5]	420
9.15.3.31 Export() [4/5]	420

9.15.3.32 Export() [5/5]	421
9.15.3.33 FastRandomWalk() [1/3]	421
9.15.3.34 FastRandomWalk() [2/3]	421
9.15.3.35 FastRandomWalk() [3/3]	422
9.15.3.36 FastRandomWalkPartition()	423
9.15.3.37 FastRandomWalkThread()	425
9.15.3.38 Generate() [1/3]	426
9.15.3.39 generate() [1/4]	426
9.15.3.40 generate() [2/4]	427
9.15.3.41 generate() [3/4]	428
9.15.3.42 generate() [4/4]	429
9.15.3.43 Generate() [2/3]	430
9.15.3.44 Generate() [3/3]	431
9.15.3.45 GenerateThread()	432
9.15.3.46 help()	433
9.15.3.47 Import() [1/4]	434
9.15.3.48 Import() [2/4]	435
9.15.3.49 Import() [3/4]	436
9.15.3.50 Import() [4/4]	437
9.15.3.51 import_model() [1/4]	437
9.15.3.52 import_model() [2/4]	438
9.15.3.53 import_model() [3/4]	439
9.15.3.54 import_model() [4/4]	440
9.15.3.55 init_post_arguments()	441
9.15.3.56 Nodes() [1/2]	441
9.15.3.57 Nodes() [2/2]	442
9.15.3.58 OpenDatasetFile() [1/2]	442
9.15.3.59 OpenDatasetFile() [2/2]	442
9.15.3.60 OptimizeEdgeOrder() [1/2]	443
9.15.3.61 OptimizeEdgeOrder() [2/2]	443
9.15.3.62 parse()	444
9.15.3.63 parse_arguments() [1/4]	444
9.15.3.64 parse_arguments() [2/4]	445
9.15.3.65 parse_arguments() [3/4]	445
9.15.3.66 parse_arguments() [4/4]	446
9.15.3.67 parse_fail()	446
9.15.3.68 process()	446
9.15.3.69 RandomWalk() [1/2]	446
9.15.3.70 RandomWalk() [2/2]	447
9.15.3.71 Save() [1/2]	448
9.15.3.72 Save() [2/2]	449
9.15.3.73 StarterNode() [1/2]	450

9.15.3.74 StarterNode() [2/2]	450
9.15.3.75 stub()	450
9.15.3.76 Train() [1/2]	450
9.15.3.77 train() [1/4]	451
9.15.3.78 train() [2/4]	453
9.15.3.79 train() [3/4]	454
9.15.3.80 train() [4/4]	456
9.15.3.81 Train() [2/2]	457
9.15.3.82 TrainThread()	457
9.15.4 Member Data Documentation	458
9.15.4.1 args	458
9.15.4.2 cli	458
9.15.4.3 datasetFile	458
9.15.4.4 edgeMatrix	459
9.15.4.5 edges	459
9.15.4.6 fileIO	459
9.15.4.7 matrixIndex	459
9.15.4.8 matrixSize	459
9.15.4.9 model [1/3]	459
9.15.4.10 model [2/3]	459
9.15.4.11 model [3/3]	460
9.15.4.12 modelSavefile	460
9.15.4.13 nodes	460
9.15.4.14 outputFile	460
9.15.4.15 parser [1/4]	460
9.15.4.16 parser [2/4]	460
9.15.4.17 parser [3/4]	460
9.15.4.18 parser [4/4]	461
9.15.4.19 print_help [1/4]	461
9.15.4.20 print_help [2/4]	461
9.15.4.21 print_help [3/4]	461
9.15.4.22 print_help [4/4]	461
9.15.4.23 ready	461
9.15.4.24 starterNode	461
9.15.4.25 totalEdgeWeights	462
9.15.4.26 valueMatrix	462
9.16 Python.Markopy.MarkovModel Class Reference	462
9.16.1 Detailed Description	466
9.16.2 Member Function Documentation	466
9.16.2.1 AdjustEdge()	466
9.16.2.2 Buff()	466
9.16.2.3 Edges()	468

9.16.2.4 Export() [1/3]	468
9.16.2.5 Export() [2/3]	468
9.16.2.6 Export() [3/3]	469
9.16.2.7 Generate() [1/2]	469
9.16.2.8 Generate() [2/2]	469
9.16.2.9 GenerateThread()	470
9.16.2.10 Import() [1/3]	471
9.16.2.11 Import() [2/3]	472
9.16.2.12 Import() [3/3]	473
9.16.2.13 Nodes()	473
9.16.2.14 OpenDatasetFile()	473
9.16.2.15 OptimizeEdgeOrder()	474
9.16.2.16 RandomWalk()	474
9.16.2.17 Save()	475
9.16.2.18 StarterNode()	475
9.16.2.19 Train() [1/2]	476
9.16.2.20 Train() [2/2]	477
9.16.2.21 TrainThread()	477
9.16.3 Member Data Documentation	478
9.16.3.1 datasetFile	478
9.16.3.2 edges	478
9.16.3.3 modelSavefile	478
9.16.3.4 nodes	478
9.16.3.5 outputFile	479
9.16.3.6 starterNode	479
9.17 Markov::API::MarkovPasswords Class Reference	479
9.17.1 Detailed Description	483
9.17.2 Constructor & Destructor Documentation	483
9.17.2.1 MarkovPasswords() [1/2]	483
9.17.2.2 MarkovPasswords() [2/2]	483
9.17.3 Member Function Documentation	483
9.17.3.1 AdjustEdge()	484
9.17.3.2 Buff()	484
9.17.3.3 Edges()	485
9.17.3.4 Export() [1/2]	486
9.17.3.5 Export() [2/2]	486
9.17.3.6 Generate()	486
9.17.3.7 GenerateThread()	488
9.17.3.8 Import() [1/2]	489
9.17.3.9 Import() [2/2]	489
9.17.3.10 Nodes()	490
9.17.3.11 OpenDatasetFile()	490

9.17.3.12 OptimizeEdgeOrder()	490
9.17.3.13 RandomWalk()	491
9.17.3.14 Save()	492
9.17.3.15 StarterNode()	492
9.17.3.16 Train()	493
9.17.3.17 TrainThread()	494
9.17.4 Member Data Documentation	495
9.17.4.1 datasetFile	495
9.17.4.2 edges	495
9.17.4.3 modelSavefile	495
9.17.4.4 nodes	495
9.17.4.5 outputFile	495
9.17.4.6 starterNode	495
9.18 Python.Markopy.MarkovPasswordsCLI Class Reference	496
9.18.1 Detailed Description	500
9.18.2 Constructor & Destructor Documentation	500
9.18.2.1 __init__()	500
9.18.3 Member Function Documentation	500
9.18.3.1 _generate()	500
9.18.3.2 add_arguments()	501
9.18.3.3 AdjustEdge()	501
9.18.3.4 Buff()	502
9.18.3.5 check_corpus_path() [1/2]	503
9.18.3.6 check_corpus_path() [2/2]	504
9.18.3.7 check_export_path() [1/2]	504
9.18.3.8 check_export_path() [2/2]	505
9.18.3.9 check_import_path() [1/2]	505
9.18.3.10 check_import_path() [2/2]	506
9.18.3.11 Edges()	506
9.18.3.12 Export() [1/3]	507
9.18.3.13 export() [1/2]	507
9.18.3.14 export() [2/2]	508
9.18.3.15 Export() [2/3]	508
9.18.3.16 Export() [3/3]	509
9.18.3.17 Generate() [1/2]	509
9.18.3.18 generate() [1/2]	509
9.18.3.19 generate() [2/2]	510
9.18.3.20 Generate() [2/2]	511
9.18.3.21 GenerateThread()	512
9.18.3.22 help() [1/2]	513
9.18.3.23 help() [2/2]	514
9.18.3.24 Import() [1/3]	514

9.18.3.25 Import() [2/3]	514
9.18.3.26 Import() [3/3]	515
9.18.3.27 import_model() [1/2]	515
9.18.3.28 import_model() [2/2]	516
9.18.3.29 init_post_arguments() [1/2]	517
9.18.3.30 init_post_arguments() [2/2]	518
9.18.3.31 Nodes()	518
9.18.3.32 OpenDatasetFile()	518
9.18.3.33 OptimizeEdgeOrder()	519
9.18.3.34 parse() [1/2]	519
9.18.3.35 parse() [2/2]	520
9.18.3.36 parse_arguments() [1/2]	521
9.18.3.37 parse_arguments() [2/2]	521
9.18.3.38 process() [1/2]	522
9.18.3.39 process() [2/2]	523
9.18.3.40 RandomWalk()	524
9.18.3.41 Save()	525
9.18.3.42 StarterNode()	526
9.18.3.43 Train() [1/2]	526
9.18.3.44 train() [1/2]	527
9.18.3.45 train() [2/2]	529
9.18.3.46 Train() [2/2]	530
9.18.3.47 TrainThread()	530
9.18.4 Member Data Documentation	531
9.18.4.1 args [1/2]	531
9.18.4.2 args [2/2]	531
9.18.4.3 datasetFile	532
9.18.4.4 edges	532
9.18.4.5 model	532
9.18.4.6 modelSavefile	532
9.18.4.7 nodes	532
9.18.4.8 outputFile	532
9.18.4.9 parser [1/2]	532
9.18.4.10 parser [2/2]	532
9.18.4.11 print_help [1/2]	533
9.18.4.12 print_help [2/2]	533
9.18.4.13 starterNode	533
9.19 Markov::GUI::MarkovPasswordsGUI Class Reference	533
9.19.1 Detailed Description	535
9.19.2 Constructor & Destructor Documentation	535
9.19.2.1 MarkovPasswordsGUI()	535
9.19.3 Member Function Documentation	535

9.19.3.1 benchmarkSelected	535
9.19.3.2 home	535
9.19.3.3 model	536
9.19.3.4 pass	536
9.19.4 Member Data Documentation	536
9.19.4.1 ui	536
9.20 Markov::API::CUDA::Random::Marsaglia Class Reference	537
9.20.1 Detailed Description	539
9.20.2 Member Function Documentation	539
9.20.2.1 CudaCheckNotifyErr()	540
9.20.2.2 CudaMalloc2DToFlat()	540
9.20.2.3 CudaMemcpy2DToFlat()	541
9.20.2.4 CudaMigrate2DFlat()	542
9.20.2.5 distribution()	543
9.20.2.6 generator()	544
9.20.2.7 ListCudaDevices()	544
9.20.2.8 MigrateToVRAM()	545
9.20.2.9 random()	545
9.20.2.10 rd()	546
9.20.3 Member Data Documentation	546
9.20.3.1 x	546
9.20.3.2 y	546
9.20.3.3 z	546
9.21 Markov::Random::Marsaglia Class Reference	547
9.21.1 Detailed Description	549
9.21.2 Constructor & Destructor Documentation	549
9.21.2.1 Marsaglia()	549
9.21.3 Member Function Documentation	549
9.21.3.1 distribution()	549
9.21.3.2 generator()	550
9.21.3.3 random()	550
9.21.3.4 rd()	551
9.21.4 Member Data Documentation	551
9.21.4.1 x	551
9.21.4.2 y	551
9.21.4.3 z	552
9.22 Markov::GUI::menu Class Reference	552
9.22.1 Detailed Description	553
9.22.2 Constructor & Destructor Documentation	553
9.22.2.1 menu()	553
9.22.3 Member Function Documentation	554
9.22.3.1 about	554

9.22.3.2 visualization	554
9.22.4 Member Data Documentation	554
9.22.4.1 ui	554
9.23 Markov::Random::Mersenne Class Reference	555
9.23.1 Detailed Description	557
9.23.2 Member Function Documentation	557
9.23.2.1 distribution()	557
9.23.2.2 generator()	557
9.23.2.3 random()	558
9.23.2.4 rd()	558
9.24 Markov::Model< NodeStorageType > Class Template Reference	559
9.24.1 Detailed Description	561
9.24.2 Constructor & Destructor Documentation	561
9.24.2.1 Model()	561
9.24.3 Member Function Documentation	562
9.24.3.1 AdjustEdge()	562
9.24.3.2 Edges()	562
9.24.3.3 Export() [1/2]	563
9.24.3.4 Export() [2/2]	563
9.24.3.5 Import() [1/2]	564
9.24.3.6 Import() [2/2]	565
9.24.3.7 Nodes()	566
9.24.3.8 OptimizeEdgeOrder()	566
9.24.3.9 RandomWalk()	567
9.24.3.10 StarterNode()	568
9.24.4 Member Data Documentation	568
9.24.4.1 edges	568
9.24.4.2 nodes	569
9.24.4.3 starterNode	569
9.25 Markov::API::ModelMatrix Class Reference	569
9.25.1 Detailed Description	573
9.25.2 Constructor & Destructor Documentation	573
9.25.2.1 ModelMatrix()	574
9.25.3 Member Function Documentation	574
9.25.3.1 AdjustEdge()	574
9.25.3.2 Buff()	574
9.25.3.3 ConstructMatrix()	576
9.25.3.4 DeallocateMatrix()	577
9.25.3.5 DumpJSON()	578
9.25.3.6 Edges()	579
9.25.3.7 Export() [1/2]	579
9.25.3.8 Export() [2/2]	579

9.25.3.9 FastRandomWalk() [1/2]	580
9.25.3.10 FastRandomWalk() [2/2]	581
9.25.3.11 FastRandomWalkPartition()	582
9.25.3.12 FastRandomWalkThread()	583
9.25.3.13 Generate()	585
9.25.3.14 GenerateThread()	586
9.25.3.15 Import() [1/2]	587
9.25.3.16 Import() [2/2]	588
9.25.3.17 Nodes()	589
9.25.3.18 OpenDatasetFile()	589
9.25.3.19 OptimizeEdgeOrder()	590
9.25.3.20 RandomWalk()	590
9.25.3.21 Save()	591
9.25.3.22 StarterNode()	591
9.25.3.23 Train()	592
9.25.3.24 TrainThread()	593
9.25.4 Member Data Documentation	594
9.25.4.1 datasetFile	594
9.25.4.2 edgeMatrix	594
9.25.4.3 edges	594
9.25.4.4 matrixIndex	594
9.25.4.5 matrixSize	594
9.25.4.6 modelSavefile	594
9.25.4.7 nodes	595
9.25.4.8 outputFile	595
9.25.4.9 ready	595
9.25.4.10 starterNode	595
9.25.4.11 totalEdgeWeights	595
9.25.4.12 valueMatrix	595
9.26 Python.Markopy.ModelMatrix Class Reference	595
9.26.1 Detailed Description	599
9.26.2 Member Function Documentation	599
9.26.2.1 AdjustEdge()	600
9.26.2.2 Buff()	600
9.26.2.3 ConstructMatrix()	601
9.26.2.4 DeallocateMatrix()	603
9.26.2.5 DumpJSON()	604
9.26.2.6 Edges()	605
9.26.2.7 Export() [1/2]	605
9.26.2.8 Export() [2/2]	605
9.26.2.9 FastRandomWalk() [1/3]	606
9.26.2.10 FastRandomWalk() [2/3]	606

9.26.2.11 FastRandomWalk()	[3/3]	607
9.26.2.12 FastRandomWalkPartition()		608
9.26.2.13 FastRandomWalkThread()		609
9.26.2.14 Generate()		611
9.26.2.15 GenerateThread()		612
9.26.2.16 Import()	[1/2]	613
9.26.2.17 Import()	[2/2]	614
9.26.2.18 Nodes()		615
9.26.2.19 OpenDatasetFile()		615
9.26.2.20 OptimizeEdgeOrder()		616
9.26.2.21 RandomWalk()		616
9.26.2.22 Save()		617
9.26.2.23 StarterNode()		618
9.26.2.24 Train()		618
9.26.2.25 TrainThread()		619
9.26.3 Member Data Documentation		620
9.26.3.1 datasetFile		620
9.26.3.2 edgeMatrix		620
9.26.3.3 edges		620
9.26.3.4 matrixIndex		621
9.26.3.5 matrixSize		621
9.26.3.6 modelSavefile		621
9.26.3.7 nodes		621
9.26.3.8 outputFile		621
9.26.3.9 ready		621
9.26.3.10 starterNode		621
9.26.3.11 totalEdgeWeights		622
9.26.3.12 valueMatrix		622
9.27 Python.Markopy.ModelMatrixCLI Class Reference		622
9.27.1 Detailed Description		627
9.27.2 Constructor & Destructor Documentation		627
9.27.2.1 __init__()		627
9.27.3 Member Function Documentation		627
9.27.3.1 _generate()		627
9.27.3.2 add_arguments()		628
9.27.3.3 AdjustEdge()		629
9.27.3.4 Buff()		629
9.27.3.5 check_corpus_path()		630
9.27.3.6 check_export_path()		631
9.27.3.7 check_import_path()		631
9.27.3.8 ConstructMatrix()		632
9.27.3.9 DeallocateMatrix()		633

9.27.3.10 DumpJSON()	634
9.27.3.11 Edges()	635
9.27.3.12 Export() [1/2]	635
9.27.3.13 export()	636
9.27.3.14 Export() [2/2]	636
9.27.3.15 FastRandomWalk() [1/3]	637
9.27.3.16 FastRandomWalk() [2/3]	637
9.27.3.17 FastRandomWalk() [3/3]	638
9.27.3.18 FastRandomWalkPartition()	639
9.27.3.19 FastRandomWalkThread()	640
9.27.3.20 generate()	642
9.27.3.21 Generate()	643
9.27.3.22 GenerateThread()	644
9.27.3.23 help()	645
9.27.3.24 Import() [1/2]	645
9.27.3.25 Import() [2/2]	646
9.27.3.26 import_model()	647
9.27.3.27 init_post_arguments()	648
9.27.3.28 Nodes()	649
9.27.3.29 OpenDatasetFile()	649
9.27.3.30 OptimizeEdgeOrder()	649
9.27.3.31 parse()	650
9.27.3.32 parse_arguments()	650
9.27.3.33 process()	651
9.27.3.34 RandomWalk()	652
9.27.3.35 Save()	653
9.27.3.36 StarterNode()	654
9.27.3.37 Train()	654
9.27.3.38 train()	655
9.27.3.39 TrainThread()	656
9.27.4 Member Data Documentation	657
9.27.4.1 args	657
9.27.4.2 datasetFile	657
9.27.4.3 edgeMatrix	658
9.27.4.4 edges	658
9.27.4.5 fileIO	658
9.27.4.6 matrixIndex	658
9.27.4.7 matrixSize	658
9.27.4.8 model	658
9.27.4.9 modelSavefile	658
9.27.4.10 nodes	659
9.27.4.11 outputFile	659

9.27.4.12 parser	659
9.27.4.13 print_help	659
9.27.4.14 ready	659
9.27.4.15 starterNode	659
9.27.4.16 totalEdgeWeights	659
9.27.4.17 valueMatrix	660
9.28 Markov::Node< storageType > Class Template Reference	660
9.28.1 Detailed Description	661
9.28.2 Constructor & Destructor Documentation	661
9.28.2.1 Node() [1/2]	662
9.28.2.2 Node() [2/2]	662
9.28.3 Member Function Documentation	662
9.28.3.1 Edges()	662
9.28.3.2 FindEdge() [1/2]	662
9.28.3.3 FindEdge() [2/2]	663
9.28.3.4 Link() [1/2]	663
9.28.3.5 Link() [2/2]	664
9.28.3.6 NodeValue()	664
9.28.3.7 RandomNext()	665
9.28.3.8 TotalEdgeWeights()	665
9.28.3.9 UpdateEdges()	666
9.28.3.10 UpdateTotalVerticeWeight()	666
9.28.4 Member Data Documentation	667
9.28.4.1 _value	667
9.28.4.2 edges	667
9.28.4.3 edgesV	667
9.28.4.4 total_edge_weights	667
9.29 QMainWindow Class Reference	667
9.30 Markov::Random::RandomEngine Class Reference	668
9.30.1 Detailed Description	670
9.30.2 Member Function Documentation	670
9.30.2.1 random()	670
9.31 Markov::API::CLI::Terminal Class Reference	670
9.31.1 Detailed Description	672
9.31.2 Member Enumeration Documentation	672
9.31.2.1 color	672
9.31.3 Constructor & Destructor Documentation	672
9.31.3.1 Terminal()	672
9.31.4 Member Data Documentation	672
9.31.4.1 colormap	673
9.31.4.2 endl	673
9.32 Markov::API::Concurrency::ThreadSharedListHandler Class Reference	673

9.32.1 Detailed Description	675
9.32.2 Constructor & Destructor Documentation	675
9.32.2.1 ThreadSharedListHandler()	675
9.32.3 Member Function Documentation	676
9.32.3.1 next()	676
9.32.4 Member Data Documentation	676
9.32.4.1 listfile	676
9.32.4.2 mlock	676
9.33 Markov::GUI::Train Class Reference	677
9.33.1 Detailed Description	678
9.33.2 Constructor & Destructor Documentation	678
9.33.2.1 Train()	678
9.33.3 Member Function Documentation	679
9.33.3.1 home	679
9.33.3.2 train	679
9.33.4 Member Data Documentation	680
9.33.4.1 ui	680
10 File Documentation	681
10.1 Markopy/CudaMarkopy/src/CLI/cudamarkopy.py File Reference	681
10.1.1 Detailed Description	681
10.2 cudamarkopy.py	681
10.3 Markopy/CudaMarkopy/src/CLI/cudammx.py File Reference	682
10.3.1 Detailed Description	683
10.4 cudammx.py	683
10.5 Markopy/CudaMarkopy/src/Module/cudaMarkopy.cu File Reference	684
10.5.1 Macro Definition Documentation	685
10.5.1.1 BOOST_PYTHON_STATIC_LIB	685
10.6 cudaMarkopy.cu	685
10.7 Markopy/CudaMarkovAPI/src/cudaDeviceController.cu File Reference	685
10.7.1 Detailed Description	686
10.8 cudaDeviceController.cu	686
10.9 Markopy/CudaMarkovAPI/src/cudaDeviceController.h File Reference	687
10.9.1 Detailed Description	688
10.10 cudaDeviceController.h	688
10.11 Markopy/CudaMarkovAPI/src/cudaModelMatrix.cu File Reference	690
10.11.1 Detailed Description	691
10.12 cudaModelMatrix.cu	691
10.13 Markopy/CudaMarkovAPI/src/cudaModelMatrix.h File Reference	694
10.13.1 Detailed Description	696
10.14 cudaModelMatrix.h	696
10.15 Markopy/CudaMarkovAPI/src/cudarandom.h File Reference	698

10.15.1 Detailed Description	699
10.16 cudarandom.h	700
10.17 Markopy/CudaMarkovAPI/src/main.cu File Reference	701
10.17.1 Detailed Description	701
10.17.2 Function Documentation	701
10.17.2.1 main()	701
10.18 main.cu	702
10.19 Markopy/documentation/dirs.docs File Reference	702
10.19.1 Detailed Description	702
10.20 dirs.docs	702
10.21 Markopy/Markopy/src/CLI/base.py File Reference	703
10.21.1 Detailed Description	704
10.22 base.py	704
10.23 Markopy/Markopy/src/CLI/importer.py File Reference	707
10.23.1 Detailed Description	708
10.24 importer.py	708
10.25 Markopy/Markopy/src/CLI/markopy.py File Reference	708
10.25.1 Detailed Description	708
10.26 markopy.py	709
10.27 Markopy/Markopy/src/CLI/mm.py File Reference	710
10.27.1 Detailed Description	711
10.28 mm.py	711
10.29 Markopy/Markopy/src/CLI/mmx.py File Reference	711
10.29.1 Detailed Description	712
10.30 mmx.py	712
10.31 Markopy/Markopy/src/CLI/mp.py File Reference	712
10.31.1 Detailed Description	713
10.32 mp.py	713
10.33 Markopy/Markopy/src/Module/markopy.cpp File Reference	713
10.33.1 Detailed Description	714
10.33.2 Macro Definition Documentation	715
10.33.2.1 BOOST_ALL_STATIC_LIB	715
10.33.2.2 BOOST_PYTHON_STATIC_LIB	715
10.34 markopy.cpp	715
10.35 Markopy/MarkovAPI/src/markovPasswords.cpp File Reference	716
10.35.1 Detailed Description	716
10.35.2 Function Documentation	717
10.35.2.1 intHandler()	717
10.35.3 Variable Documentation	717
10.35.3.1 keepRunning	717
10.36 markovPasswords.cpp	717
10.37 Markopy/MarkovAPI/src/markovPasswords.h File Reference	720

10.37.1 Detailed Description	720
10.38 markovPasswords.h	721
10.39 Markopy/MarkovAPI/src/modelMatrix.cpp File Reference	722
10.39.1 Detailed Description	723
10.40 modelMatrix.cpp	723
10.41 Markopy/MarkovAPI/src/modelMatrix.h File Reference	726
10.41.1 Detailed Description	727
10.42 modelMatrix.h	727
10.43 Markopy/MarkovAPI/src/threadSharedListHandler.cpp File Reference	730
10.43.1 Detailed Description	730
10.44 threadSharedListHandler.cpp	731
10.45 Markopy/MarkovAPI/src/threadSharedListHandler.h File Reference	731
10.45.1 Detailed Description	732
10.46 threadSharedListHandler.h	732
10.47 Markopy/MarkovAPICLI/src/argparse.cpp File Reference	734
10.47.1 Detailed Description	734
10.48 argparse.cpp	734
10.49 Markopy/MarkovAPICLI/src/argparse.h File Reference	735
10.49.1 Detailed Description	736
10.49.2 Macro Definition Documentation	736
10.49.2.1 BOOST_ALL_STATIC_LIB	736
10.49.2.2 BOOST_PROGRAM_OPTIONS_STATIC_LIB	736
10.50 argparse.h	736
10.51 Markopy/MarkovAPICLI/src/color/term.cpp File Reference	739
10.51.1 Detailed Description	740
10.51.2 Function Documentation	740
10.51.2.1 operator<<()	740
10.52 term.cpp	740
10.53 Markopy/MarkovAPICLI/src/color/term.h File Reference	741
10.53.1 Detailed Description	743
10.53.2 Macro Definition Documentation	743
10.53.2.1 TERM_FAIL	743
10.53.2.2 TERM_INFO	743
10.53.2.3 TERM_SUCC	743
10.53.2.4 TERM_WARN	743
10.54 term.h	743
10.55 Markopy/MarkovAPICLI/src/main.cpp File Reference	744
10.55.1 Detailed Description	745
10.55.2 Function Documentation	745
10.55.2.1 main()	745
10.56 main.cpp	746
10.57 Markopy/MarkovPasswordsGUI/src/main.cpp File Reference	747

10.57.1 Detailed Description	747
10.57.2 Function Documentation	747
10.57.2.1 main()	747
10.58 main.cpp	748
10.59 Markopy/MarkovAPICLI/src/scripts/model_2gram.py File Reference	748
10.60 model_2gram.py	748
10.61 Markopy/MarkovAPICLI/src/scripts/random_model.py File Reference	749
10.62 random_model.py	749
10.63 Markopy/MarkovModel/src/dllmain.cpp File Reference	749
10.63.1 Detailed Description	750
10.64 dllmain.cpp	750
10.65 Markopy/MarkovModel/src/edge.h File Reference	751
10.65.1 Detailed Description	751
10.66 edge.h	752
10.67 Markopy/MarkovModel/src/framework.h File Reference	754
10.67.1 Detailed Description	754
10.67.2 Macro Definition Documentation	754
10.67.2.1 WIN32_LEAN_AND_MEAN	754
10.68 framework.h	755
10.69 Markopy/MarkovModel/src/model.h File Reference	755
10.69.1 Detailed Description	756
10.70 model.h	756
10.71 Markopy/MarkovModel/src/node.h File Reference	760
10.71.1 Detailed Description	761
10.72 node.h	761
10.73 Markopy/MarkovModel/src/pch.cpp File Reference	765
10.73.1 Detailed Description	765
10.74 pch.cpp	765
10.75 Markopy/UnitTests/pch.cpp File Reference	766
10.75.1 Detailed Description	766
10.76 pch.cpp	766
10.77 Markopy/MarkovModel/src/pch.h File Reference	766
10.77.1 Detailed Description	767
10.78 pch.h	767
10.79 Markopy/UnitTests/pch.h File Reference	768
10.79.1 Detailed Description	768
10.80 pch.h	768
10.81 Markopy/MarkovModel/src/random.h File Reference	768
10.81.1 Detailed Description	770
10.82 random.h	770
10.83 Markopy/MarkovPasswordsGUI/src/about.cpp File Reference	772
10.83.1 Detailed Description	773

10.84 about.cpp	773
10.85 Markopy/MarkovPasswordsGUI/src/about.h File Reference	773
10.85.1 Detailed Description	774
10.86 about.h	775
10.87 Markopy/MarkovPasswordsGUI/src/CLI.cpp File Reference	775
10.87.1 Detailed Description	775
10.88 CLI.cpp	776
10.89 Markopy/MarkovPasswordsGUI/src/CLI.h File Reference	776
10.89.1 Detailed Description	777
10.90 CLI.h	777
10.91 Markopy/MarkovPasswordsGUI/src/Generate.cpp File Reference	777
10.91.1 Detailed Description	778
10.92 Generate.cpp	778
10.93 Markopy/MarkovPasswordsGUI/src/Generate.h File Reference	780
10.93.1 Detailed Description	780
10.94 Generate.h	781
10.95 Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.cpp File Reference	781
10.95.1 Detailed Description	781
10.96 MarkovPasswordsGUI.cpp	782
10.97 Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.h File Reference	782
10.97.1 Detailed Description	783
10.98 MarkovPasswordsGUI.h	783
10.99 Markopy/MarkovPasswordsGUI/src/menu.cpp File Reference	784
10.99.1 Detailed Description	784
10.100 menu.cpp	784
10.101 Markopy/MarkovPasswordsGUI/src/menu.h File Reference	785
10.101.1 Detailed Description	786
10.102 menu.h	786
10.103 Markopy/MarkovPasswordsGUI/src/Train.cpp File Reference	786
10.103.1 Detailed Description	787
10.104 Train.cpp	787
10.105 Markopy/MarkovPasswordsGUI/src/Train.h File Reference	788
10.105.1 Detailed Description	789
10.106 Train.h	789
10.107 Markopy/NVSight/report.md File Reference	790
10.108 Markopy/README.md File Reference	790
10.109 Markopy/UnitTests/UnitTests.cpp File Reference	790
10.109.1 Detailed Description	791
10.110 UnitTests.cpp	791
Index	799

CHAPTER1

Markopy

Markopy

Generate wordlists with markov models.

[HTML documentation](#) · [PDF documentation](#) · [Github Page](#) · [Report Bug](#) · [Add a Bug](#)

Table of Contents

1. [About The Project](#)
 - [Possible Use Cases](#)
 - [Getting Started](#)
 - [Releases](#)
 2. [Building](#)
 - [CMake configuration](#)
 - [Build everything](#)
 - [Build libraries only](#)
 - [Build CUDA-accelerated libraries](#)
 - [Build python module & libraries](#)
 - [Build CUDA accelerated python module](#)
 - [Build CUDA accelerated python module and the GUI](#)
 - [Prerequisites](#)
 - [General Prerequisites](#)
 - * [Linux](#)
 - * [Windows](#)
 - [MarkovModel](#)
 - [MarkovAPI](#)
 - [MarkovAPICLI](#)
 - [Markopy](#)
 - [CudaMarkovAPI](#)
 - [CudaMarkopy](#)
 - [MarkovPasswordsGUI](#)
 - [Installing Dependencies](#)
 - [Windows](#)
 - [Linux](#)
 3. [Known Common Issues](#)
 4. [Contributing](#)
 5. [Contact](#)
-

1.1 About The Project

This projects primary goal is to create a comfortable development environment for working with [Markov](#) Models, as well as creating an end product which can be used for generating password wordlists using [Markov](#) Models. This project contains following sub-projects:

- MarkovModel
 - A versatile header-only template library for basic [Markov](#) Model structure.
- MarkovAPI
 - A static/dynamic library built on MarkovModel, specialized to generate single-word lines.
- MarkovAPICLI
 - A command line interface built on top of MarkovAPI
- Markopy
 - A CPython extension wrapper for MarkovAPI, along with its own command line interface.
- MarkovPasswordsGUI
 - A graphical user interface for MarkovAPI
- CudaMarkovAPI
 - GPU-accelerated wrapper for MarkovAPI
- CudaMarkopy
 - GPU-accelereted wrapper for CudaMarkovAPI

1.1.1 Possible Use Cases

While main focus of the development has been towards random walk performance and password generation, underlying libraries could be used for other applications such as specific use cases of hidden markov models in bioinformatics and gene research.

1.1.2 Getting Started

If you'd just like to use the project without contributing, check out the releases page. Latest minor release (0.8.x, 0.9.x) is even with main branch, and latest patch release (0.8.1, 0.8.2) is even with development branch.

1.1.3 Releases

Releases are maintained automatically via github actions. Each push to the main branch will trigger a minor version release, while each accepted pull request into the development branch will trigger a patch version release.

Pull requests to the development branch will also trigger a draft release only visible to the maintainers.

Release files contain:

- libmarkov-{version}-{platform}.zip
 - Depending on the platform, contains the libmarkov.so or markov.lib from that version.
- libcudamarkov-{version}-{platform}.zip
 - Depending on the platform, contains the libcudamarkov.so or cudamarkov.lib from that version.
- markopy-{version}-{platform}-py{ver}.{extension}.zip
 - Depending on the paltform, contains markopy.so or markopy.pyd. CPython extensions are compiled for specific versions. If your python version is not supported by the releases, you can create an issue, or build it yourself using the python3.x-dev package.

- cudamarkopy-{version}-{platform}-py{ver}.{extension}.zip
 - Depending on the platform, contains cudamarkopy.so or cudamarkopy.pyd. CPython extensions are compiled for specific versions. If your python version is not supported by the releases, you can create an issue, or build it yourself using the python3.x-dev package.
 - models-{version}.zip
 - Contains the latest models with the release version. Contains base models (untrained), trained models, and language-specific models.
-

1.2 Building

You can build the project using cmake with g++ & nvcc on linux, and msbuild & nvcc on windows.

1.3 Prerequisites

You can find a list of the dependencies below. If you have any missing, check out the [setting up prerequisites](#) part.

1.3.1 General prerequisites

To build the simple core of this project, you'll need:

1.3.1.1 Linux

- CMake, preferably one of the latest versions.
- CXX compiler, preferably g++ or clang++ (LLVM 3.9+).

1.3.1.2 Windows

- CMake, preferably one of the latest versions.
- CXX compiler, preferably msbuild(cl.exe) or clang++ (LLVM 3.9+). Please note that mingw is not recommended as it is not officially supported by the nvcc.exe, and might not be linkable if you are building the CUDA components too.

1.3.2 MarkovModel

This project does not have any extra dependencies, and it can be compiled with general dependencies without anything extra.

1.3.3 MarkovAPI

This project does not have any extra dependencies, and it can be compiled with general dependencies without anything extra.

1.3.4 MarkovAPICLI

- Boost.program_options (tested on 1.71.0-1.76.0)

1.3.5 Markopy

- Boost.Python (tested on 1.71.0-1.76.0)
 - Python development package (tested on python 36-39)
-

1.3.6 CudaMarkovAPI

- CUDA toolkit (11.0+, c++17 support required)

1.3.7 CudaMarkopy

- CUDA toolkit (11.0+, c++17 support required)
- Boost.Python (tested on 1.71.0-1.76.0)
- Python development package (tested on python 36-39)

1.3.8 MarkovPasswordsGUI

- QT5 development environment. (qt5-qmake on ubuntu apt-get)
- QTWebEngine5 plugin. (qtwebengine5-dev on ubuntu apt-get)

1.3.9 CMake Configuration

You can build this project with cmake.

If you don't have prerequisites for some of the projects set up, you can use the partial set up configuration to ignore those projects when setting the project up.

If you do not meet the prerequisites, you'll have to partially set up the CMake file (you can't use -target to build some of the targets, because configuration phase will fail too).

Some examples for partially setting up and building the project are below.

1.3.9.1 Build everything

```
$ cmake . -DPYTHON_VER=38 && cmake --build .
```

This will build all the libraries and executables. Requires python-dev, CUDA, QT5, QT5-Webview

1.3.9.2 Build libraries only

```
$ cmake . -DPARTIAL=1 -DB_LIBS=1 && cmake --build .
```

Only build basic libraries. Requires only CXX compiler.

1.3.9.3 Build CUDA-accelerated libraries

```
$ cmake . -DPARTIAL=1 -DB_CUDA=1 && cmake --build .
```

Build libraries along with cuda accelerated ones.

1.3.9.4 Build python module & libraries

```
$ cmake . -DPARTIAL=1 -DPYTHON_VER=39 && cmake --build .
```

Will build basic libraries and python modules.

1.3.9.5 Build CUDA accelerated python module

```
$ cmake . -DPARTIAL=1 -DPYTHON_VER=39 -DB_CUDA=1 && cmake --build .
```

Will build cudamarkopy.

1.3.9.6 Build CUDA accelerated python module and the GUI

```
$ cmake . -DPARTIAL=1 -DPYTHON_VER=39 -DB_CUDA=1 -DB_GUI && cmake --build .
```

Combine methods

1.3.10 Installing Dependencies

1.3.10.0.1 Windows

- QT: Install [QT For Windows](#)
- Boost (program_options and python):
 - Download Boost from [its website](#). Prefer one of the tested versions, 1.71.0 to 1.76.0
 - Unzip the contents.
 - Launch "Visual Studio Developer Command Prompt" (If you don't have this, properly set up the PATH% variable for cl.exe)
 - Move to the boost installation directory. Bootstrap libraries with your python version:
`.\bootstrap.bat --with-python=$(which python3.6) --with-python-version=3.6;`
 - Run b2 to build the libraries.
`.\b2.exe --layout=system address-model=64 variant=release link=static runtime-link=shared threading=multi --with-program_options --with-python stage;`
- Python: You can use the windows app store to download python runtime and libraries.

1.3.10.0.2 Linux

- QT: Follow [this guide](#) to install QT on Linux. Alternatively, on ubuntu you can `sudo apt-get install qt5-qmake qtwebengine5-dev`
- Boost (program options and python):
 - Download Boost from [its website](#). Prefer one of the tested versions, 1.71.0 to 1.76.0
 - Unzip the contents.
 - Move to the boost installation directory. Bootstrap libraries with your python version:
`./bootstrap.sh --with-python=$(which python3.6) --with-python-version=3.6;`
 - Run b2 to build the libraries.
`./b2 variant=release link=static threading=multi --with-program_options install;`
`./b2 --with-python --buildid=3.6 install;`
- Boost (alternative)
 - Use a package manager to install boost
`sudo apt-get install libboost-all-dev`
- Python:
`sudo apt-get install python3-dev`

1.4 Known Common issues

1.4.1 Linux

1.4.1.1 Markopy - Python.h - Not found

Make sure you have the development version of python package, which includes the required header files. Check if header files exist: `/usr/include/python*` or `locate Python.h`. If it doesn't, run `sudo apt-get install python3-dev`

1.4.1.2 Markopy/MarkovAPI - *.so not found, or other library related issues when building

Run:

```
ls /usr/lib/x86_64-linux-gnu/ | grep boost
```

and check the shared object filenames. A common issue is that lboost is required but filenames are formatted as libboost, or vice versa.

Do the same for python related library issues, run:

```
ls /usr/lib/x86_64-linux-gnu/ | grep python
```

to verify filename format is as required.

If not, you can modify the makefile, or create symlinks such as:

```
ln -s /usr/lib/x86_64-linux-gnu/libboost_python38.so /usr/lib/x86_64-linux-gnu/boost_python38.so
```

1.4.2 Windows

1.4.2.1 Boost - Bootstrap.bat "ctype.h" not found

- Make sure you are working in the "Visual Studio Developer Command Prompt" terminal.
- Make sure you have Windows 10 SDK installed.
- From VS developer terminal, run echo INCLUDE%. If result does not have the windows sdk folders, run the following before running bootstrap (change your sdk version instead of 10.0.19041.0):

```
set INCLUDE=%INCLUDE%;C:\Program Files (x86)\Windows Kits\NETFXSDK\4.8\include\um;C:\Program Files (x86)\Windows Kits\10\include\10.0.19041.0\ucrt;C:\Program Files (x86)\Windows Kits\10\include\10.0.19041.0\shared;C:\Program Files (x86)\Windows Kits\10\include\10.0.19041.0\um;C:\Program Files (x86)\Windows Kits\10\include\10.0.19041.0\winrt;C:\Program Files (x86)\Windows Kits\10\include\10.0.19041.0\cppwinrt  
set LIB=%LIB%;C:\Program Files (x86)\Windows Kits\10\lib\10.0.19041.0\ucrt\x64;C:\Program Files (x86)\Windows Kits\10\lib\10.0.19041.0\um\x64
```

1.4.2.2 Cannot open file "*.lib"

Make sure you have set the BOOST_ROOT environment variable correctly. Make sure you ran b2 to build library files from boost sources.

1.4.2.3 Python.h not found

Make sure you have python installed, and make sure you set PYTHON_PATH environment variable.

1.5 Contributing

Feel free to contribute. We welcome all the issues and pull requests.

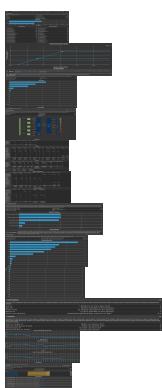
1.6 Contact

Twitter - [@ahakcil](#)

CHAPTER2

NVSight Report

2.0.1 Overview



2.0.2 Session



2.0.3 PTX



CHAPTER3

Deprecated List

Member **Markov::API::MarkovPasswords::Generate** (`unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20`)

See `Markov::API::MatrixModel::FastRandomWalk` for more information.

CHAPTER4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<code>base</code>	25
<code>cudamarkopy</code>	25
<code>cudammx</code>	25
<code>importer</code>	26
<code>markopy</code>	27
<code>Markov</code>	27
Namespace for the markov-model related classes. Contains <code>Model</code> , <code>Node</code> and <code>Edge</code> classes	27
<code>Markov::API</code>	28
Namespace for the <code>MarkovPasswords API</code>	28
<code>Markov::API::CLI</code>	28
Structure to hold parsed cli arguements	28
<code>Markov::API::Concurrency</code>	29
Namespace for <code>Concurrency</code> related classes	29
<code>Markov::API::CUDA</code>	29
Namespace for objects requiring <code>CUDA</code> libraries	29
<code>Markov::API::CUDA::Random</code>	32
Namespace for <code>Random</code> engines operable under <code>device</code> space	32
<code>Markov::GUI</code>	32
Namespace for <code>MarkovPasswords API</code> GUI wrapper	32
<code>Markov::Markopy</code>	33
CPython module for <code>Markov::API</code> objects	33
<code>Markov::Markopy::CUDA</code>	34
CPython module for <code>Markov::API::CUDA</code> objects	34
<code>Markov::Random</code>	35
Objects related to RNG	35
<code>mm</code>	35
<code>mmx</code>	35
<code>model_2gram</code>	36
<code>mp</code>	36
<code>Python.CudaMarkopy</code>	37
Wrapper scripts for <code>CudaMarkopy</code>	37
<code>Python.Markopy</code>	37
Wrapper scripts for <code>Markopy</code>	37
<code>random_model</code>	38
<code>Testing</code>	38
Namespace for Microsoft Native Unit <code>Testing</code> Classes	38
<code>Testing::MarkovModel</code>	38
Testing namespace for <code>MarkovModel</code>	38
<code>Testing::MarkovPasswords</code>	42
Testing namespace for <code>MarkovPasswords</code>	42
<code>Testing::MVP</code>	42
Testing Namespace for Minimal Viable Product	42
<code>Testing::MVP::MarkovModel</code>	42
Testing Namespace for <code>MVP MarkovModel</code>	42
<code>Testing::MVP::MarkovPasswords</code>	48
Testing namespace for <code>MVP MarkovPasswords</code>	48

CHAPTER5

Hierarchical Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Markov::API::CLI::_programOptions	51
Markov::API::CLI::Argparse	93
Python.Markopy.BaseCLI	99
Python.Markopy.AbstractGenerationModelCLI	55
Python.CudaMarkopy.CudaModelMatrixCLI	300
Python.CudaMarkopy.CudaMarkopyCLI	124
Python.Markopy.AbstractTrainingModelCLI	69
Python.Markopy.MarkovPasswordsCLI	496
Python.Markopy.MarkopyCLI	394
Python.CudaMarkopy.CudaMarkopyCLI	124
Python.Markopy.ModelMatrixCLI	622
Python.CudaMarkopy.CudaModelMatrixCLI	300
Python.Markopy.MarkopyCLI	394
Python.Markopy.AbstractTrainingModelCLI	69
Python.Markopy.MarkopyCLI	394
std::basic_string< char >::const_iterator	??
std::basic_string< char16_t >::const_iterator	??
std::basic_string< char32_t >::const_iterator	??
std::basic_string< char8_t >::const_iterator	??
std::basic_string< wchar_t >::const_iterator	??
std::basic_string_view< char >::const_iterator	??
std::basic_string_view< char16_t >::const_iterator	??
std::basic_string_view< char32_t >::const_iterator	??
std::basic_string_view< char8_t >::const_iterator	??
std::basic_string_view< wchar_t >::const_iterator	??
std::map< char, Markov::Edge< char > * >::const_iterator	??
std::map< char, Markov::Node< char > * >::const_iterator	??
std::map< Markov::API::CLI::Terminal::color, int >::const_iterator	??
std::map< NodeStorageType, Markov::Edge< NodeStorageType > * >::const_iterator	??
std::map< NodeStorageType, Markov::Node< NodeStorageType > * >::const_iterator	??
std::map< storageType, Markov::Edge< storageType > * >::const_iterator	??
std::vector< Markov::Edge< char > * >::const_iterator	??
std::vector< Markov::Edge< NodeStorageType > * >::const_iterator	??
std::vector< Markov::Edge< storageType > * >::const_iterator	??
std::basic_string< char >::const_reverse_iterator	??
std::basic_string< char16_t >::const_reverse_iterator	??
std::basic_string< char32_t >::const_reverse_iterator	??
std::basic_string< char8_t >::const_reverse_iterator	??
std::basic_string< wchar_t >::const_reverse_iterator	??
std::basic_string_view< char >::const_reverse_iterator	??
std::basic_string_view< char16_t >::const_reverse_iterator	??
std::basic_string_view< char32_t >::const_reverse_iterator	??
std::basic_string_view< char8_t >::const_reverse_iterator	??
std::basic_string_view< wchar_t >::const_reverse_iterator	??
std::map< char, Markov::Edge< char > * >::const_reverse_iterator	??
std::map< char, Markov::Node< char > * >::const_reverse_iterator	??
std::map< Markov::API::CLI::Terminal::color, int >::const_reverse_iterator	??

std::map< NodeStorageType, Markov::Edge< NodeStorageType > * >::const_reverse_iterator	??
std::map< NodeStorageType, Markov::Node< NodeStorageType > * >::const_reverse_iterator	??
std::map< storageType, Markov::Edge< storageType > * >::const_reverse_iterator	??
std::vector< Markov::Edge< char > * >::const_reverse_iterator	??
std::vector< Markov::Edge< NodeStorageType > * >::const_reverse_iterator	??
std::vector< Markov::Edge< storageType > * >::const_reverse_iterator	??
Markov::API::CUDA::CUDADeviceController	117
Markov::API::CUDA::CUDAModelMatrix	262
Python.CudaMarkopy.CudaModelMatrixCLI	300
Markov::API::CUDA::Random::Marsaglia	537
Markov::Edge< NodeStorageType >	385
Markov::Edge< char >	385
Markov::Edge< storageType >	385
std::basic_string< char >::iterator	??
std::basic_string< char16_t >::iterator	??
std::basic_string< char32_t >::iterator	??
std::basic_string< char8_t >::iterator	??
std::basic_string< wchar_t >::iterator	??
std::basic_string_view< char >::iterator	??
std::basic_string_view< char16_t >::iterator	??
std::basic_string_view< char32_t >::iterator	??
std::basic_string_view< char8_t >::iterator	??
std::basic_string_view< wchar_t >::iterator	??
std::map< char, Markov::Edge< char > * >::iterator	??
std::map< char, Markov::Node< char > * >::iterator	??
std::map< Markov::API::CLI::Terminal::color, int >::iterator	??
std::map< NodeStorageType, Markov::Edge< NodeStorageType > * >::iterator	??
std::map< NodeStorageType, Markov::Node< NodeStorageType > * >::iterator	??
std::map< storageType, Markov::Edge< storageType > * >::iterator	??
std::vector< Markov::Edge< char > * >::iterator	??
std::vector< Markov::Edge< NodeStorageType > * >::iterator	??
std::vector< Markov::Edge< storageType > * >::iterator	??
Markov::Model< NodeStorageType >	559
Markov::Model< char >	559
Markov::API::MarkovPasswords	479
Markov::API::ModelMatrix	569
Markov::API::CUDA::CUDAModelMatrix	262
Python.Markopy.ModelMatrix	595
Python.Markopy.ModelMatrixCLI	622
Python.Markopy.MarkovModel	462
Python.Markopy.MarkovPasswordsCLI	496
Markov::Node< storageType >	660
Markov::Node< char >	660
Markov::Node< NodeStorageType >	660
QMainWindow	667
Markov::GUI::CLI	114
Markov::GUI::Generate	390
Markov::GUI::MarkovPasswordsGUI	533
Markov::GUI::Train	677
Markov::GUI::about	54
Markov::GUI::menu	552
Markov::Random::RandomEngine	668
Markov::Random::DefaultRandomEngine	380
Markov::Random::Marsaglia	547
Markov::API::CUDA::Random::Marsaglia	537
Markov::Random::Mersenne	555
std::basic_string< char >::reverse_iterator	??

std::basic_string< char16_t >::reverse_iterator	??
std::basic_string< char32_t >::reverse_iterator	??
std::basic_string< char8_t >::reverse_iterator	??
std::basic_string< wchar_t >::reverse_iterator	??
std::basic_string_view< char >::reverse_iterator	??
std::basic_string_view< char16_t >::reverse_iterator	??
std::basic_string_view< char32_t >::reverse_iterator	??
std::basic_string_view< char8_t >::reverse_iterator	??
std::basic_string_view< wchar_t >::reverse_iterator	??
std::map< char, Markov::Edge< char > * >::reverse_iterator	??
std::map< char, Markov::Node< char > * >::reverse_iterator	??
std::map< Markov::API::CLI::Terminal::color, int >::reverse_iterator	??
std::map< NodeStorageType, Markov::Edge< NodeStorageType > * >::reverse_iterator	??
std::map< NodeStorageType, Markov::Node< NodeStorageType > * >::reverse_iterator	??
std::map< storageType, Markov::Edge< storageType > * >::reverse_iterator	??
std::vector< Markov::Edge< char > * >::reverse_iterator	??
std::vector< Markov::Edge< NodeStorageType > * >::reverse_iterator	??
std::vector< Markov::Edge< storageType > * >::reverse_iterator	??
Markov::API::CLI::Terminal	670
Markov::API::Concurrency::ThreadSharedListHandler	673
long int	??
Node< storageType > *	??
NodeStorageType	??

CHAPTER6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Markov::API::CLI::_programOptions	Structure to hold parsed cli arguements	51
Markov::GUI::about	QT Class for about page	54
Python.Markopy.AbstractGenerationModelCLI	Abstract class for generation capable models	55
Python.Markopy.AbstractTrainingModelCLI	Abstract class for training capable models	69
Markov::API::CLI::Argparse	Parse command line arguements	93
Python.Markopy.BaseCLI	Base CLI class to handle user interactions 99	
Markov::GUI::CLI	QT CLI Class	114
std::basic_string< char >::const_iterator	??
std::basic_string< char16_t >::const_iterator	??
std::basic_string< char32_t >::const_iterator	??
std::basic_string< char8_t >::const_iterator	??
std::basic_string< wchar_t >::const_iterator	??
std::basic_string_view< char >::const_iterator	??
std::basic_string_view< char16_t >::const_iterator	??
std::basic_string_view< char32_t >::const_iterator	??
std::basic_string_view< char8_t >::const_iterator	??
std::basic_string_view< wchar_t >::const_iterator	??
std::map< char, Markov::Edge< char > * >::const_iterator	??
std::map< char, Markov::Node< char > * >::const_iterator	??
std::map< Markov::API::CLI::Terminal::color, int >::const_iterator	??
std::map< NodeStorageType, Markov::Edge< NodeStorageType > * >::const_iterator	??
std::map< NodeStorageType, Markov::Node< NodeStorageType > * >::const_iterator	??
std::map< storageType, Markov::Edge< storageType > * >::const_iterator	??
std::vector< Markov::Edge< char > * >::const_iterator	??
std::vector< Markov::Edge< NodeStorageType > * >::const_iterator	??
std::vector< Markov::Edge< storageType > * >::const_iterator	??
std::basic_string< char >::const_reverse_iterator	??
std::basic_string< char16_t >::const_reverse_iterator	??
std::basic_string< char32_t >::const_reverse_iterator	??
std::basic_string< char8_t >::const_reverse_iterator	??
std::basic_string< wchar_t >::const_reverse_iterator	??
std::basic_string_view< char >::const_reverse_iterator	??
std::basic_string_view< char16_t >::const_reverse_iterator	??
std::basic_string_view< char32_t >::const_reverse_iterator	??
std::basic_string_view< char8_t >::const_reverse_iterator	??
std::basic_string_view< wchar_t >::const_reverse_iterator	??
std::map< char, Markov::Edge< char > * >::const_reverse_iterator	??
std::map< char, Markov::Node< char > * >::const_reverse_iterator	??
std::map< Markov::API::CLI::Terminal::color, int >::const_reverse_iterator	??
std::map< NodeStorageType, Markov::Edge< NodeStorageType > * >::const_reverse_iterator	??

std::map< NodeStorageType, Markov::Node< NodeStorageType > * >::const_reverse_iterator	??
std::map< storageType, Markov::Edge< storageType > * >::const_reverse_iterator	??
std::vector< Markov::Edge< char > * >::const_reverse_iterator	??
std::vector< Markov::Edge< NodeStorageType > * >::const_reverse_iterator	??
std::vector< Markov::Edge< storageType > * >::const_reverse_iterator	??
Markov::API::CUDA::CUDADeviceController	
Controller class for CUDA device	117
Python.CudaMarkopy.CudaMarkopyCLI	
CUDA extension to MarkopyCLI	124
Markov::API::CUDA::CUDAModelMatrix	
Extension of Markov::API::ModelMatrix which is modified to run on GPU devices	262
Python.CudaMarkopy.CudaModelMatrixCLI	
Python CLI wrapper for CudaModelMatrix	300
Markov::Random::DefaultRandomEngine	
Implementation using Random.h default random engine	380
Markov::Edge< NodeStorageType >	
Edge class used to link nodes in the model together	385
Markov::GUI::Generate	
QT Generation page class	390
std::basic_string< char >::iterator	??
std::basic_string< char16_t >::iterator	??
std::basic_string< char32_t >::iterator	??
std::basic_string< char8_t >::iterator	??
std::basic_string< wchar_t >::iterator	??
std::basic_string_view< char >::iterator	??
std::basic_string_view< char16_t >::iterator	??
std::basic_string_view< char32_t >::iterator	??
std::basic_string_view< char8_t >::iterator	??
std::basic_string_view< wchar_t >::iterator	??
std::map< char, Markov::Edge< char > * >::iterator	??
std::map< char, Markov::Node< char > * >::iterator	??
std::map< Markov::API::CLI::Terminal::color, int >::iterator	??
std::map< NodeStorageType, Markov::Edge< NodeStorageType > * >::iterator	??
std::map< NodeStorageType, Markov::Node< NodeStorageType > * >::iterator	??
std::map< storageType, Markov::Edge< storageType > * >::iterator	??
std::vector< Markov::Edge< char > * >::iterator	??
std::vector< Markov::Edge< NodeStorageType > * >::iterator	??
std::vector< Markov::Edge< storageType > * >::iterator	??
Python.Markopy.MarkopyCLI	
Top level model selector for Markopy CLI	394
Python.Markopy.MarkovModel	
Abstract representation of a markov model	462
Markov::API::MarkovPasswords	
Markov::Model with char represented nodes	479
Python.Markopy.MarkovPasswordsCLI	
Extension of Python.Markopy.Base.BaseCLI for Markov::API::MarkovPasswords	496
Markov::GUI::MarkovPasswordsGUI	
Reporting UI	533
Markov::API::CUDA::Random::Marsaglia	
Extension of Markov::Random::Marsaglia which is capable of working on device space	537
Markov::Random::Marsaglia	
Implementation of Marsaglia Random Engine	547
Markov::GUI::menu	
QT Menu class	552
Markov::Random::Mersenne	
Implementation of Mersenne Twister Engine	555
Markov::Model< NodeStorageType >	
Class for the final Markov Model, constructed from nodes and edges	559

Markov::API::ModelMatrix	
Class to flatten and reduce Markov::Model to a Matrix	569
Python.Markopy.ModelMatrix	
Abstract representation of a matrix based model	595
Python.Markopy.ModelMatrixCLI	
Extension of Python.Markopy.Base.BaseCLI for Markov::API::ModelMatrix	622
Markov::Node< storageType >	
A node class that for the vertices of model. Connected with eachother using Edge	660
QMainWindow
Markov::Random::RandomEngine	
An abstract class for Random Engine	668
std::basic_string< char >::reverse_iterator
std::basic_string< char16_t >::reverse_iterator
std::basic_string< char32_t >::reverse_iterator
std::basic_string< char8_t >::reverse_iterator
std::basic_string< wchar_t >::reverse_iterator
std::basic_string_view< char >::reverse_iterator
std::basic_string_view< char16_t >::reverse_iterator
std::basic_string_view< char32_t >::reverse_iterator
std::basic_string_view< char8_t >::reverse_iterator
std::basic_string_view< wchar_t >::reverse_iterator
std::map< char, Markov::Edge< char > * >::reverse_iterator
std::map< char, Markov::Node< char > * >::reverse_iterator
std::map< Markov::API::CLI::Terminal::color, int >::reverse_iterator
std::map< NodeStorageType, Markov::Edge< NodeStorageType > * >::reverse_iterator
std::map< NodeStorageType, Markov::Node< NodeStorageType > * >::reverse_iterator
std::map< storageType, Markov::Edge< storageType > * >::reverse_iterator
std::vector< Markov::Edge< char > * >::reverse_iterator
std::vector< Markov::Edge< NodeStorageType > * >::reverse_iterator
std::vector< Markov::Edge< storageType > * >::reverse_iterator
Markov::API::CLI::Terminal	
Pretty colors for Terminal . Windows Only	670
Markov::API::Concurrency::ThreadSharedListHandler	
Simple class for managing shared access to file	673
Markov::GUI::Train	
QT Training page class	677

CHAPTER7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

Markopy/CudaMarkopy/src/CLI/cudamarkopy.py	Base command line interface for python	681
Markopy/CudaMarkopy/src/CLI/cudammx.py	CUDAModelMatrix CLI wrapper	682
Markopy/CudaMarkopy/src/Module/cudaMarkopy.cu	684
Markopy/CudaMarkovAPI/src/cudaDeviceController.cu	Simple static class for basic CUDA device controls	685
Markopy/CudaMarkovAPI/src/cudaDeviceController.h	Simple static class for basic CUDA device controls	687
Markopy/CudaMarkovAPI/src/cudaModelMatrix.cu	CUDA accelerated extension of <code>Markov::API::ModelMatrix</code>	690
Markopy/CudaMarkovAPI/src/cudaModelMatrix.h	CUDA accelerated extension of <code>Markov::API::ModelMatrix</code>	694
Markopy/CudaMarkovAPI/src/cudarandom.h	Extension of <code>Markov::Random::Marsaglia</code> for CUDA	698
Markopy/CudaMarkovAPI/src/main.cu	Simple test file to check libcudamarkov	701
Markopy/documentation/dirs.docs	Doxygen information about the directories	702
Markopy/Markopy/src/CLI/base.py	Base command line interface for python	703
Markopy/Markopy/src/CLI/importer.py	Dynamic import wrapper for markopy model	707
Markopy/Markopy/src/CLI/markopy.py	Entry point for markopy scripts	708
Markopy/Markopy/src/CLI/mm.py	Abstract representation of CPP/Python intermediate layer classes	710
Markopy/Markopy/src/CLI/mmx.py	ModelMatrix CLI wrapper	711
Markopy/Markopy/src/CLI/mp.py	CLI wrapper for MarkovPasswords	712
Markopy/Markopy/src/Module/markopy.cpp	CPython wrapper for libmarkov utils	713
Markopy/MarkovAPI/src/markovPasswords.cpp	Wrapper for <code>Markov::Model</code> to use with char represented models	716
Markopy/MarkovAPI/src/markovPasswords.h	Wrapper for <code>Markov::Model</code> to use with char represented models	720
Markopy/MarkovAPI/src/modelMatrix.cpp	An extension of <code>Markov::API::MarkovPasswords</code>	722
Markopy/MarkovAPI/src/modelMatrix.h	An extension of <code>Markov::API::MarkovPasswords</code>	726
Markopy/MarkovAPI/src/threadSharedListHandler.cpp	Thread-safe wrapper for <code>std::ifstream</code>	730
Markopy/MarkovAPI/src/threadSharedListHandler.h	Thread-safe wrapper for <code>std::ifstream</code>	731
Markopy/MarkovAPICLI/src/argparse.cpp	Arguement handler class for native CPP cli	734

Markopy/MarkovAPICLI/src/ argparse.h	Arguement handler class for native CPP cli	735
Markopy/MarkovAPICLI/src/ main.cpp	Test cases for Markov::API::ModelMatrix	744
Markopy/MarkovAPICLI/src/color/ term.cpp	Terminal handler for pretty stuff like colors	739
Markopy/MarkovAPICLI/src/color/ term.h	Terminal handler for pretty stuff like colors	741
Markopy/MarkovAPICLI/src/scripts/ model_2gram.py		748
Markopy/MarkovAPICLI/src/scripts/ random_model.py		749
Markopy/MarkovModel/src/ dllmain.cpp	DLLMain for dynamic windows library	749
Markopy/MarkovModel/src/ edge.h	Edge class template	751
Markopy/MarkovModel/src/ framework.h	For windows dynamic library	754
Markopy/MarkovModel/src/ model.h	Model class template	755
Markopy/MarkovModel/src/ node.h	Node class template	760
Markopy/MarkovModel/src/ pch.cpp	For windows dynamic library	765
Markopy/MarkovModel/src/ pch.h	For windows dynamic library	766
Markopy/MarkovModel/src/ random.h	Random engine implementations for Markov	768
Markopy/MarkovPasswordsGUI/src/ about.cpp	About page	772
Markopy/MarkovPasswordsGUI/src/ about.h	About page	773
Markopy/MarkovPasswordsGUI/src/ CLI.cpp	CLI page	775
Markopy/MarkovPasswordsGUI/src/ CLI.h	CLI page	776
Markopy/MarkovPasswordsGUI/src/ Generate.cpp	Generation Page	777
Markopy/MarkovPasswordsGUI/src/ Generate.h	Generation Page	780
Markopy/MarkovPasswordsGUI/src/ main.cpp	Entry point for GUI	747
Markopy/MarkovPasswordsGUI/src/ MarkovPasswordsGUI.cpp	Main activity page for GUI	781
Markopy/MarkovPasswordsGUI/src/ MarkovPasswordsGUI.h	Main activity page for GUI	782
Markopy/MarkovPasswordsGUI/src/ menu.cpp	Menu page	784
Markopy/MarkovPasswordsGUI/src/ menu.h	Menu page	785
Markopy/MarkovPasswordsGUI/src/ Train.cpp	Training page for GUI	786
Markopy/MarkovPasswordsGUI/src/ Train.h	Training page for GUI	788
Markopy/UnitTests/ pch.cpp	For windows dynamic library	766
Markopy/UnitTests/ pch.h	For windows dynamic library	768
Markopy/UnitTests/ UnitTests.cpp	Unit tests with Microsoft::VisualStudio::CppUnitTestFramework	790

CHAPTER8

Namespace Documentation

8.1 base Namespace Reference

8.2 cudamarkopy Namespace Reference

Variables

- `spec` = `spec_from_loader("markopy", SourceFileLoader("markopy", os.path.abspath("markopy.py")))`
- `markopy` = `module_from_spec(spec)`
- `mp` = `CudaMarkopyCLI()`

8.2.1 Variable Documentation

8.2.1.1 markopy

```
cudamarkopy.markopy = module_from_spec(spec)  
Definition at line 20 of file cudamarkopy.py.
```

8.2.1.2 mp

```
cudamarkopy.mp = CudaMarkopyCLI()  
Definition at line 107 of file cudamarkopy.py.
```

8.2.1.3 spec

```
cudamarkopy.spec = spec_from_loader("markopy", SourceFileLoader("markopy", os.path.abspath("markopy.py")))  
Definition at line 19 of file cudamarkopy.py.
```

8.3 cudammx Namespace Reference

Variables

- string `ext` = "so"
- `spec` = `spec_from_loader("cudamarkopy", ExtensionFileLoader("cudamarkopy", os.path.abspath(f"cudamarkopy.{ext}")))`
- `cudamarkopy` = `module_from_spec(spec)`
- `markopy` = `module_from_spec(spec)`
- `mp` = `CudaModelMatrixCLI()`

8.3.1 Variable Documentation

8.3.1.1 cudamarkopy

```
cudammx.cudamarkopy = module_from_spec(spec)  
Definition at line 20 of file cudammx.py.
```

8.3.1.2 ext

```
string cudammx.ext = "so"
Definition at line 15 of file cudammx.py.
```

8.3.1.3 markopy

```
cudammx.markopy = module_from_spec(spec)
Definition at line 35 of file cudammx.py.
```

8.3.1.4 mp

```
cudammx.mp = CudaModelMatrixCLI()
Definition at line 79 of file cudammx.py.
```

8.3.1.5 spec

```
cudammx.spec = spec_from_loader("cudamarkopy", ExtensionFileLoader("cudamarkopy", os.path.←
abspath(f"cudamarkopy.{ext}")))
Definition at line 19 of file cudammx.py.
```

8.4 importer Namespace Reference

Functions

- def [import_markopy \(\)](#)
import and return markopy module

8.4.1 Function Documentation

8.4.1.1 import_markopy()

```
def importer.import_markopy ( )
import and return markopy module
```

Returns

markopy module

Definition at line 12 of file importer.py.

```
00012 def import_markopy():
00013     """! @brief import and return markopy module
00014         @returns markopy module
00015     """
00016     ext = "so"
00017     if os.name == 'nt':
00018         ext="pyd"
00019     try:
00020         spec = spec_from_loader("markopy", ExtensionFileLoader("markopy",
00021             os.path.abspath(f"markopy.{ext}")))
00022         markopy = module_from_spec(spec)
00023         spec.loader.exec_module(markopy)
00024     except ImportError as e:
00025         print(f"({_file_}) Working in development mode. Trying to load markopy.{ext} from
00026             ../../../{out}/")
00027         if(os.path.exists(f"../../../{out}/lib/markopy.{ext}")):
00028             spec = spec_from_loader("markopy", ExtensionFileLoader("markopy",
00029                 os.path.abspath(f"../../../{out}/lib/markopy.{ext}")))
00030             markopy = module_from_spec(spec)
00031             spec.loader.exec_module(markopy)
00032         else:
00033             raise e
```

8.5 markopy Namespace Reference

Variables

- string `ext` = "so"
- `spec` = spec_from_loader("markopy", ExtensionFileLoader("markopy", os.path.abspath(f"markopy.{ext}")))
- `markopy` = module_from_spec(`spec`)
- `mp` = MarkopyCLI()

8.5.1 Variable Documentation

8.5.1.1 ext

`string markopy.ext = "so"`

Definition at line 24 of file [markopy.py](#).

8.5.1.2 markopy

`markopy.markopy = module_from_spec(spec)`

Definition at line 29 of file [markopy.py](#).

8.5.1.3 mp

`markopy.mp = MarkopyCLI()`

Definition at line 162 of file [markopy.py](#).

8.5.1.4 spec

`markopy.spec = spec_from_loader("markopy", ExtensionFileLoader("markopy", os.path.abspath(f"markopy.{ext}")))`

Definition at line 28 of file [markopy.py](#).

8.6 Markov Namespace Reference

Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.

Namespaces

- [API](#)
Namespace for the [MarkovPasswords API](#).
- [GUI](#)
namespace for [MarkovPasswords API GUI](#) wrapper
- [Markopy](#)
C_Python module for [Markov::API](#) objects.
- [Random](#)
Objects related to RNG.

Classes

- class [Node](#)
A node class that for the vertices of model. Connected with eachother using [Edge](#).
- class [Edge](#)

[Edge](#) class used to link nodes in the model together.

- class [Model](#)

class for the final [Markov Model](#), constructed from nodes and edges.

8.6.1 Detailed Description

Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.

8.7 Markov::API Namespace Reference

Namespace for the [MarkovPasswords API](#).

Namespaces

- [CLI](#)

Structure to hold parsed cli arguements.

- [Concurrency](#)

Namespace for [Concurrency](#) related classes.

- [CUDA](#)

Namespace for objects requiring [CUDA](#) libraries.

Classes

- class [MarkovPasswords](#)

[Markov::Model](#) with char represented nodes.

- class [ModelMatrix](#)

Class to flatten and reduce [Markov::Model](#) to a Matrix.

8.7.1 Detailed Description

Namespace for the [MarkovPasswords API](#).

8.8 Markov::API::CLI Namespace Reference

Structure to hold parsed cli arguements.

Classes

- struct [_programOptions](#)

Structure to hold parsed cli arguements.

- class [Argparse](#)

Parse command line arguements.

- class [Terminal](#)

pretty colors for [Terminal](#). Windows Only.

Typedefs

- typedef struct [Markov::API::CLI::_programOptions](#) ProgramOptions

Structure to hold parsed cli arguements.

Functions

- std::ostream & [operator<<](#) (std::ostream &os, const [Markov::API::CLI::Terminal::color](#) &c)

8.8.1 Detailed Description

Structure to hold parsed cli arguements.
Namespace for the [CLI](#) objects

8.8.2 Typedef Documentation

8.8.2.1 ProgramOptions

```
typedef struct Markov::API::CLI::__programOptions Markov::API::CLI::ProgramOptions
```

Structure to hold parsed cli arguements.

8.8.3 Function Documentation

8.8.3.1 operator<<()

```
std::ostream& Markov::API::CLI::operator<< (
    std::ostream & os,
    const Markov::API::CLI::Terminal::color & c )
```

overload for std::cout.

8.9 Markov::API::Concurrency Namespace Reference

Namespace for [Concurrency](#) related classes.

Classes

- class [ThreadSharedListHandler](#)
Simple class for managing shared access to file.

8.9.1 Detailed Description

Namespace for [Concurrency](#) related classes.

8.10 Markov::API::CUDA Namespace Reference

Namespace for objects requiring [CUDA](#) libraries.

Namespaces

- [Random](#)
Namespace for [Random](#) engines operable under [device](#) space.

Classes

- class [CUDADeviceController](#)
Controller class for [CUDA](#) device.
- class [CUDAModelMatrix](#)
Extension of [Markov::API::ModelMatrix](#) which is modified to run on GPU devices.

Functions

- `__global__ void FastRandomWalkCUDAKernel (unsigned long int n, int minLen, int maxLen, char *outputBuffer, char *matrixIndex, long int *totalEdgeWeights, long int *valueMatrix, char *edgeMatrix, int matrixSize, int memoryPerKernelGrid, unsigned long *seed)`
`CUDA kernel for the FastRandomWalk operation.`
- `__device__ char * strchr (char *p, char c, int s_len)`
`strchr implementation on device space`

8.10.1 Detailed Description

Namespace for objects requiring CUDA libraries.

8.10.2 Function Documentation

8.10.2.1 FastRandomWalkCUDAKernel()

```
__global__ void Markov::API::CUDA::FastRandomWalkCUDAKernel (
    unsigned long int n,
    int minLen,
    int maxLen,
    char * outputBuffer,
    char * matrixIndex,
    long int * totalEdgeWeights,
    long int * valueMatrix,
    char * edgeMatrix,
    int matrixSize,
    int memoryPerKernelGrid,
    unsigned long * seed )
```

CUDA kernel for the FastRandomWalk operation.

Will be initiated by CPU and continued by GPU (**global** tag)

Parameters

<code>n</code>	- Number of passwords to generate.
<code>minlen</code>	- minimum string length for a single generation
<code>maxLen</code>	- maximum string length for a single generation
<code>outputBuffer</code>	- VRAM ptr to the output buffer
<code>matrixIndex</code>	- VRAM ptr to the matrix indices
<code>totalEdgeWeights</code>	- VRAM ptr to the totalEdgeWeights array
<code>valueMatrix</code>	- VRAM ptr to the edge weights array
<code>edgeMatrix</code>	- VRAM ptr to the edge representations array
<code>matrixSize</code>	- Size of the matrix dimensions
<code>memoryPerKernelGrid</code>	- Maximum memory usage per kernel grid
<code>seed</code>	- seed chunk to generate the random from (generated & used by Marsaglia)

Definition at line 194 of file [cudaModelMatrix.cu](#).

```
00195
00196     int kernelWorkerIndex = threadIdx.x + blockIdx.x * blockDim.x;
00197
00198     if(n==0) return;
00199
00200     char* e;
00201     int index = 0;
00202     char next;
00203     int len=0;
```

```

00205     long int selection;
00206     char cur;
00207     long int bufferctr = 0;
00208     unsigned long int *x,*y,*z,t;
00209     char* res = &outputBuffer[kernelWorkerIndex*memoryPerKernelGrid];
00210     x=&seed[kernelWorkerIndex*3];
00211     y=&seed[kernelWorkerIndex*3+1];
00212     z=&seed[kernelWorkerIndex*3+2];
00213     for (int i = 0; i < n; i++) {
00214         cur=199;
00215         len=0;
00216         while (true) {
00217             e = strchr(matrixIndex, cur, matrixSize);
00218             index = e - matrixIndex;
00219             /*selection = Markov::API::Random::devrandom(
00220                 seed[kernelWorkerIndex*3],
00221                 seed[kernelWorkerIndex*3+1],
00222                 seed[kernelWorkerIndex*3+2]) % totalEdgeWeights[index];*/
00223             *xx ^= *xx << 16;
00224             *xx ^= *xx >> 5;
00225             *xx ^= *xx << 1;
00226
00227             t = *xx;
00228             *xx = *y;
00229             *y = *z;
00230             *z = t ^ *x ^ *y;
00231             selection = *z % totalEdgeWeights[index];
00232             for(int j=0;j<matrixSize-1;j++){
00233                 selection -= valueMatrix[index*matrixSize + j];
00234                 if (selection < 0){
00235                     next = edgeMatrix[index* sizeof(char)*matrixSize + j];
00236                     break;
00237                 }
00238             }
00239
00240             if (len >= maxLen) break;
00241             else if ((next < 0) && (len < minLen)) continue;
00242             else if (next < 0) break;
00243             cur = next;
00244             res[bufferctr + len++] = cur;
00245         }
00246         res[bufferctr + len++] = '\n';
00247         bufferctr+=len;
00248     }
00249     res[bufferctr] = '\0';
00250 }
```

8.10.2.2 strchr()

```
__device__ char * Markov::API::strchr (
    char * p,
    char c,
    int s_len )
```

srtchr implementation on **device** space

Fint the first matching index of a string

Parameters

<i>p</i>	- string to check
<i>c</i>	- character to match
<i>s_len</i>	- maximum string length

Returns

pointer to the match

Definition at line 252 of file [cudaModelMatrix.cu](#).

```

00252     for (;;) ++p, s_len--) {
00253         if (*p == c)
00254             return((char *)p);
00255         if (!*p)
00256             return((char *)NULL);
00257     }
00259 }
```

8.11 Markov::API::CUDA::Random Namespace Reference

Namespace for [Random](#) engines operable under **device** space.

Classes

- class [Marsaglia](#)

*Extension of [Markov::Random::Marsaglia](#) which is capable of working on **device** space.*

Functions

- `__device__ unsigned long devrandom (unsigned long &x, unsigned long &y, unsigned long &z)`
*Marsaglia Random Generation function operable in **device** space.*

8.11.1 Detailed Description

Namespace for [Random](#) engines operable under **device** space.

8.11.2 Function Documentation

8.11.2.1 devrandom()

```
__device__ unsigned long Markov::API::CUDA::Random::devrandom (
    unsigned long &x,
    unsigned long &y,
    unsigned long &z )
```

[Marsaglia Random](#) Generation function operable in **device** space.

Parameters

<code>x</code>	<code>marsaglia internal x. Not constant, (ref)</code>
<code>y</code>	<code>marsaglia internal y. Not constant, (ref)</code>
<code>z</code>	<code>marsaglia internal z. Not constant, (ref)</code>

Returns

`returns z`

Definition at line 54 of file [cudarandom.h](#).

```
00054
00055     unsigned long t;
00056     x ^= x << 16;
00057     x ^= x >> 5;
00058     x ^= x << 1;
00059
00060     t = x;
00061     x = y;
00062     y = z;
00063     z = t ^ x ^ y;
00064
00065     return z;
00066 }
```

8.12 Markov::GUI Namespace Reference

namespace for MarkovPasswords [API GUI](#) wrapper

Classes

- class [about](#)

QT Class for about page.

- class [CLI](#)
QT CLI Class.
- class [Train](#)
QT Training page class.
- class [Generate](#)
QT Generation page class.
- class [MarkovPasswordsGUI](#)
Reporting UI.
- class [menu](#)
QT Menu class.

8.12.1 Detailed Description

namespace for [MarkovPasswords API GUI](#) wrapper

8.13 Markov::Markopy Namespace Reference

C_Python module for [Markov::API](#) objects.

Namespaces

- [CUDA](#)
C_Python module for [Markov::API::CUDA](#) objects.

Functions

- [BOOST_PYTHON_MODULE](#) (markopy)

8.13.1 Detailed Description

C_Python module for [Markov::API](#) objects.

8.13.2 Function Documentation

8.13.2.1 BOOST_PYTHON_MODULE()

```
Markov::Markopy::BOOST_PYTHON_MODULE (
    markopy )
```

Definition at line 37 of file [markopy.cpp](#).

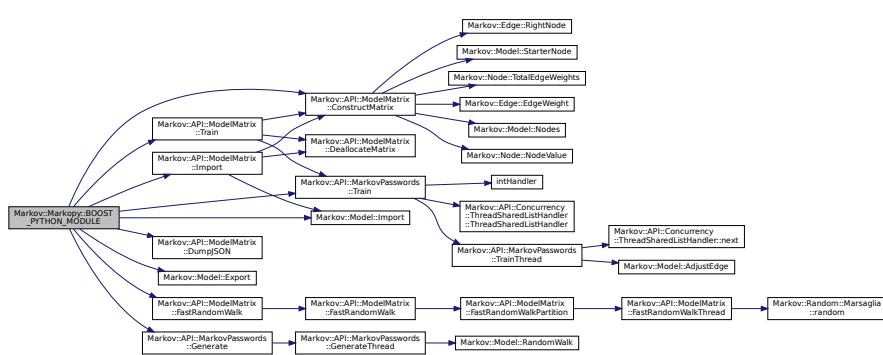
```
00038     {
00039         bool (Markov::API::MarkovPasswords::*Import) (const char\*) = &Markov::Model<char>::Import;
00040         bool (Markov::API::MarkovPasswords::*Export) (const char\*) = &Markov::Model<char>::Export;
00041         class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
00042             .def(init<>())
00043             .def("Train", &Markov::API::MarkovPasswords::Train)
00044             .def("Generate", &Markov::API::MarkovPasswords::Generate)
00045             .def("Import", Import, "Import a model file.")
00046             .def("Export", Export, "Export a model to file.")
00047     ;
00048
00049         int (Markov::API::ModelMatrix::*FastRandomWalk) (unsigned long int, const char\*, int, int, int,
00050             bool) = &Markov::API::ModelMatrix::FastRandomWalk;
00051         class_<Markov::API::ModelMatrix>("ModelMatrix", init<>())
00052             .def(init<>())
00053             .def("Train", &Markov::API::ModelMatrix::Train)
00054             .def("Import", &Markov::API::ModelMatrix::Import, "Import a model file.")
00055             .def("Export", Export, "Export a model to file.")
00056             .def("ConstructMatrix", &Markov::API::ModelMatrix::ConstructMatrix)
```

```

00058     .def("DumpJSON", &Markov::API::ModelMatrix::DumpJSON)
00059     .def("FastRandomWalk", FastRandomWalk)
00060 ;
00061 };

```

References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), [Markov::Model< NodeStorageType >::Import\(\)](#), [Markov::API::ModelMatrix::FastRandomWalk\(\)](#), [Markov::API::MarkovPasswords::Generate\(\)](#), [Markov::API::ModelMatrix::Import\(\)](#), [Markov::Model< NodeStorageType >::Import\(\)](#), [Markov::API::MarkovPasswords::Train\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).
Here is the call graph for this function:



8.14 Markov::Markopy::CUDA Namespace Reference

C Python module for [Markov::API::CUDA](#) objects.

Functions

- [BOOST_PYTHON_MODULE](#) (cudamarkopy)

8.14.1 Detailed Description

C Python module for [Markov::API::CUDA](#) objects.

8.14.2 Function Documentation

8.14.2.1 BOOST_PYTHON_MODULE()

```
Markov::Markopy::CUDA::BOOST_PYTHON_MODULE (
    cudamarkopy )
```

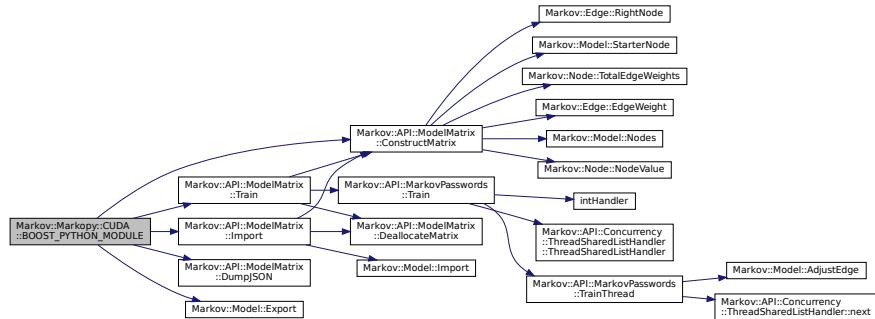
Definition at line 20 of file [cudaMarkopy.cu](#).

```

00021 {
00022     bool (Markov::API::MarkovPasswords::Export) (const char*) = &Markov::Model<char>::Export;
00023     void (Markov::API::CUDA::CUDAModelMatrix::*FastRandomWalk) (unsigned long int, const char*, int, int, bool, bool) = &Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk;
00024     class_<Markov::API::CUDA::CUDAModelMatrix> ("CUDAModelMatrix", init<>())
00025         .def(init<>())
00026         .def("Train", &Markov::API::ModelMatrix::Train)
00027         .def("Import", &Markov::API::ModelMatrix::Import, "Import a model file.")
00028         .def("Export", Export, "Export a model to file.")
00029         .def("ConstructMatrix", &Markov::API::ModelMatrix::ConstructMatrix)
00030         .def("DumpJSON", &Markov::API::ModelMatrix::DumpJSON)
00031         .def("FastRandomWalk", FastRandomWalk)
00032     ;
00033 }
```

References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), [Markov::Model< NodeStorageType >::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



8.15 Markov::Random Namespace Reference

Objects related to RNG.

Classes

- class **RandomEngine**
An abstract class for Random Engine.
 - class **DefaultRandomEngine**
Implementation using Random.h default random engine.
 - class **Marsaglia**
Implementation of Marsaglia Random Engine.
 - class **Mersenne**
Implementation of Mersenne Twister Engine.

8.15.1 Detailed Description

Objects related to RNG.

8.16 mm Namespace Reference

Variables

- `markopy = import_markopy()`

8.16.1 Variable Documentation

8.16.1.1 markopy

```
mm.markopy = import_markopy()  
Definition at line 11 of file mm.py.
```

8.17 mmx Namespace Reference

Variables

- `markopy = import_markopy()`
 - `mp = ModelMatrixCLI()`

8.17.1 Variable Documentation

8.17.1.1 markopy

```
mmx.markopy = import_markopy()  
Definition at line 12 of file mmx.py.
```

8.17.1.2 mp

```
mmx.mp = ModelMatrixCLI()  
Definition at line 44 of file mmx.py.
```

8.18 model_2gram Namespace Reference

Variables

- **alphabet** = string.printable
password alphabet
- **f** = open('..../models/2gram.mdl', "wb")
output file handle

8.18.1 Detailed Description

python script for generating a 2gram model

8.18.2 Variable Documentation

8.18.2.1 alphabet

```
model_2gram.alphabet = string.printable  
password alphabet  
Definition at line 10 of file model\_2gram.py.
```

8.18.2.2 f

```
model_2gram.f = open('..../models/2gram.mdl', "wb")  
output file handle  
Definition at line 16 of file model\_2gram.py.
```

8.19 mp Namespace Reference

Variables

- **markopy** = import_markopy()
- **mp** = MarkovPasswordsCLI()

8.19.1 Variable Documentation

8.19.1.1 markopy

```
mp.markopy = import_markopy()  
Definition at line 13 of file mp.py.
```

8.19.1.2 mp

```
mp.mp = MarkovPasswordsCLI()  
Definition at line 36 of file mp.py.
```

8.20 Python.CudaMarkopy Namespace Reference

wrapper scripts for [CudaMarkopy](#)

Classes

- class [CudaMarkopyCLI](#)
CUDA extension to MarkopyCLI.
- class [CudaModelMatrixCLI](#)
Python CLI wrapper for CudaModelMatrix.

8.20.1 Detailed Description

wrapper scripts for [CudaMarkopy](#)

8.21 Python.Markopy Namespace Reference

wrapper scripts for [Markopy](#)

Classes

- class [BaseCLI](#)
Base CLI class to handle user interactions
- class [AbstractGenerationModelCLI](#)
abstract class for generation capable models
- class [AbstractTrainingModelCLI](#)
abstract class for training capable models
- class [MarkopyCLI](#)
Top level model selector for Markopy CLI.
- class [MarkovModel](#)
Abstract representation of a markov model.
- class [ModelMatrix](#)
Abstract representation of a matrix based model.
- class [ModelMatrixCLI](#)
Extension of Python.Markopy.Base.BaseCLI for Markov::API::ModelMatrix.
- class [MarkovPasswordsCLI](#)
Extension of Python.Markopy.Base.BaseCLI for Markov::API::MarkovPasswords.

8.21.1 Detailed Description

wrapper scripts for [Markopy](#)

8.22 random_model Namespace Reference

Variables

- **alphabet** = string.printable
password alphabet
- **f** = open('../models/random.mdl', "wb")
output file handle

8.22.1 Detailed Description

python script for generating a 2gram model

8.22.2 Variable Documentation

8.22.2.1 alphabet

```
random_model.alphabet = string.printable
password alphabet
Definition at line 10 of file random\_model.py.
```

8.22.2.2 f

```
random_model.f = open('../models/random.mdl', "wb")
output file handle
Definition at line 16 of file random\_model.py.
```

8.23 Testing Namespace Reference

Namespace for Microsoft Native Unit [Testing](#) Classes.

Namespaces

- [MarkovModel](#)
Testing namespace for [MarkovModel](#).
- [MarkovPasswords](#)
Testing namespace for [MarkovPasswords](#).
- [MVP](#)
Testing Namespace for Minimal Viable Product.

8.23.1 Detailed Description

Namespace for Microsoft Native Unit [Testing](#) Classes.

8.24 Testing::MarkovModel Namespace Reference

[Testing](#) namespace for [MarkovModel](#).

Functions

- [TEST_CLASS \(Edge\)](#)

Test class for rest of Edge cases.

- [TEST_CLASS \(Node\)](#)

Test class for rest of Node cases.

- [TEST_CLASS \(Model\)](#)

Test class for rest of model cases.

8.24.1 Detailed Description

Testing namespace for [MarkovModel](#).

8.24.2 Function Documentation

8.24.2.1 TEST_CLASS() [1/3]

```
Testing::MarkovModel::TEST_CLASS (
    Edge )
```

Test class for rest of Edge cases.

send exception on integer underflow

test integer overflows

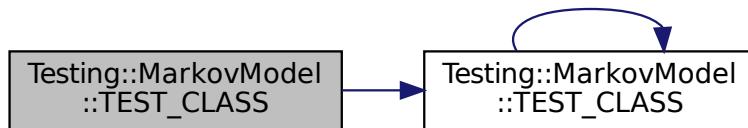
Definition at line 500 of file [UnitTests.cpp](#).

```
00501     {
00502         public:
00503             TEST_METHOD(except_integer_underflow) {
00504                 auto _underflow_adjust = [] {
00505                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00506                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00507                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00508                         RightNode);
00509                     e->AdjustEdge(15);
00510                     e->AdjustEdge(-30);
00511                     delete LeftNode;
00512                     delete RightNode;
00513                     delete e;
00514                 };
00515                 Assert::ExpectException<std::underflow_error>(_underflow_adjust);
00516             }
00517         }
00518
00519         TEST_METHOD(except_integer_overflow) {
00520             auto _overflow_adjust = [] {
00521                 Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00522                 Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00523                 Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00524                         RightNode);
00525                     e->AdjustEdge(~0ull);
00526                     e->AdjustEdge(1);
00527                     delete LeftNode;
00528                     delete RightNode;
00529                     delete e;
00530                 };
00531                 Assert::ExpectException<std::underflow_error>(_overflow_adjust);
00532             }
00533         };
00534     };

```

References [TEST_CLASS\(\)](#).

Here is the call graph for this function:



8.24.2.2 TEST_CLASS() [2/3]

```
Testing::MarkovModel::TEST_CLASS (
    Model )
```

Test class for rest of model cases.

Definition at line 598 of file [UnitTests.cpp](#).

```

00599         {
00600     public:
00601     TEST_METHOD(functional_random_walk) {
00602         unsigned char* res2 = new unsigned char[12 + 5];
00603         Markov::Random::Marsaglia MarsagliaRandomEngine;
00604         Markov::Model<unsigned char> m;
00605         Markov::Node<unsigned char>* starter = m.StarterNode();
00606         Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00607         Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00608         Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00609         Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00610         starter->Link(a)->AdjustEdge(1);
00611         a->Link(b)->AdjustEdge(1);
00612         b->Link(c)->AdjustEdge(1);
00613         c->Link(end)->AdjustEdge(1);
00614
00615         char* res = (char*)m.RandomWalk(&MarsagliaRandomEngine, 1, 12, res2);
00616         Assert::IsFalse(strcmp(res, "abc"));
00617     }
00618     TEST_METHOD(functionoal_random_walk_without_any) {
00619         Markov::Model<unsigned char> m;
00620         Markov::Node<unsigned char>* starter = m.StarterNode();
00621         Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00622         Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00623         Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00624         Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00625         Markov::Edge<unsigned char>* res = NULL;
00626         starter->Link(a)->AdjustEdge(1);
00627         a->Link(b)->AdjustEdge(1);
00628         b->Link(c)->AdjustEdge(1);
00629         c->Link(end)->AdjustEdge(1);
00630
00631         res = starter->FindEdge('D');
00632         Assert::IsNull(res);
00633     }
00634 }
```

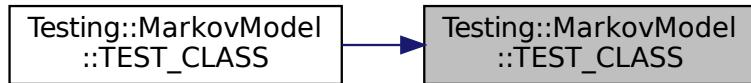
References [TEST_CLASS\(\)](#).

Referenced by [TEST_CLASS\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.24.2.3 TEST_CLASS() [3/3]

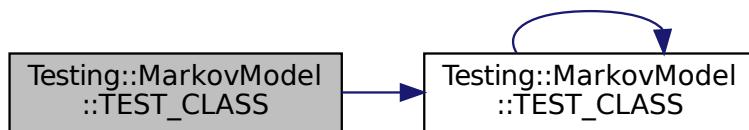
```

Testing::MarkovModel::TEST_CLASS (
    Node )
Test class for rest of Node cases.
test RandomNext with 64 bit high values
test RandomNext with 64 bit high values
randomNext when no edges are present
Definition at line 538 of file UnitTests.cpp.
00539     {
00540         public:
00541
00542             TEST_METHOD(rand_next_u64) {
00543                 Markov::Random::Marsaglia MarsagliaRandomEngine;
00544                 Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00545                 Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00546                 Markov::Edge<unsigned char>* e = src->Link(target1);
00547                 e->AdjustEdge((unsigned long)(ull << 63));
00548                 Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00549                 Assert::IsTrue(res == target1);
00550                 delete src;
00551                 delete target1;
00552                 delete e;
00553
00554             }
00555
00556             TEST_METHOD(rand_next_u64_max) {
00557                 Markov::Random::Marsaglia MarsagliaRandomEngine;
00558                 Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00559                 Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00560                 Markov::Edge<unsigned char>* e = src->Link(target1);
00561                 e->AdjustEdge((0xffffffff));
00562                 Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00563                 Assert::IsTrue(res == target1);
00564                 delete src;
00565                 delete target1;
00566                 delete e;
00567             }
00568
00569         }
00570     }
00571 }
```

```

00573     TEST_METHOD(uninitialized_rand_next) {
00574
00575         auto _invalid_next = [] {
00576             Markov::Random::Marsaglia MarsagliaRandomEngine;
00577             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00578             Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00579             Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(src, target1);
00580             Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00581
00582             delete src;
00583             delete target1;
00584             delete e;
00585         };
00586
00587         Assert::ExpectException<std::logic_error>(_invalid_next);
00588     }
00589
00590     References TEST\_CLASS\(\).
00591 
```

Here is the call graph for this function:



8.25 Testing::MarkovPasswords Namespace Reference

[Testing](#) namespace for [MarkovPasswords](#).

8.25.1 Detailed Description

[Testing](#) namespace for [MarkovPasswords](#).

8.26 Testing::MVP Namespace Reference

[Testing](#) Namespace for Minimal Viable Product.

Namespaces

- [MarkovModel](#)
Testing Namespace for MVP MarkovModel.
- [MarkovPasswords](#)
Testing namespace for MVP MarkovPasswords.

8.26.1 Detailed Description

[Testing](#) Namespace for Minimal Viable Product.

8.27 Testing::MVP::MarkovModel Namespace Reference

[Testing](#) Namespace for [MVP MarkovModel](#).

Functions

- **TEST_CLASS** (Edge)
Test class for minimal viable Edge.
- **TEST_CLASS** (Node)
Test class for minimal viable Node.
- **TEST_CLASS** (Model)
Test class for minimal viable Model.

8.27.1 Detailed Description

Testing Namespace for MVP MarkovModel.

8.27.2 Function Documentation

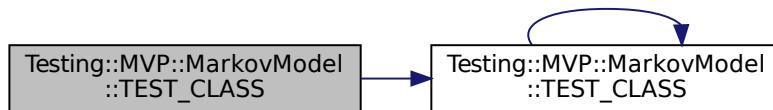
8.27.2.1 TEST_CLASS() [1/3]

```
Testing::MVP::MarkovModel::TEST_CLASS (
    Edge
)
Test class for minimal viable Edge.
test default constructor
test linked constructor with two nodes
test AdjustEdge function
test TraverseNode returning RightNode
test LeftNode/RightNode setter
test negative adjustments
Definition at line 27 of file UnitTests.cpp.
00028     {
00029         public:
00030
00033             TEST_METHOD(default_constructor) {
00034                 Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>;
00035                 Assert::IsNull(e->LeftNode());
00036                 Assert::IsNull(e->RightNode());
00037                 delete e;
00038             }
00039
00042             TEST_METHOD(linked_constructor) {
00043                 Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00044                 Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00045                 Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00046                                         RightNode);
00047                 Assert::IsTrue(LeftNode == e->LeftNode());
00048                 Assert::IsTrue(RightNode == e->RightNode());
00049                 delete LeftNode;
00050                 delete RightNode;
00051                 delete e;
00052             }
00055             TEST_METHOD(AdjustEdge) {
00056                 Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00057                 Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00058                 Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00059                                         RightNode);
00060                 e->AdjustEdge(15);
00061                 Assert::AreEqual(15ull, e->EdgeWeight());
00062                 e->AdjustEdge(15);
00063                 Assert::AreEqual(30ull, e->EdgeWeight());
00064                 delete LeftNode;
00065                 delete RightNode;
00066                 delete e;
00067             }
00070             TEST_METHOD(TraverseNode) {
00071                 Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00072                 Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00073                 Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00074                                         RightNode);
00075                 Assert::IsTrue(RightNode == e->TraverseNode());
00076                 delete LeftNode;
00077                 delete RightNode;
```

```
00077     delete e;
00078 }
00079
00082 TEST_METHOD(set_left_and_right) {
00083     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00084     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00085     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(LeftNode,
00086     RightNode);
00087
00088     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>;
00089     e2->SetLeftEdge(LeftNode);
00090     e2->SetRightEdge(RightNode);
00091
00092     Assert::IsTrue(e1->LeftNode() == e2->LeftNode());
00093     Assert::IsTrue(e1->RightNode() == e2->RightNode());
00094     delete LeftNode;
00095     delete RightNode;
00096     delete e1;
00097     delete e2;
00098 }
00099
00101 TEST_METHOD(negative_adjust) {
00102     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00103     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00104     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00105     RightNode);
00106     e->AdjustEdge(15);
00107     Assert::AreEqual(15ull, e->EdgeWeight());
00108     e->AdjustEdge(-15);
00109     Assert::AreEqual(0ull, e->EdgeWeight());
00110     delete LeftNode;
00111     delete RightNode;
00112     delete e;
00113 }
00114 };
```

References TEST_CLASS().

Here is the call graph for this function:



8.27.2.2 TEST_CLASS() [2/3]

```
Testing:::MVP::MarkovModel::TEST_CLASS (
```

Test class for minimal viable Model.

test model constructor for starter node

test import

test export

test random walk

Definition at line 353 of file UnitTests.cpp.

00354

00355
00358

00359

```
00361     }
00362
00363     TEST_METHOD(import_filename) {
00364         Markov::Model<unsigned char> m;
00365         Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
00366     }
00367
00368     TEST_METHOD(export_filename) {
00369         Markov::Model<unsigned char> m;
00370         Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
00371     }
00372 }
```

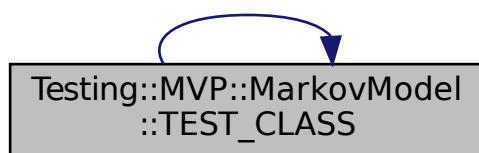
```

00375         }
00376
00377     TEST_METHOD(random_walk) {
00378         unsigned char* res = new unsigned char[12 + 5];
00379         Markov::Random::Marsaglia MarsagliaRandomEngine;
00380         Markov::Model<unsigned char> m;
00381         Assert::IsTrue(m.Import("../Models/finished2.mdl"));
00382         Assert::IsNotNull(m.RandomWalk(&MarsagliaRandomEngine, 1, 12, res));
00383     }
00384 }
```

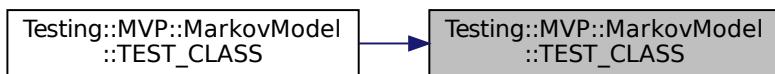
References [TEST_CLASS\(\)](#).

Referenced by [TEST_CLASS\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.27.2.3 TEST_CLASS() [3/3]

```
Testing::MVP::MarkovModel::TEST_CLASS (
    Node )
```

Test class for minimal viable Node.

test default constructor

test custom constructor with unsigned char

test link function

test link function

test RandomNext with low values

test RandomNext with 32 bit high values

random next on a node with no follow-ups

random next on a node with no follow-ups

test updateEdges

test updateEdges

test FindVertice

test FindVertice

test FindVertice

Definition at line 117 of file [UnitTests.cpp](#).

```

00118     {
00119         public:
00120             TEST_METHOD(default_constructor) {
```

```

00124     Markov::Node<unsigned char>* n = new Markov::Node<unsigned char>();
00125     Assert::AreEqual((unsigned char)0, n->NodeValue());
00126     delete n;
00127 }
00128
00129 TEST_METHOD(uchar_constructor) {
00130     Markov::Node<unsigned char>* n = NULL;
00131     unsigned char test_cases[] = { 'c', 0x00, 0xff, -32 };
00132     for (unsigned char tcase : test_cases) {
00133         n = new Markov::Node<unsigned char>(tcase);
00134         Assert::AreEqual(tcase, n->NodeValue());
00135         delete n;
00136     }
00137 }
00138
00139 TEST_METHOD(link_left) {
00140     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00141     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00142
00143     Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
00144     delete LeftNode;
00145     delete RightNode;
00146     delete e;
00147 }
00148
00149 TEST_METHOD(link_right) {
00150     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00151     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00152
00153     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(NULL, RightNode);
00154     LeftNode->Link(e);
00155     Assert::IsTrue(LeftNode == e->LeftNode());
00156     Assert::IsTrue(RightNode == e->RightNode());
00157     delete LeftNode;
00158     delete RightNode;
00159     delete e;
00160 }
00161
00162 TEST_METHOD(rand_next_low) {
00163     Markov::Random::Marsaglia MarsagliaRandomEngine;
00164     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00165     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00166     Markov::Edge<unsigned char>* e = src->Link(target1);
00167     e->AdjustEdge(15);
00168     Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00169     Assert::IsTrue(res == target1);
00170     delete src;
00171     delete target1;
00172     delete e;
00173 }
00174
00175 TEST_METHOD(rand_next_u32) {
00176     Markov::Random::Marsaglia MarsagliaRandomEngine;
00177     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00178     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00179     Markov::Edge<unsigned char>* e = src->Link(target1);
00180     e->AdjustEdge(1 << 31);
00181     Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00182     Assert::IsTrue(res == target1);
00183     delete src;
00184     delete target1;
00185     delete e;
00186 }
00187
00188 TEST_METHOD(rand_next_choice_1) {
00189     Markov::Random::Marsaglia MarsagliaRandomEngine;
00190     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00191     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00192     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00193     Markov::Edge<unsigned char>* el = src->Link(target1);
00194     Markov::Edge<unsigned char>* e2 = src->Link(target2);
00195     el->AdjustEdge(1);
00196     e2->AdjustEdge((unsigned long)(ull << 31));
00197     Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00198     Assert::IsNotNull(res);
00199     Assert::IsTrue(res == target2);
00200     delete src;
00201     delete target1;
00202     delete el;
00203     delete e2;
00204 }
00205
00206 TEST_METHOD(rand_next_choice_2) {
00207     Markov::Random::Marsaglia MarsagliaRandomEngine;
00208     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224

```

```

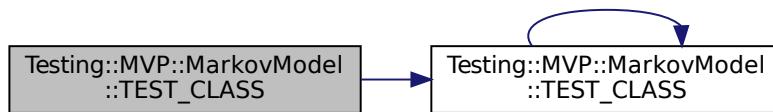
00225
00226     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00227     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00228     Markov::Edge<unsigned char>* e1 = src->Link(target1);
00229     Markov::Edge<unsigned char>* e2 = src->Link(target2);
00230     e2->AdjustEdge(1);
00231     e1->AdjustEdge((unsigned long)(ull << 31));
00232     Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00233     Assert::IsNotNull(res);
00234     Assert::IsTrue(res == target1);
00235     delete src;
00236     delete target1;
00237     delete e1;
00238     delete e2;
00239 }
00240
00241 TEST_METHOD(update_edges_count) {
00242
00243     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00244     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00245     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00246     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00247     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00248     e1->AdjustEdge(25);
00249     src->UpdateEdges(e1);
00250     e2->AdjustEdge(30);
00251     src->UpdateEdges(e2);
00252
00253     Assert::AreEqual((size_t)2, src->Edges()->size());
00254
00255     delete src;
00256     delete target1;
00257     delete e1;
00258     delete e2;
00259 }
00260
00261 TEST_METHOD(update_edges_total) {
00262
00263     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00264     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00265     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00266     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target1);
00267     e1->AdjustEdge(25);
00268     src->UpdateEdges(e1);
00269     e2->AdjustEdge(30);
00270     src->UpdateEdges(e2);
00271
00272 //Assert::AreEqual(55ull, src->TotalEdgeWeights());
00273
00274     delete src;
00275     delete target1;
00276     delete e1;
00277     delete e2;
00278 }
00279
00280 TEST_METHOD(find_vertice) {
00281
00282     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00283     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00284     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00285     Markov::Edge<unsigned char>* res = NULL;
00286     src->Link(target1);
00287     src->Link(target2);
00288
00289     res = src->FindEdge('b');
00290     Assert::IsNotNull(res);
00291     Assert::AreEqual((unsigned char)'b', res->TraverseNode()->nodeValue());
00292     res = src->FindEdge('c');
00293     Assert::IsNotNull(res);
00294     Assert::AreEqual((unsigned char)'c', res->TraverseNode()->nodeValue());
00295
00296     delete src;
00297     delete target1;
00298     delete target2;
00299 }
00300
00301 TEST_METHOD(find_vertice_without_any) {
00302
00303     auto _invalid_next = [] {
00304         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00305         Markov::Edge<unsigned char>* res = NULL;
00306     };
00307 }
```

```

00320
00321             res = src->FindEdge('b');
00322             Assert::IsNull(res);
00323
00324             delete src;
00325         };
00326
00327         //Assert::ExpectException<std::logic_error>(_invalid_next);
00328     }
00329
00330     TEST_METHOD(find_vertex_nonexistent) {
00331
00332         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00333         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00334         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00335         Markov::Edge<unsigned char>* res = NULL;
00336         src->Link(target1);
00337         src->Link(target2);
00338
00339         res = src->FindEdge('D');
00340         Assert::IsNull(res);
00341
00342         delete src;
00343         delete target1;
00344         delete target2;
00345     }
00346 }
```

References [TEST_CLASS\(\)](#).

Here is the call graph for this function:



8.28 Testing::MVP::MarkovPasswords Namespace Reference

[Testing](#) namespace for [MVP MarkovPasswords](#).

Functions

- [TEST_CLASS](#) (ArgParser)

Test Class for Argparse class.

8.28.1 Detailed Description

[Testing](#) namespace for [MVP MarkovPasswords](#).

8.28.2 Function Documentation

8.28.2.1 TEST_CLASS()

```
Testing::MVP::MarkovPasswords::TEST_CLASS (
    ArgParser )
```

Test Class for Argparse class.

test basic generate
test basic generate reordered params
test basic generate param longnames

test basic generate

test basic train

test basic generate

Definition at line 395 of file [UnitTests.cpp](#).

```

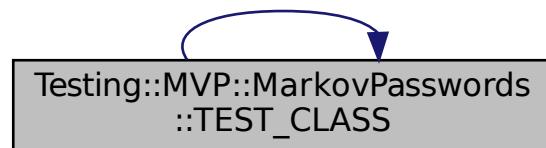
00396         {
00397             public:
00400                 TEST_METHOD(generate_basic) {
00401                     int argc = 8;
00402                     char *argv[] = {"markov.exe", "generate", "-if", "model.mdl", "-of",
00403                     "passwords.txt", "-n", "100"};
00404                     /*ProgramOptions *p = Argparse::parse(argc, argv);
00405                     Assert::IsNotNull(p);
00406
00407                     Assert::AreEqual(p->bImport, true);
00408                     Assert::AreEqual(p->bExport, false);
00409                     Assert::AreEqual(p->importname, "model.mdl");
00410                     Assert::AreEqual(p->outputfilename, "passwords.txt");
00411                     Assert::AreEqual(p->generateN, 100); */
00412
00413     }
00414
00417     TEST_METHOD(generate_basic_reorder) {
00418         int argc = 8;
00419         char *argv[] = { "markov.exe", "generate", "-n", "100", "-if", "model.mdl", "-of",
00420         "passwords.txt" };
00421         /*ProgramOptions* p = Argparse::parse(argc, argv);
00422         Assert::IsNotNull(p);
00423
00424         Assert::AreEqual(p->bImport, true);
00425         Assert::AreEqual(p->bExport, false);
00426         Assert::AreEqual(p->importname, "model.mdl");
00427         Assert::AreEqual(p->outputfilename, "passwords.txt");
00428         Assert::AreEqual(p->generateN, 100);*/
00429     }
00430
00433     TEST_METHOD(generate_basic_longname) {
00434         int argc = 8;
00435         char *argv[] = { "markov.exe", "generate", "-n", "100", "--inputfilename",
00436         "model.mdl", "--outputfilename", "passwords.txt" };
00437         /*ProgramOptions* p = Argparse::parse(argc, argv);
00438         Assert::IsNotNull(p);
00439
00440         Assert::AreEqual(p->bImport, true);
00441         Assert::AreEqual(p->bExport, false);
00442         Assert::AreEqual(p->importname, "model.mdl");
00443         Assert::AreEqual(p->outputfilename, "passwords.txt");
00444         Assert::AreEqual(p->generateN, 100); */
00445     }
00446
00449     TEST_METHOD(generate_fail_badmethod) {
00450         int argc = 8;
00451         char *argv[] = { "markov.exe", "junk", "-n", "100", "--inputfilename",
00452         "model.mdl", "--outputfilename", "passwords.txt" };
00453         /*ProgramOptions* p = Argparse::parse(argc, argv);
00454         Assert::IsNull(p); */
00455     }
00456
00459     TEST_METHOD(train_basic) {
00460         int argc = 4;
00461         char *argv[] = { "markov.exe", "train", "-ef", "model.mdl" };
00462
00463         /*ProgramOptions* p = Argparse::parse(argc, argv);
00464         Assert::IsNotNull(p);
00465
00466         Assert::AreEqual(p->bImport, false);
00467         Assert::AreEqual(p->bExport, true);
00468         Assert::AreEqual(p->exportname, "model.mdl"); */
00469     }
00470
00474     TEST_METHOD(train_basic_longname) {
00475         int argc = 4;
00476         char *argv[] = { "markov.exe", "train", "--exportfilename", "model.mdl" };
00477
00478         /*ProgramOptions* p = Argparse::parse(argc, argv);
00479         Assert::IsNotNull(p);
00480
00481         Assert::AreEqual(p->bImport, false);
00482         Assert::AreEqual(p->bExport, true);
00483         Assert::AreEqual(p->exportname, "model.mdl"); */
00484     }
00485

```

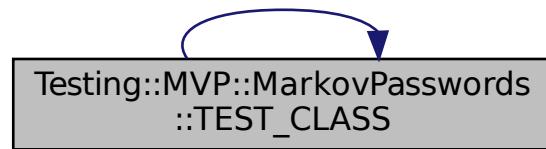
```
00486  
00487  
00488      };  
References TEST_CLASS().
```

Referenced by TEST_CLASS().

Here is the call graph for this function:



Here is the caller graph for this function:



CHAPTER9

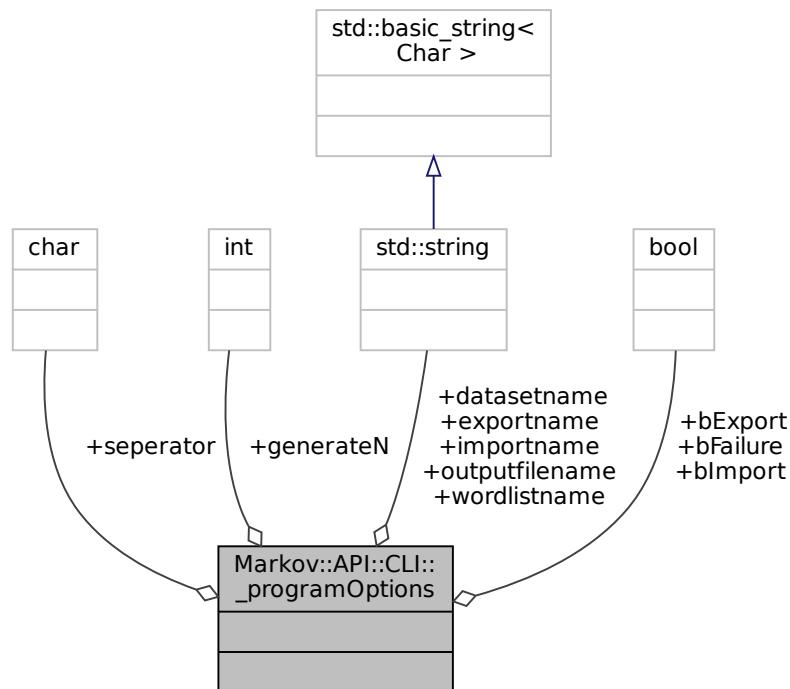
Class Documentation

9.1 Markov::API::CLI::_programOptions Struct Reference

Structure to hold parsed cli arguments.

```
#include <argparse.h>
```

Collaboration diagram for Markov::API::CLI::_programOptions:



Public Attributes

- bool **bImport**
Import flag to validate import
- bool **bExport**
Export flag to validate export
- bool **bFailure**
Failure flag to validate succesfull running
- char **seperator**
Seperator character to use with training data. (character between occurence and value)"
- std::string **importname**

- std::string [exportname](#)
Import name of our model.
- std::string [wordlistname](#)
Import name of our given wordlist
- std::string [outputfilename](#)
Output name of our generated password list
- std::string [datasetname](#)
The name of the given dataset
- int [generateN](#)
Number of passwords to be generated

9.1.1 Detailed Description

Structure to hold parsed cli arguments.

Definition at line [26](#) of file [argparse.h](#).

9.1.2 Member Data Documentation

9.1.2.1 bExport

```
bool Markov::API::CLI::_programOptions::bExport
```

Export flag to validate export

Definition at line [35](#) of file [argparse.h](#).

Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

9.1.2.2 bFailure

```
bool Markov::API::CLI::_programOptions::bFailure
```

Failure flag to validate succesfull running

Definition at line [40](#) of file [argparse.h](#).

Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

9.1.2.3 bImport

```
bool Markov::API::CLI::_programOptions::bImport
```

Import flag to validate import

Definition at line [30](#) of file [argparse.h](#).

Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

9.1.2.4 datasetname

```
std::string Markov::API::CLI::_programOptions::datasetname
```

The name of the given dataset

Definition at line 70 of file [argparse.h](#).

Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

9.1.2.5 exportname

```
std::string Markov::API::CLI::_programOptions::exportname
```

Import name of our given wordlist

Definition at line 55 of file [argparse.h](#).

Referenced by [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

9.1.2.6 generateN

```
int Markov::API::CLI::_programOptions::generateN
```

Number of passwords to be generated

Definition at line 75 of file [argparse.h](#).

Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

9.1.2.7 importname

```
std::string Markov::API::CLI::_programOptions::importname
```

Import name of our model.

Definition at line 50 of file [argparse.h](#).

Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

9.1.2.8 outputfilename

```
std::string Markov::API::CLI::_programOptions::outputfilename
```

Output name of our generated password list

Definition at line 65 of file [argparse.h](#).

Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#), and [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

9.1.2.9 seperator

```
char Markov::API::CLI::_programOptions::seperator
```

Separator character to use with training data. (character between occurrence and value)"

Definition at line 45 of file [argparse.h](#).

Referenced by [Markov::API::CLI::Argparse::setProgramOptions\(\)](#).

9.1.2.10 wordlistname

```
std::string Markov::API::CLI::_programOptions::wordlistname
```

Import name of our given wordlist

Definition at line 60 of file [argparse.h](#).

Referenced by [Markov::API::CLI::Argparse::Argparse\(\)](#).

The documentation for this struct was generated from the following file:

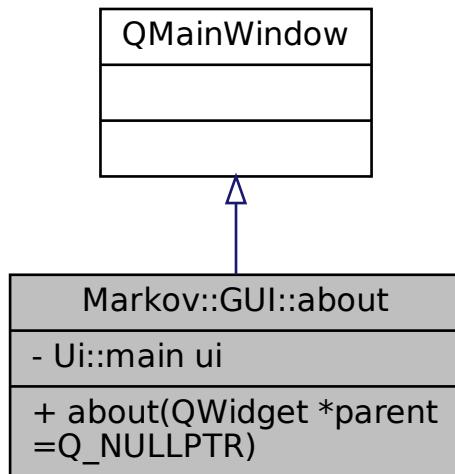
- [Markopy/MarkovAPICLI/src/argparse.h](#)

9.2 Markov::GUI::about Class Reference

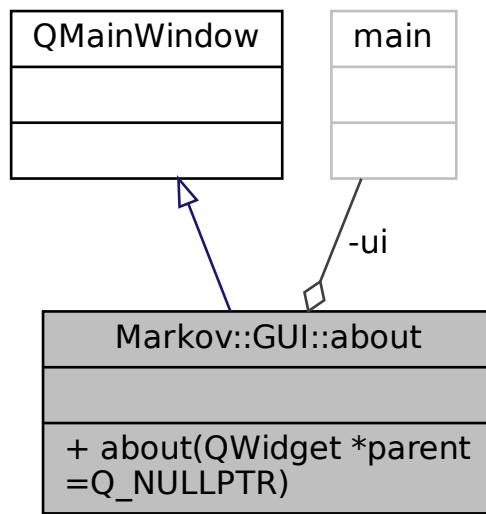
QT Class for about page.

```
#include <about.h>
```

Inheritance diagram for Markov::GUI::about:



Collaboration diagram for Markov::GUI::about:



Public Member Functions

- `about (QWidget *parent=Q_NULLPTR)`

Private Attributes

- `Ui::main ui`

9.2.1 Detailed Description

QT Class for about page.

Definition at line 18 of file [about.h](#).

9.2.2 Constructor & Destructor Documentation

9.2.2.1 `about()`

```
about::about (
    QWidget * parent = Q_NULLPTR )
```

Definition at line 14 of file [about.cpp](#).

```
00015     : QMainWindow(parent)
00016 {
00017     ui.setupUi(this);
00018 }
00019 }
```

References [ui](#).

9.2.3 Member Data Documentation

9.2.3.1 `ui`

```
Ui:: main Markov::GUI::about::ui [private]
```

Definition at line 24 of file [about.h](#).

Referenced by [about\(\)](#).

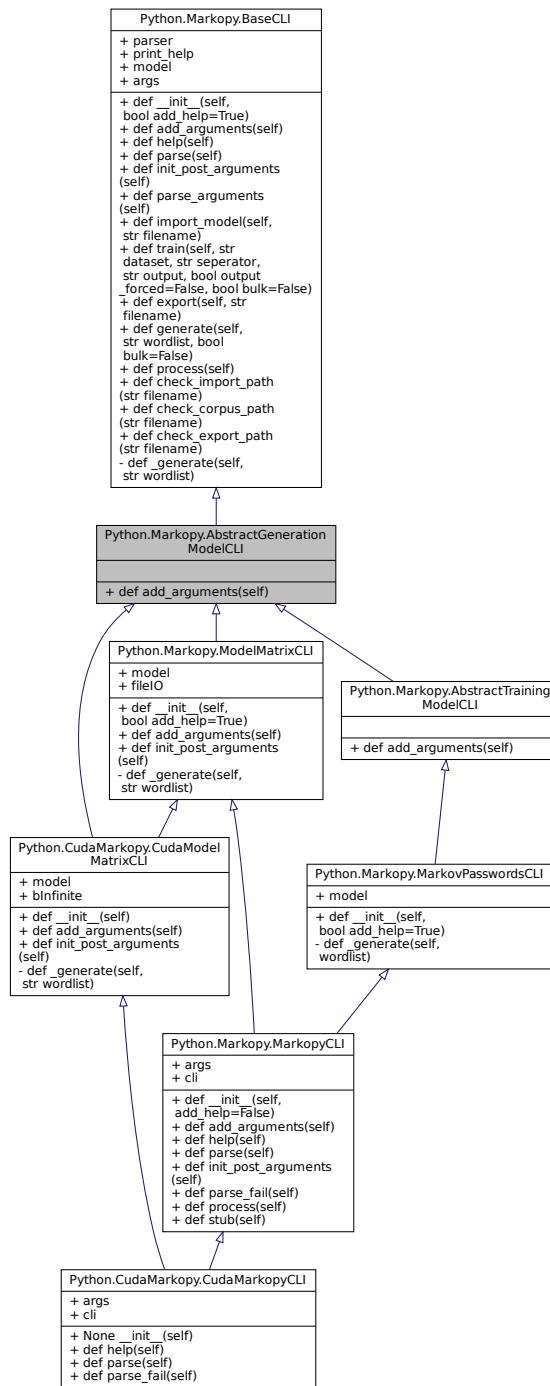
The documentation for this class was generated from the following files:

- [Markopy/MarkovPasswordsGUI/src/about.h](#)
- [Markopy/MarkovPasswordsGUI/src/about.cpp](#)

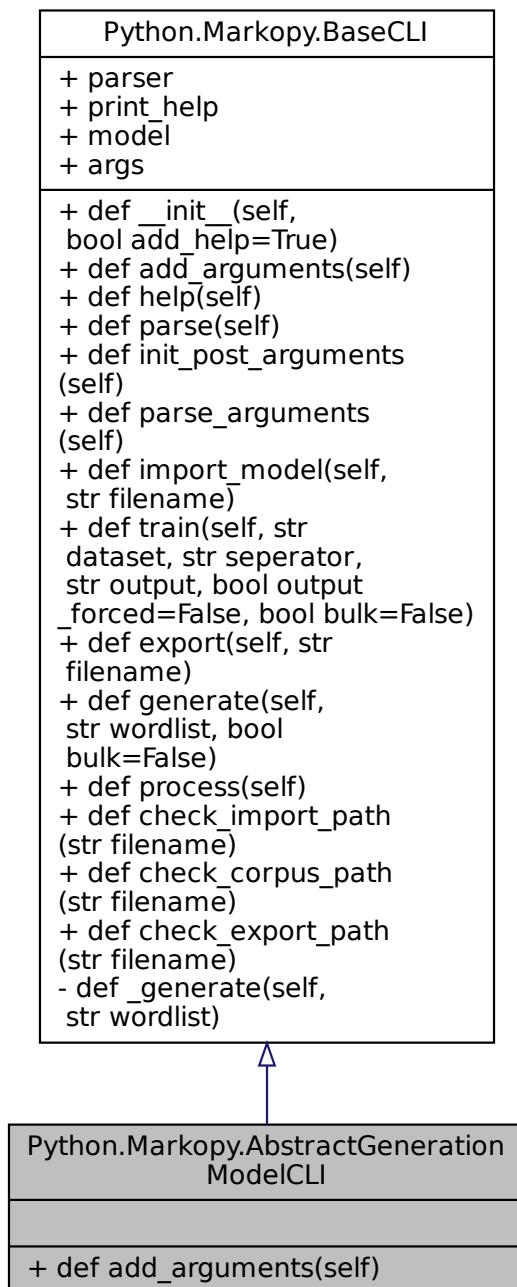
9.3 Python.Markopy.AbstractGenerationModelCLI Class Reference

abstract class for generation capable models

Inheritance diagram for Python.Markopy.AbstractGenerationModelCLI:



Collaboration diagram for Python.Markopy.AbstractGenerationModelCLI:



Public Member Functions

- def [add_arguments](#) (self)
- def [help](#) (self)
- def [parse](#) (self)
- def [init_post_arguments](#) (self)
- def [parse_arguments](#) (self)
- def [import_model](#) (self, str filename)

- def **train** (self, str dataset, str separator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def **export** (self, str filename)

Export model to a file.
- def **generate** (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def **process** (self)

Process parameters for operation.

Static Public Member Functions

- def **check_import_path** (str filename)

check import path for validity
- def **check_corpus_path** (str filename)

check import path for validity
- def **check_export_path** (str filename)

check import path for validity

Public Attributes

- **parser**
- **print_help**
- **model**
- **args**

Private Member Functions

- def **_generate** (self, str wordlist)

wrapper for generate function.

9.3.1 Detailed Description

abstract class for generation capable models
 Definition at line 257 of file [base.py](#).

9.3.2 Member Function Documentation

9.3.2.1 **_generate()**

```
def Python.Markopy.BaseCLI._generate (
    self,
    str wordlist ) [private], [inherited]
```

wrapper for generate function.

This can be overloaded by other models

Parameters

wordlist	filename to generate to
-----------------	-------------------------

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

Definition at line 161 of file [base.py](#).

```
00161     def _generate(self, wordlist : str):
00162         """!
```

```

00163     @brief wrapper for generate function. This can be overloaded by other models
00164     @param wordlist filename to generate to
00165     """
00166     self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
00167     int(self.args.threads))

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.3.2.2 add_arguments()

```
def Python.Markopy.AbstractGenerationModelCLI.add_arguments (
    self )
```

Reimplemented from [Python.Markopy.BaseCLI](#).

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), [Python.Markopy.MarkopyCLI](#), [Python.Markopy.AbstractTrainingModelCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

Definition at line 264 of file [base.py](#).

```

00264     def add_arguments(self):
00265         "Add command line arguments to the parser"
00266         super().add_arguments()
00267         self.parser.add_argument("input",
00268             model will be imported before starting operation.)
00269             self.parser.add_argument("-w", "--wordlist",
00270             export generation results to. Will be ignored for training mode")
00271             self.parser.add_argument("--min", default=6,
00272             allowed during generation)
00273             self.parser.add_argument("--max", default=12,
00274             allowed during generation)
00275             self.parser.add_argument("-n", "--count",
00276             generate. Ignored in training mode.)"
00277
00278

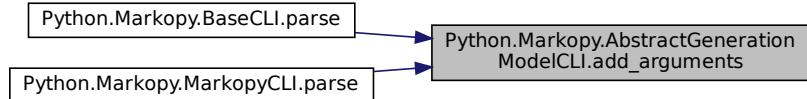
```

help="Input model file. This
help="Wordlist file path to
help="Minimum length that is
help="Maximum length that is
help="Number of lines to

References [Python.Markopy.BaseCLI.parser](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.3.2.3 check_corpus_path()

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.3.2.4 check_export_path()

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
```

```

00200         return True
00201

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.3.2.5 check_import_path()

```

def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

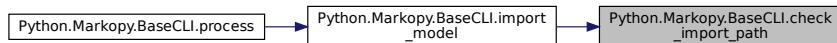
```

00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179

```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.3.2.6 export()

```

def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
Export model to a file.

```

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

Definition at line 138 of file [base.py](#).

```

00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """

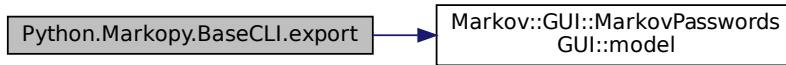
```

```
00143     self.model.Export(filename)
00144
```

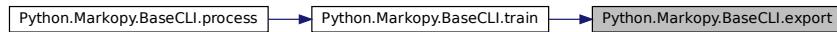
References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.3.2.7 generate()

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

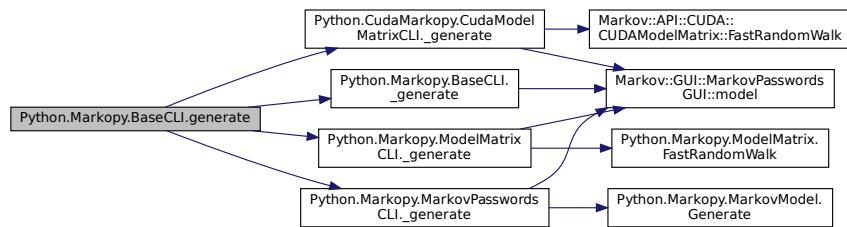
Definition at line 145 of file [base.py](#).

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159
```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI._generate\(\)](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.3.2.8 help()

```
def Python.Markopy.BaseCLI.help (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 51 of file [base.py](#).

```
00051     def help(self):
00052         """! @brief Handle help strings. Defaults to argparse's help"""
00053         self.print_help()
00054 
```

References [Python.Markopy.BaseCLI.print_help](#).

Referenced by [Python.Markopy.MarkopyCLI.add_arguments\(\)](#).

Here is the caller graph for this function:



9.3.2.9 import_model()

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

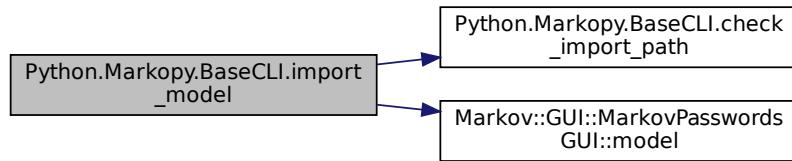
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
00086             directory")
00087             return False
00088
00089         self.model.Import(filename)
00090         logging pprint("Model imported successfully.", 2)
00091
00092         return True
00093
```

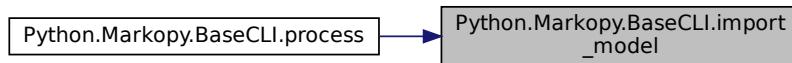
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model\(\)](#), [Python.Markopy.BaseCLI.model\(\)](#), [Python.Markopy.ModelMatrixCLI.model\(\)](#), [Python.Markopy.MarkovPasswordsCLI.model\(\)](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.3.2.10 init_post_arguments()

```
def Python.Markopy.BaseCLI.init_post_arguments (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), [Python.MarkopyMarkopyCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

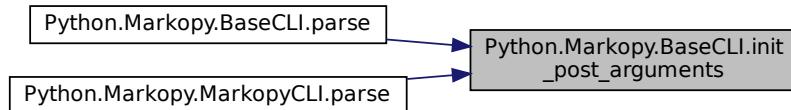
Definition at line 62 of file [base.py](#).

```
00062     def init_post_arguments(self):
00063         """
00064             ! @brief set up stuff that is collected from command line arguments"
00065             logging.VERBOSITY = 0
00066             try:
00067                 if self.args.verbosity:
00068                     logging.VERBOSITY = self.args.verbosity
00069             logging pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00070             except:
00071                 pass
00071
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), and [Python.MarkopyMarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.MarkopyMarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.3.2.11 parse()

```
def Python.Markopy.BaseCLI.parse (
    self ) [inherited]
```

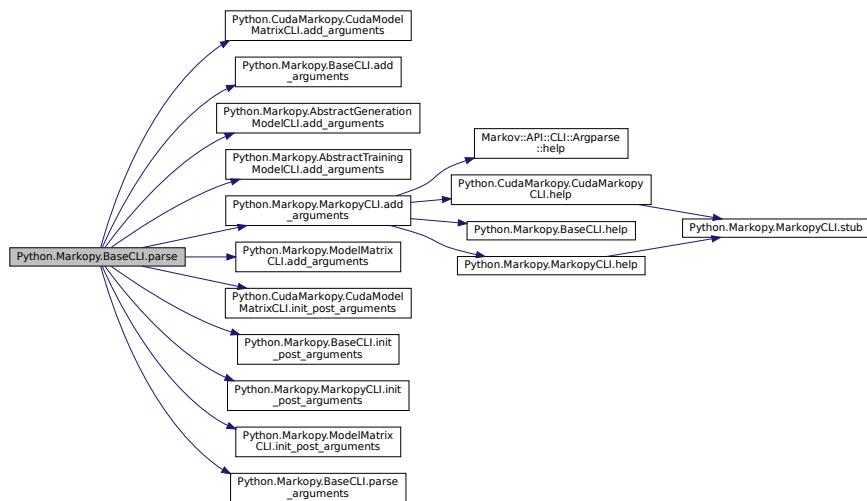
Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 55 of file [base.py](#).

```
00055     def parse(self):
00056         """@brief add, parse and hook arguments"""
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060 
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaModelCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.init_post_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.init_post_arguments\(\)](#) and [Python.Markopy.BaseCLI.parse_arguments\(\)](#).

Here is the call graph for this function:



9.3.2.12 parse_arguments()

```
def Python.Markopy.BaseCLI.parse_arguments (
    self ) [inherited]
```

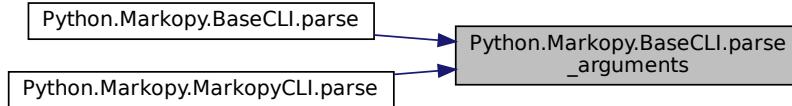
Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """@brief trigger parser"
```

```
00075     self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.3.2.13 process()

```
def Python.Markopy.BaseCLI.process (
    self ) [inherited]
```

Process parameters for operation.

Reimplemented in [Python.Markopy.MarkopyCLI](#).

Definition at line 202 of file [base.py](#).

```
00202     def process(self):
00203         """
00204             @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging pprint(f"Bulk mode operation chosen.", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
00210                     (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00211                     corpus_list = os.listdir(self.args.dataset)
00212                     for corpus in corpus_list:
00213                         self.import_model(self.args.input)
00214                         logging pprint(f"Training {self.args.input} with {corpus}", 2)
00215                         output_file_name = corpus
00216                         model_extension = ""
00217                         if "." in self.args.input:
00218                             model_extension = self.args.input.split(".")[-1]
00219                             self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
00220                                       f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00221                         else:
00222                             logging pprint("In bulk training, output and dataset should be a directory.")
00223                             exit(1)
00224
00225             elif (self.args.mode.lower() == "generate"):
00226                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
00227                     (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00228                     model_list = os.listdir(self.args.input)
00229                     print(model_list)
00230                     for input in model_list:
00231                         logging pprint(f"Generating from {self.args.input}/{input} to
00232 {self.args.wordlist}/{input}.txt", 2)
00233                         self.import_model(f"{self.args.input}/{input}")
00234                         model_base = input
00235                         if "." in self.args.input:
00236                             model_base = input.split(".")[1]
00237                             self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00238                         else:
00239                             logging pprint("In bulk generation, input and wordlist should be directory.")
00240
00241             else:
00242                 self.import_model(self.args.input)
00243                 if (self.args.mode.lower() == "generate"):
00244                     self.generate(self.args.wordlist)
00245
00246             elif (self.args.mode.lower() == "train"):
00247                 self.train(self.args.dataset, self.args.seperator, self.args.output,
00248                           output_forced=True)
00249
00250             elif(self.args.mode.lower() == "combine"):
00251                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00252                 self.generate(self.args.wordlist)
```

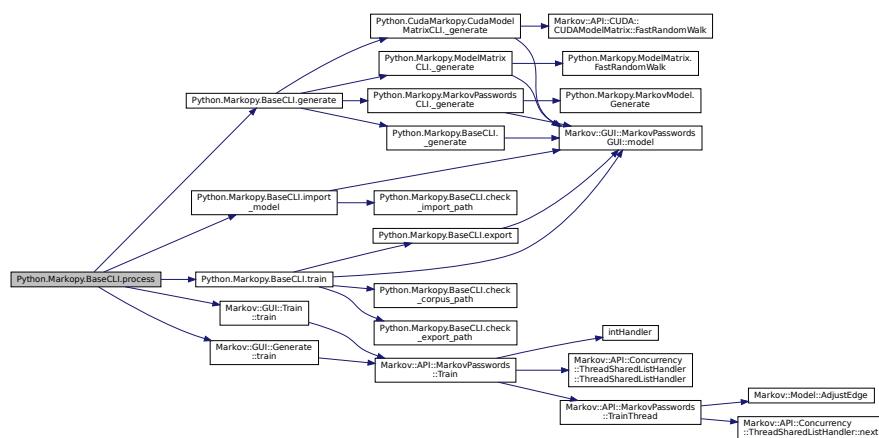
```

00250
00251
00252     else:
00253         logging pprint("Invalid mode arguement given.")
00254         logging pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255         exit(5)
00256

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), [Markov::GUI::Generate.train\(\)](#), [Markov::GUI::Train.train\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



9.3.2.14 train()

```

def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False ) [inherited]

```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

```

00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104         """

```

```

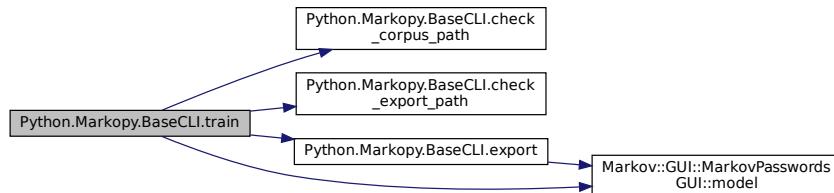
00104     logging pprint("Training.")
00105
00106     if not (dataset and separator and (output or not output_forced)):
00107         logging pprint(f"Training mode requires -d/--dataset(', -o/--output' if output_forced
00108 else") and -s/--separator parameters. Exiting.")
00109         return False
00110
00111     if not bulk and not self.check_corpus_path(dataset):
00112         logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00113         return False
00114
00115     if not self.check_export_path(output):
00116         logging pprint(f"Cannot create output at {output}")
00117         return False
00118
00119     if(separator == '\\\\t'):
00120         logging pprint("Escaping seperator.", 3)
00121         separator = '\\t'
00122
00123     if(len(separator)!=1):
00124         logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00125 accepted.')
00126         exit(4)
00127
00128     logging pprint(f'Starting training.', 3)
00129     self.model.Train(dataset,separator, int(self.args.threads))
00130     logging pprint(f'Training completed.', 2)
00131
00132     if(output):
00133         logging pprint(f'Exporting model to {output}', 2)
00134         self.export(output)
00135     else:
00136         logging pprint(f'Model will not be exported.', 1)
00137
00138     return True
00139

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.3.3 Member Data Documentation

9.3.3.1 args

[Python.Markopy.BaseCLI.args](#) [inherited]

Definition at line 75 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.MarkopyCLI.help\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.parse\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.parse_fail\(\)](#), [Python.Markopy.BaseCLI.process\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.3.3.2 model

[Python.Markopy.BaseCLI.model](#) [inherited]

Definition at line 40 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.3.3.3 parser

[Python.Markopy.BaseCLI.parser](#) [inherited]

Definition at line 25 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.3.3.4 print_help

[Python.Markopy.BaseCLI.print_help](#) [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

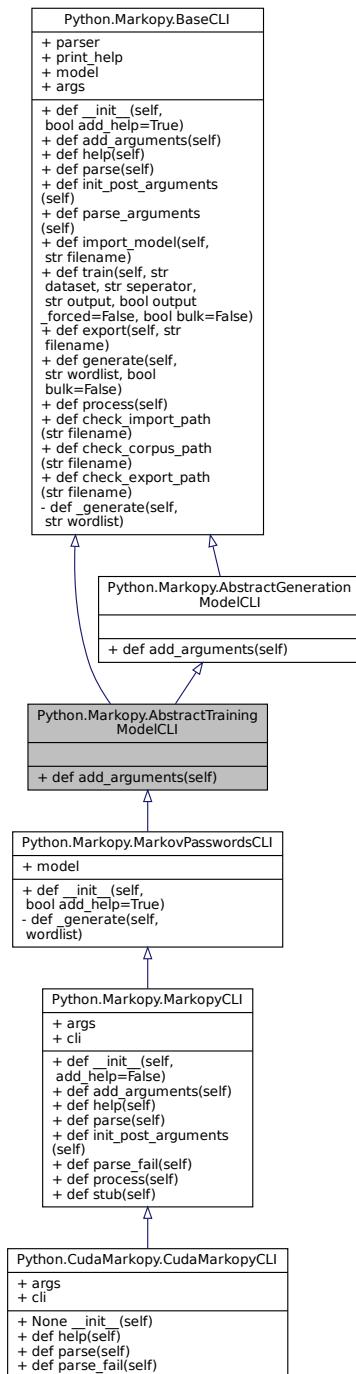
The documentation for this class was generated from the following file:

- [Markopy/Markopy/src/CLI/base.py](#)

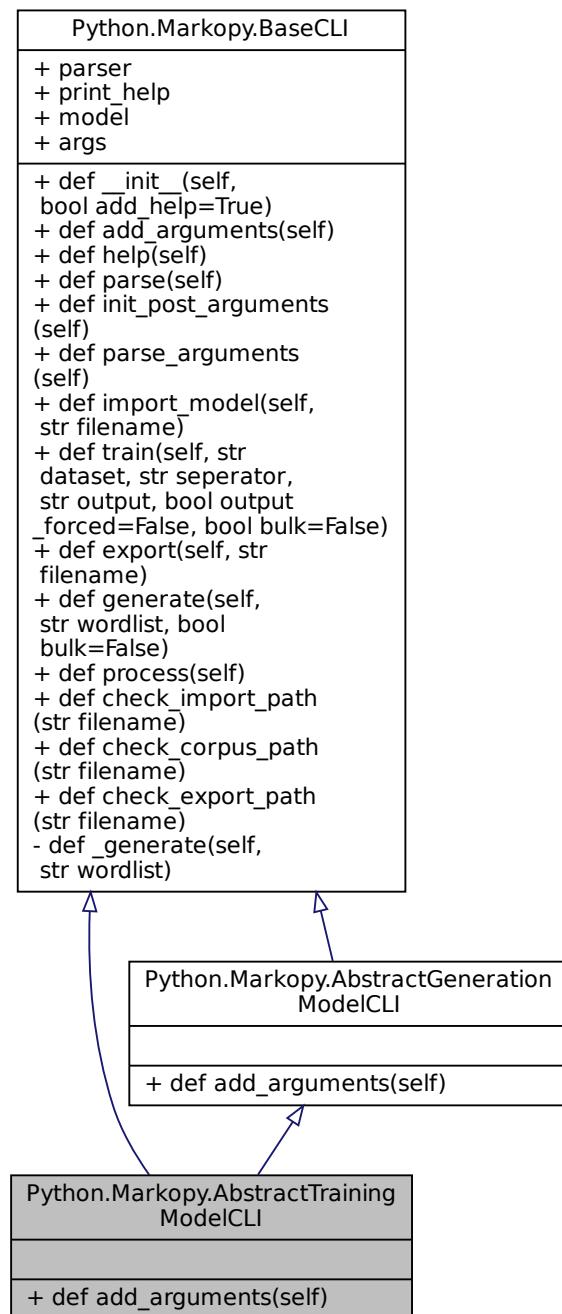
9.4 Python.Markopy.AbstractTrainingModelCLI Class Reference

abstract class for training capable models

Inheritance diagram for Python.Markopy.AbstractTrainingModelCLI:



Collaboration diagram for Python.Markopy.AbstractTrainingModelCLI:



Public Member Functions

- def [add_arguments](#) (self)
- def [help](#) (self)
- def [parse](#) (self)
- def [init_post_arguments](#) (self)
- def [parse_arguments](#) (self)
- def [import_model](#) (self, str filename)

- def **train** (self, str dataset, str separator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def **export** (self, str filename)

Export model to a file.
- def **generate** (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def **process** (self)

Process parameters for operation.
- def **help** (self)
- def **parse** (self)
- def **init_post_arguments** (self)
- def **parse_arguments** (self)
- def **import_model** (self, str filename)

Import a model file.
- def **train** (self, str dataset, str separator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def **export** (self, str filename)

Export model to a file.
- def **generate** (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def **process** (self)

Process parameters for operation.

Static Public Member Functions

- def **check_import_path** (str filename)

check import path for validity
- def **check_corpus_path** (str filename)

check import path for validity
- def **check_export_path** (str filename)

check import path for validity
- def **check_import_path** (str filename)

check import path for validity
- def **check_corpus_path** (str filename)

check import path for validity
- def **check_export_path** (str filename)

check import path for validity

Public Attributes

- **parser**
- **print_help**
- **model**
- **args**
- **parser**
- **print_help**
- **model**
- **args**

Private Member Functions

- def **_generate** (self, str wordlist)

wrapper for generate function.

9.4.1 Detailed Description

abstract class for training capable models
 Definition at line 274 of file [base.py](#).

9.4.2 Member Function Documentation

9.4.2.1 `_generate()`

```
def Python.Markopy.BaseCLI._generate (
    self,
    str wordlist ) [private], [inherited]
```

wrapper for generate function.

This can be overloaded by other models

Parameters

<code>wordlist</code>	filename to generate to
-----------------------	-------------------------

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

Definition at line 161 of file [base.py](#).

```
00161     def _generate(self, wordlist : str):
00162         """
00163             @brief wrapper for generate function. This can be overloaded by other models
00164             @param wordlist filename to generate to
00165         """
00166         self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
00167                             int(self.args.threads))
```

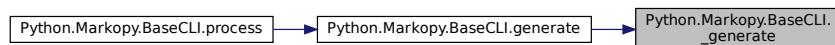
References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.4.2.2 `add_arguments()`

```
def Python.Markopy.AbstractTrainingModelCLI.add_arguments (
    self )
```

Reimplemented from [Python.Markopy.BaseCLI](#).

Reimplemented in [Python.Markopy.MarkopyCLI](#).

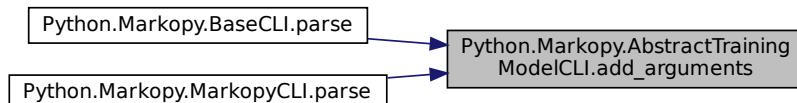
Definition at line 282 of file [base.py](#).

```
00282     def add_arguments(self):
00283         "Add command line arguments to the parser"
00284         self.parser.add_argument("-o", "--output",
00285                                help="Output model file. This
00286                                model will be exported when done. Will be ignored for generation mode.")
00287         self.parser.add_argument("-d", "--dataset",
00288                                help="Dataset file to read input
00289                                from for training. Will be ignored for generation mode.")
00286         self.parser.add_argument("-s", "--separator",
00287                                help="Separator character to use
00288                                with training data.(character between occurrence and value)")
00287         super().add_arguments()
```

References [Python.Markopy.BaseCLI.parser](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.4.2.3 check_corpus_path() [1/2]

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

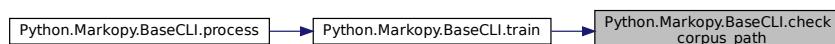
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.4.2.4 check_corpus_path() [2/2]

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

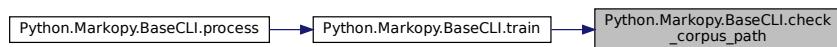
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.4.2.5 check_export_path() [1/2]

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

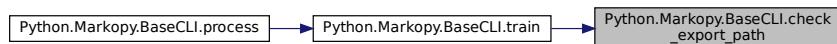
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.4.2.6 check_export_path() [2/2]

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

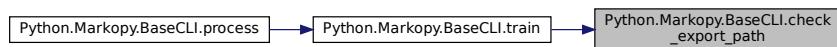
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return False
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:

**9.4.2.7 check_import_path() [1/2]**

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

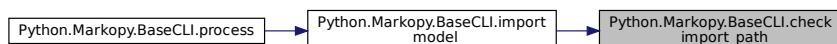
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:

**9.4.2.8 check_import_path() [2/2]**

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

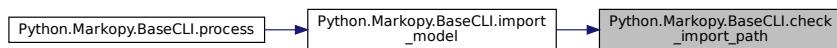
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.4.2.9 export() [1/2]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

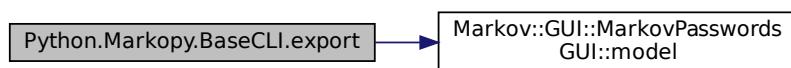
Definition at line 138 of file [base.py](#).

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

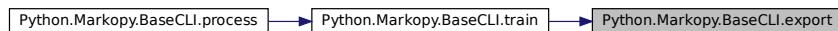
References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.4.2.10 `export()` [2/2]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<code>filename</code>	filename to export to
-----------------------	-----------------------

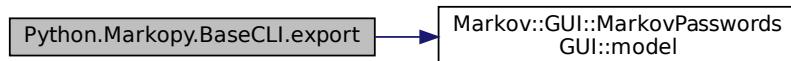
Definition at line 138 of file [base.py](#).

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144 
```

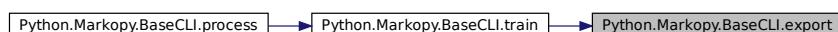
References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.4.2.11 `generate()` [1/2]

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00154         return False
00155
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159
```

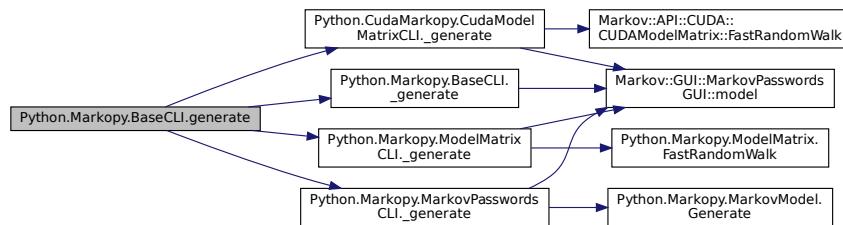
References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#),

[Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopy](#)

[Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.4.2.12 generate() [2/2]

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename

Parameters

<code>bulk</code>	marks bulk operation with directories
-------------------	---------------------------------------

Definition at line 145 of file [base.py](#).

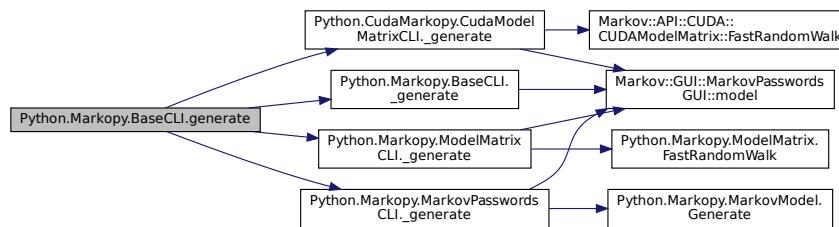
```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159
References Python.CudaMarkopy.CudaModelMatrixCLI.\_generate\(\), Python.Markopy.BaseCLI.\_generate\(\),  

Python.Markopy.ModelMatrixCLI.\_generate\(\), Python.Markopy.MarkovPasswordsCLI.\_generate\(\), Python.CudaMarkopy.CudaMarkopyCLI.\_generate\(\),  

Python.Markopy.BaseCLI.args, and Python.Markopy.MarkopyCLI.args.
```

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.4.2.13 help() [1/2]

```
def Python.Markopy.BaseCLI.help (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 51 of file [base.py](#).

```
00051     def help(self):
00052         """
00053             ! @brief Handle help strings. Defaults to argparse's help"
00054             self.print_help()
```

References [Python.Markopy.BaseCLI.print_help](#).

Referenced by [Python.Markopy.MarkopyCLI.add_arguments\(\)](#).

Here is the caller graph for this function:



9.4.2.14 help() [2/2]

```
def Python.Markopy.BaseCLI.help (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 51 of file [base.py](#).

```
00051     def help(self):
00052         """! @brief Handle help strings. Defaults to argparse's help"""
00053         self.print_help()
00054
```

References [Python.Markopy.BaseCLI.print_help](#).

Referenced by [Python.Markopy.MarkopyCLI.add_arguments\(\)](#).

Here is the caller graph for this function:



9.4.2.15 import_model() [1/2]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

Definition at line 77 of file [base.py](#).

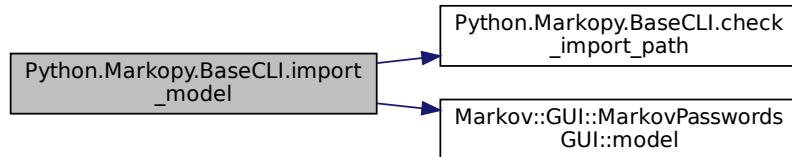
```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#),

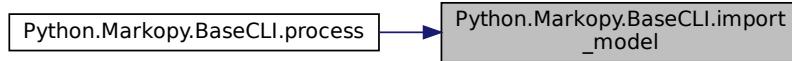
[Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.4.2.16 import_model() [2/2]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<code>filename</code>	filename to import
-----------------------	--------------------

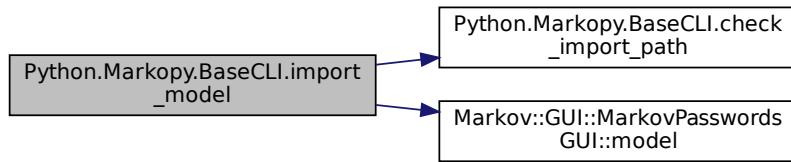
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

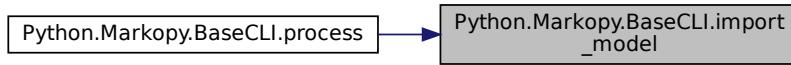
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.4.2.17 init_post_arguments() [1/2]

```
def Python.Markopy.BaseCLI.init_post_arguments (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

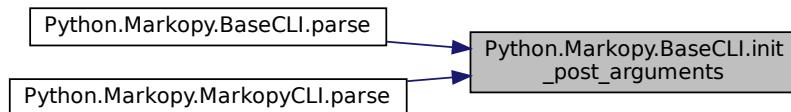
Definition at line 62 of file [base.py](#).

```
00062     def init_post_arguments(self):
00063         """! @brief set up stuff that is collected from command line arguments"""
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.args.verbosity:
00067                 logging.VERBOSITY = self.args.verbosity
00068                 logging pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
00070             pass
00071 
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.4.2.18 init_post_arguments() [2/2]

```
def Python.Markopy.BaseCLI.init_post_arguments (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

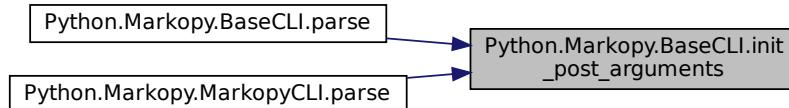
Definition at line 62 of file [base.py](#).

```
00062     def init_post_arguments(self):
00063         """ @brief set up stuff that is collected from command line arguments"""
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.args.verbosity:
00067                 logging.VERBOSITY = self.args.verbosity
00068                 logging pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
00070             pass
00071
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.4.2.19 parse() [1/2]

```
def Python.Markopy.BaseCLI.parse (
    self ) [inherited]
```

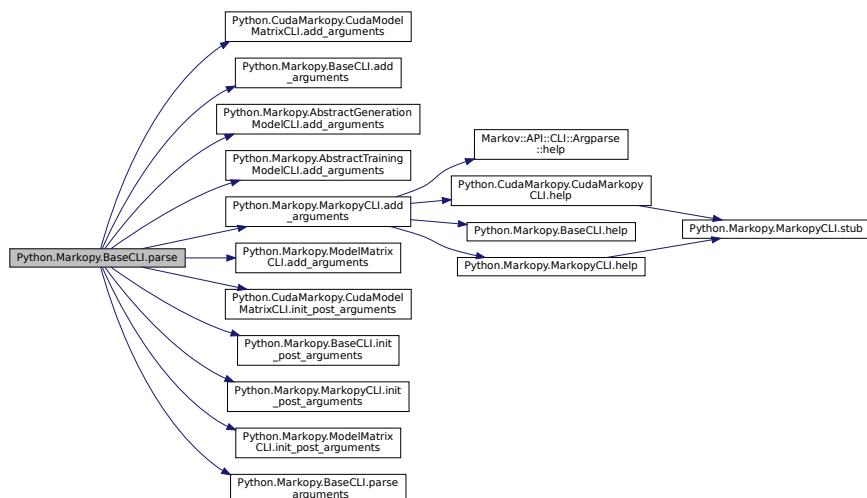
Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 55 of file [base.py](#).

```
00055     def parse(self):
00056         """ @brief add, parse and hook arguments"""
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaModelCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.init_post_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.init_post_arguments\(\)](#), and [Python.Markopy.BaseCLI.parse_arguments\(\)](#).

Here is the call graph for this function:



9.4.2.20 parse() [2/2]

```
def Python.Markopy.BaseCLI.parse (
    self ) [inherited]
```

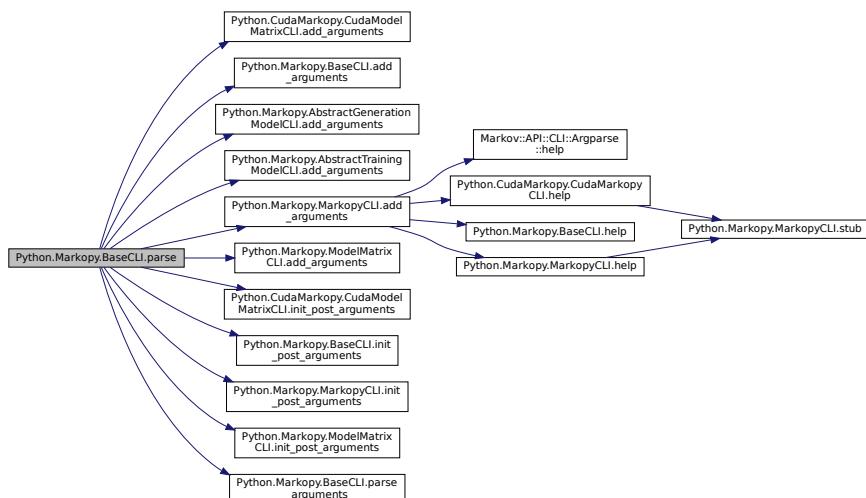
Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 55 of file [base.py](#).

```
00055     def parse(self):
00056         "!" @brief add, parse and hook arguments"
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.init_post_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.init_post_arguments\(\)](#) and [Python.Markopy.BaseCLI.parse_arguments\(\)](#).

Here is the call graph for this function:



9.4.2.21 parse_arguments() [1/2]

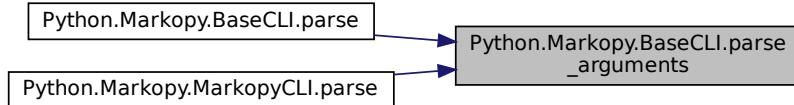
```
def Python.Markopy.BaseCLI.parse_arguments (
    self ) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         "!" @brief trigger parser"
00075         self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.4.2.22 parse_arguments() [2/2]

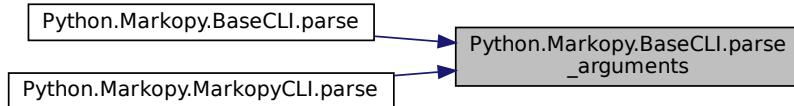
```
def Python.Markopy.BaseCLI.parse_arguments (
    self) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args()[0]
00076 
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.4.2.23 process() [1/2]

```
def Python.Markopy.BaseCLI.process (
    self) [inherited]
```

Process parameters for operation.

Reimplemented in [Python.Markopy.MarkopyCLI](#).

Definition at line 202 of file [base.py](#).

```
00202     def process(self):
00203         """
00204             @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging pprint(f"\"Bulk mode operation chosen.\", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
00210                     (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00211                     corpus_list = os.listdir(self.args.dataset)
00212                     for corpus in corpus_list:
00213                         self.import_model(self.args.input)
00214                         logging pprint(f"\"Training {self.args.input} with {corpus}\", 2)
00215                         output_file_name = corpus
00216                         model_extension = ""
00217                         if "." in self.args.input:
00218                             model_extension = self.args.input.split(".")[-1]
00219                             self.train(f"\"{self.args.dataset}/{corpus}\", self.args.separator,
00220                                     f\"{self.args.output}/{corpus}. {model_extension}\", output_forced=True, bulk=True)
00221                         else:
00222                             logging pprint("In bulk training, output and dataset should be a directory.")
00223                             exit(1)
00224             elif (self.args.mode.lower() == "generate"):
```

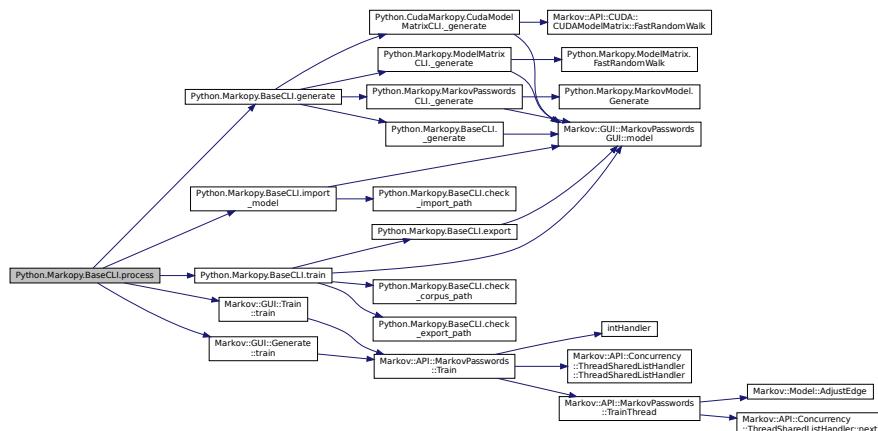
```

00224         if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
00225             (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00226                 model_list = os.listdir(self.args.input)
00227                 for input in model_list:
00228                     logging pprint(f"Generating from {self.args.input}/{input} to
{self.args.wordlist}/{input}.txt", 2)
00229                     self.import_model(f"{self.args.input}/{input}")
00230                     model_base = input
00231                     if "." in self.args.input:
00232                         model_base = input.split(".") [1]
00233                     self.generate(f'{self.args.wordlist}/{model_base}.txt', bulk=True)
00234                 else:
00235                     logging pprint("In bulk generation, input and wordlist should be directory.")
00236
00237             else:
00238                 self.import_model(self.args.input)
00239                 if (self.args.mode.lower() == "generate"):
00240                     self.generate(self.args.wordlist)
00241
00242
00243                 elif (self.args.mode.lower() == "train"):
00244                     self.train(self.args.dataset, self.args.seperator, self.args.output,
output_forced=True)
00245
00246
00247                 elif(self.args.mode.lower() == "combine"):
00248                     self.train(self.args.dataset, self.args.seperator, self.args.output)
00249                     self.generate(self.args.wordlist)
00250
00251
00252             else:
00253                 logging pprint("Invalid mode arguement given.")
00254                 logging pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255                 exit(5)
00256

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), [Markov::GUI::Generate.train\(\)](#), [Markov::GUI::Train.train\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



9.4.2.24 process() [2/2]

```

def Python.Markopy.BaseCLI.process (
    self ) [inherited]

```

Process parameters for operation.

Reimplemented in [Python.Markopy.MarkopyCLI](#).

Definition at line 202 of file [base.py](#).

```

00202     def process(self):
00203         """
00204             @brief Process parameters for operation
00205             """
00206             if(self.args.bulk):

```

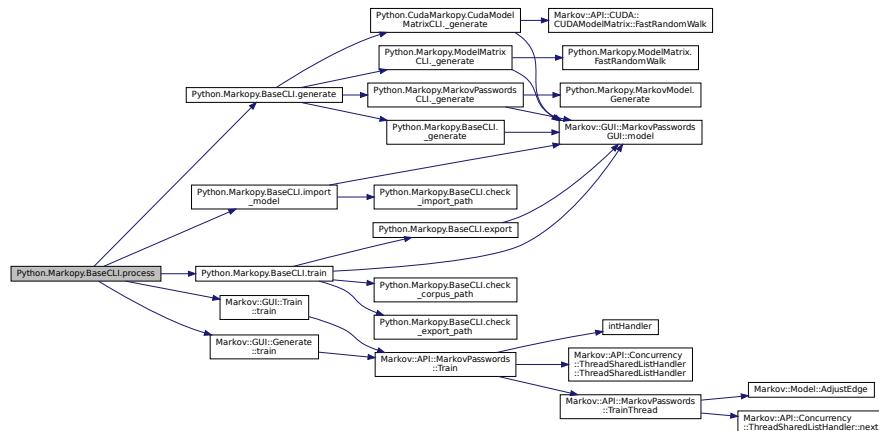
```

00207         logging pprint(f"\"Bulk mode operation chosen.\", 4)
00208         if (self.args.mode.lower() == "train"):
00209             if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
00210                 (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00211                 corpus_list = os.listdir(self.args.dataset)
00212                 for corpus in corpus_list:
00213                     self.import_model(self.args.input)
00214                     logging pprint(f"Training {self.args.input} with {corpus}", 2)
00215                     output_file_name = corpus
00216                     model_extension = ""
00217                     if "." in self.args.input:
00218                         model_extension = self.args.input.split(".")[-1]
00219                         self.train(f"{self.args.dataset}/{corpus}", self.args.separator,
00220                             f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00221             else:
00222                 logging pprint("In bulk training, output and dataset should be a directory.")
00223                 exit(1)
00224
00225             elif (self.args.mode.lower() == "generate"):
00226                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
00227                     (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00228                     model_list = os.listdir(self.args.input)
00229                     print(model_list)
00230                     for input in model_list:
00231                         logging pprint(f"Generating from {self.args.input}/{input} to
00232                             {self.args.wordlist}/{input}.txt", 2)
00233                         self.import_model(f"{self.args.input}/{input}")
00234                         model_base = input
00235                         if "." in self.args.input:
00236                             model_base = input.split(".")[1]
00237                             self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00238             else:
00239                 logging pprint("In bulk generation, input and wordlist should be directory.")
00240
00241
00242             elif (self.args.mode.lower() == "train"):
00243                 self.train(self.args.dataset, self.args.separator, self.args.output,
00244                     output_forced=True)
00245
00246             elif(self.args.mode.lower() == "combine"):
00247                 self.train(self.args.dataset, self.args.separator, self.args.output)
00248                 self.generate(self.args.wordlist)
00249
00250
00251             else:
00252                 logging pprint("Invalid mode arguement given.")
00253                 logging pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00254                 exit(5)
00255
00256

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), [Markov::GUI::Generate.train\(\)](#), [Markov::GUI::Train.train\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



9.4.2.25 train() [1/2]

```
def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False ) [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

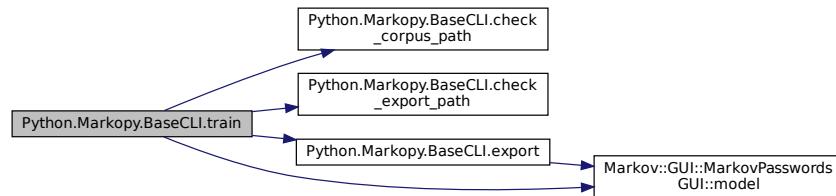
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104         """
00105         logging pprint("Training.")
00106
00107         if not (dataset and seperator and (output or not output_forced)):
00108             logging pprint(f"Training mode requires -d/--dataset', -o/--output' if output_forced
00109             else") and -s/--seperator parameters. Exiting."
00110             return False
00111
00112         if not bulk and not self.check_corpus_path(dataset):
00113             logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00114             return False
00115
00116         if not self.check_export_path(output):
00117             logging pprint(f"Cannot create output at {output}")
00118             return False
00119
00120         if(seperator == '\\t'):
00121             logging pprint("Escaping seperator.", 3)
00122             seperator = '\t'
00123
00124         if(len(seperator)!=1):
00125             logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00126             accepted.')
00127             exit(4)
00128
00129         if(output):
00130             logging pprint(f'Starting training.', 3)
00131             self.model.Train(dataset,seperator, int(self.args.threads))
00132             logging pprint(f'Training completed.', 2)
00133
00134         if(output):
00135             logging pprint(f'Exporting model to {output}', 2)
00136             self.export(output)
00137
00138     else:
00139         logging pprint(f'Model will not be exported.', 1)
00140
00141     return True
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#),

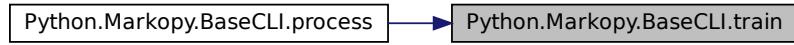
`Python.Markopy.MarkovPasswordsCLI.model`, and `Markov::GUI::MarkovPasswordsGUI.model()`.

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.4.2.26 train() [2/2]

```
def Python.Markopy.BaseCLI.train (
        self,
        str dataset,
        str separator,
        str output,
        bool output_forced = False,
        bool bulk = False ) [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

```
Definition at line 3 of file base.py.
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104         """
00105         logging pprint("Training.")
00106
00107         if not (dataset and seperator and (output or not output_forced)):
```

```

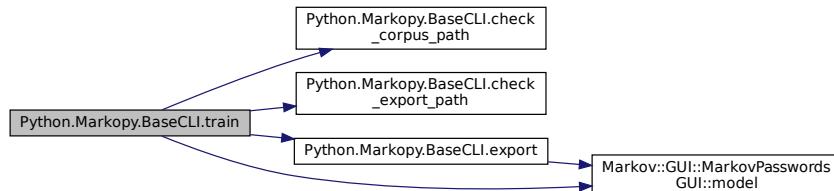
00107         logging pprint(f"Training mode requires -d/--dataset', -o/--output' if output_forced
00108     else") and -s/--seperator parameters. Exiting.")
00109     return False
00110
00110     if not bulk and not self.check_corpus_path(dataset):
00111         logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112         return False
00113
00114     if not self.check_export_path(output):
00115         logging pprint(f"Cannot create output at {output}")
00116         return False
00117
00118     if(seperator == '\\t'):
00119         logging pprint("Escaping seperator.", 3)
00120         seperator = '\t'
00121
00122     if(len(seperator) !=1):
00123         logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00124 accepted.')
00125         exit(4)
00126
00126     logging pprint(f'Starting training.', 3)
00127     self.model.Train(dataset,seperator, int(self.args.threads))
00128     logging pprint(f'Training completed.', 2)
00129
00130     if(output):
00131         logging pprint(f'Exporting model to {output}', 2)
00132         self.export(output)
00133     else:
00134         logging pprint(f'Model will not be exported.', 1)
00135
00136     return True
00137

```

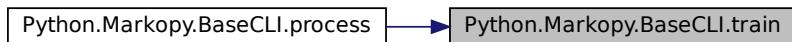
References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.4.3 Member Data Documentation

9.4.3.1 args [1/2]

[Python.Markopy.BaseCLI.args](#) [inherited]

Definition at line 75 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.MarkopyCLI.help\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.parse\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.parse_fail\(\)](#), [Python.Markopy.BaseCLI.process\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.4.3.2 args [2/2]

`Python.Markopy.BaseCLI.args` [inherited]
Definition at line [75](#) of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.MarkopyCLI.help\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.parse\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.parse_fail\(\)](#), [Python.Markopy.BaseCLI.process\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.4.3.3 model [1/2]

`Python.Markopy.BaseCLI.model` [inherited]
Definition at line [40](#) of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.4.3.4 model [2/2]

`Python.Markopy.BaseCLI.model` [inherited]
Definition at line [40](#) of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.4.3.5 parser [1/2]

`Python.Markopy.BaseCLI.parser` [inherited]
Definition at line [25](#) of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.](#)

9.4.3.6 parser [2/2]

`Python.Markopy.BaseCLI.parser` [inherited]
Definition at line [25](#) of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.](#)

9.4.3.7 print_help [1/2]

`Python.Markopy.BaseCLI.print_help` [inherited]
Definition at line [39](#) of file [base.py](#).
Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.4.3.8 print_help [2/2]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

The documentation for this class was generated from the following file:

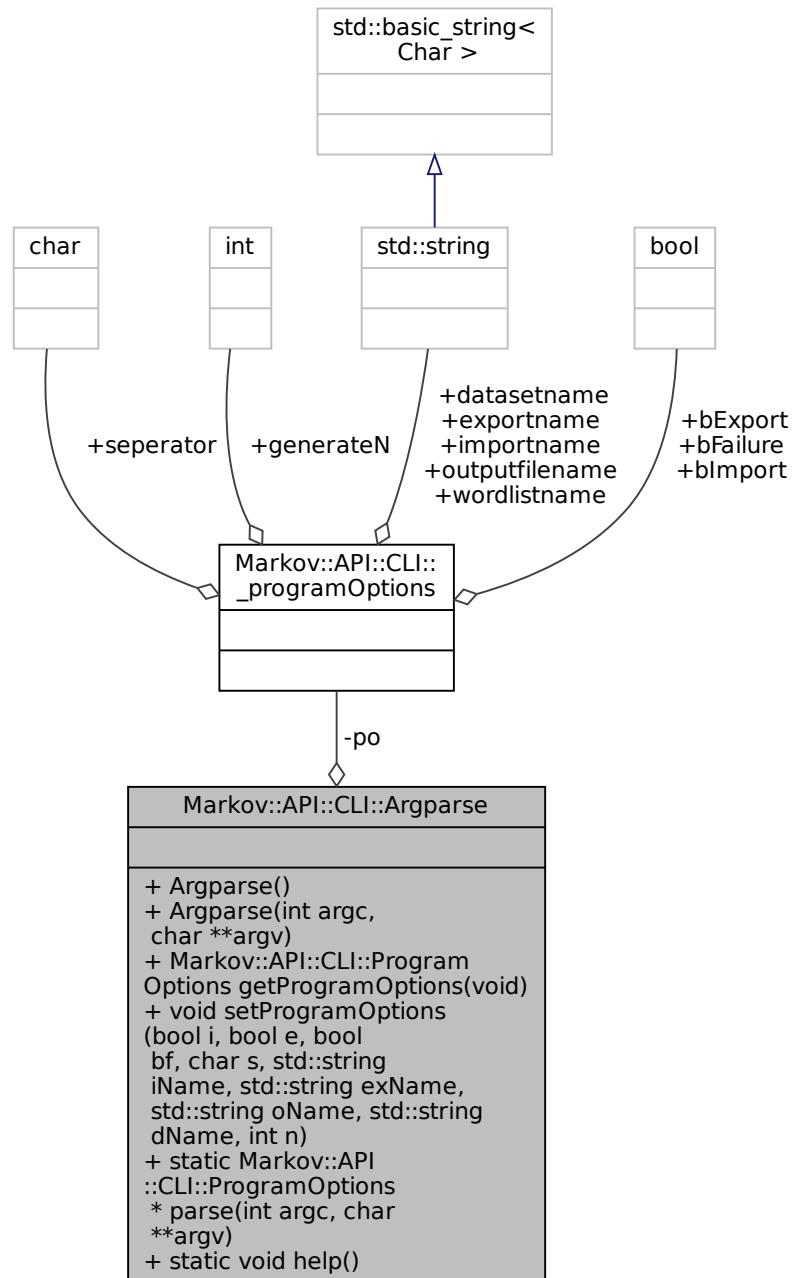
- [Markopy/Markopy/src/CLI/base.py](#)

9.5 Markov::API::CLI::Argparse Class Reference

Parse command line arguments.

```
#include <argparse.h>
```

Collaboration diagram for `Markov::API::CLI::Argparse`:



Public Member Functions

- [Argparse \(\)](#)
- [Argparse \(int argc, char **argv\)](#)

Parse command line arguments.
- [Markov::API::CLI::ProgramOptions getProgramOptions \(void\)](#)

Getter for command line options.
- [void setProgramOptions \(bool i, bool e, bool bf, char s, std::string iName, std::string exName, std::string oName, std::string dName, int n\)](#)

Initialize program options structure.

Static Public Member Functions

- static [Markov::API::CLI::ProgramOptions * parse \(int argc, char **argv\)](#)
parse cli commands and return
- static void [help \(\)](#)
Print help string.

Private Attributes

- [Markov::API::CLI::ProgramOptions po](#)
ProgramOptions structure object.

9.5.1 Detailed Description

Parse command line arguments.

Definition at line 82 of file [argparse.h](#).

9.5.2 Constructor & Destructor Documentation

9.5.2.1 Argparse() [1/2]

[Markov::API::CLI::Argparse::Argparse \(\)](#)

9.5.2.2 Argparse() [2/2]

```
Markov::API::CLI::Argparse::Argparse (
    int argc,
    char ** argv ) [inline]
```

Parse command line arguments.

Parses command line arguments to populate ProgramOptions structure.

Parameters

<i>argc</i>	Number of command line arguments
<i>argv</i>	Array of command line parameters

Definition at line 94 of file [argparse.h](#).

```
00094
00095
00096     /*bool bImp;
00097     bool bExp;
00098     bool bFail;
00099     char sprt;
00100     std::string imports;
00101     std::string exports;
00102     std::string outputs;
00103     std::string datasets;
00104     int generateN;
00105     */
00106     opt::options_description desc("Options");
00107
00108
00109     desc.add_options()
00110         ("generate", "Generate strings with given parameters")
00111         ("train", "Train model with given parameters")
00112         ("combine", "Combine")
00113         ("import", opt::value<std::string>(), "Import model file")
00114         ("output", opt::value<std::string>(), "Output model file. This model will be exported
when done. Will be ignored for generation mode")
```

```

00115         ("dataset", opt::value<std::string>(), "Dataset file to read input from training. Will
00116     be ignored for generation mode")
00117         ("seperator", opt::value<char>(), "Seperator character to use with training data.
00118     (character between occurrence and value)")
00119         ("wordlist", opt::value<std::string>(), "Wordlist file path to export generation
00120     results to. Will be ignored for training mode")
00121         ("count", opt::value<int>(), "Number of lines to generate. Ignored in training mode")
00122         ("verbosity", "Output verbosity")
00123         ("help", "Option definitions");
00124
00125     opt::variables_map vm;
00126
00127     opt::store(opt::parse_command_line(argc, argv, desc), vm);
00128
00129     opt::notify(vm);
00130
00131     //std::cout << desc << std::endl;
00132     if (vm.count("help")) {
00133         std::cout << desc << std::endl;
00134     }
00135
00136     if (vm.count("output") == 0) this->po.outputfilename = "NULL";
00137     else if (vm.count("output") == 1) {
00138         this->po.outputfilename = vm["output"].as<std::string>();
00139         this->po.bExport = true;
00140     }
00141     else {
00142         this->po.bFailure = true;
00143         std::cout << "UNIDENTIFIED INPUT" << std::endl;
00144         std::cout << desc << std::endl;
00145     }
00146
00147     if (vm.count("dataset") == 0) this->po.datasetname = "NULL";
00148     else if (vm.count("dataset") == 1) {
00149         this->po.datasetname = vm["dataset"].as<std::string>();
00150     }
00151     else {
00152         this->po.bFailure = true;
00153         std::cout << "UNIDENTIFIED INPUT" << std::endl;
00154         std::cout << desc << std::endl;
00155     }
00156
00157     if (vm.count("wordlist") == 0) this->po.wordlistname = "NULL";
00158     else if (vm.count("wordlist") == 1) {
00159         this->po.wordlistname = vm["wordlist"].as<std::string>();
00160     }
00161     else {
00162         this->po.bFailure = true;
00163         std::cout << "UNIDENTIFIED INPUT" << std::endl;
00164         std::cout << desc << std::endl;
00165     }
00166
00167     if (vm.count("import") == 0) this->po.importname = "NULL";
00168     else if (vm.count("import") == 1) {
00169         this->po.importname = vm["import"].as<std::string>();
00170         this->po.bImport = true;
00171     }
00172     else {
00173         this->po.bFailure = true;
00174         std::cout << "UNIDENTIFIED INPUT" << std::endl;
00175         std::cout << desc << std::endl;
00176     }
00177
00178     if (vm.count("count") == 0) this->po.generateN = 0;
00179     else if (vm.count("count") == 1) {
00180         this->po.generateN = vm["count"].as<int>();
00181     }
00182     else {
00183         this->po.bFailure = true;
00184         std::cout << "UNIDENTIFIED INPUT" << std::endl;
00185         std::cout << desc << std::endl;
00186     }
00187
00188     /*std::cout << vm["output"].as<std::string>() << std::endl;
00189     std::cout << vm["dataset"].as<std::string>() << std::endl;
00190     std::cout << vm["wordlist"].as<std::string>() << std::endl;
00191     std::cout << vm["output"].as<std::string>() << std::endl;
00192     std::cout << vm["count"].as<int>() << std::endl;*/
00193
00194     //else if (vm.count("train")) std::cout << "train oldu" << std::endl;
00195 }

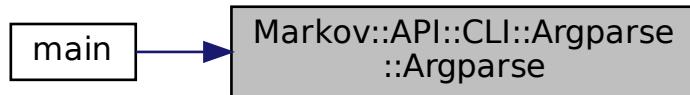
```

References [Markov::API::CLI::_programOptions::bExport](#), [Markov::API::CLI::_programOptions::bFailure](#), [Markov::API::CLI::_program](#)

`Markov::API::CLI::_programOptions::datasetname`, `Markov::API::CLI::_programOptions::generateN`, `Markov::API::CLI::_programOptions::outputfilename`, `po`, and `Markov::API::CLI::_programOptions::wordlistname`.

Referenced by `main()`.

Here is the caller graph for this function:



9.5.3 Member Function Documentation

9.5.3.1 getProgramOptions()

```
Markov::API::CLI::ProgramOptions Markov::API::CLI::Argparse::getProgramOptions (
    void ) [inline]
```

Getter for command line options.

Getter for ProgramOptions populated by the argument parser

Returns

ProgramOptions structure.

Definition at line 203 of file `argparse.h`.

```
00203
00204     return this->po;
00205 }
```

References `po`.

9.5.3.2 help()

```
void Markov::API::CLI::Argparse::help ( ) [static]
```

Print help string.

Definition at line 15 of file `argparse.cpp`.

```
00015
00016     std::cout <<
00017     "Markov Passwords - Help\n"
00018     "Options:\n"
00019     "  \n"
00020     "  -of --outputfilename\n"
00021     "    Filename to output the generation results\n"
00022     "  -ef --exportfilename\n"
00023     "    filename to export built model to\n"
00024     "  -if --importfilename\n"
00025     "    filename to import model from\n"
00026     "  -n (generate count)\n"
00027     "    Number of lines to generate\n"
00028     "  \n"
00029     "Usage: \n"
00030     "  markov.exe -if empty_model.mdl -ef model.mdl\n"
00031     "    import empty_model.mdl and train it with data from stdin. When done, output the model to
model.mdl\n"
00032     "\n"
00033     "  markov.exe -if empty_model.mdl -n 15000 -of wordlist.txt\n"
00034     "    import empty_model.mdl and generate 15000 words to wordlist.txt\n"
00035
00036     << std::endl;
00037 }
```

Referenced by `Python.Markopy.MarkopyCLI::add_arguments()`.

Here is the caller graph for this function:



9.5.3.3 parse()

```
Markov::API::CLI::ProgramOptions * Markov::API::CLI::Argparse::parse (
    int argc,
    char ** argv ) [static]
parse cli commands and return
```

Parameters

<i>argc</i>	- Program argument count
<i>argv</i>	- Program argument values array

Returns

ProgramOptions structure.

Definition at line 11 of file [argparse.cpp](#).

```
00011 { return 0; }
```

9.5.3.4 setProgramOptions()

```
void Markov::API::CLI::Argparse::setProgramOptions (
    bool i,
    bool e,
    bool bf,
    char s,
    std::string iName,
    std::string exName,
    std::string oName,
    std::string dName,
    int n ) [inline]
```

Initialize program options structure.

Parameters

<i>i</i>	boolean, true if import operation is flagged
<i>e</i>	boolean, true if export operation is flagged
<i>bf</i>	boolean, true if there is something wrong with the command line parameters
<i>s</i>	separator character for the import function
<i>iName</i>	import filename
<i>exName</i>	export filename
<i>oName</i>	output filename
<i>dName</i>	corpus filename
<i>n</i>	number of passwords to be generated

Definition at line 220 of file [argparse.h](#).

```

00220
00221     this->po.bImport = i;
00222     this->po.bExport = e;
00223     this->po.seperator = s;
00224     this->po.bFailure = bf;
00225     this->po.generateN = n;
00226     this->po.importname = iName;
00227     this->po.exportname = exName;
00228     this->po.outputfilename = oName;
00229     this->po.datasetname = dName;
00230
00231     /*strcpy_s(this->po.importname,256,iName);
00232      strcpy_s(this->po.exportname,256,exName);
00233      strcpy_s(this->po.outputfilename,256,oName);
00234      strcpy_s(this->po.datasetname,256,dName);*/
00235
00236 }
```

References [Markov::API::CLI::_programOptions::bExport](#), [Markov::API::CLI::_programOptions::bFailure](#), [Markov::API::CLI::_programOptions::datasetname](#), [Markov::API::CLI::_programOptions::exportname](#), [Markov::API::CLI::_programOptions::importname](#), [Markov::API::CLI::_programOptions::outputfilename](#), [po](#), and [Markov::API::CLI::_programOptions::seperator](#).

9.5.4 Member Data Documentation

9.5.4.1 po

`Markov::API::CLI::ProgramOptions` `Markov::API::CLI::Argparse::po` [private]
ProgramOptions structure object.

Definition at line 255 of file [argparse.h](#).

Referenced by [Argparse\(\)](#), [getProgramOptions\(\)](#), and [setProgramOptions\(\)](#).

The documentation for this class was generated from the following files:

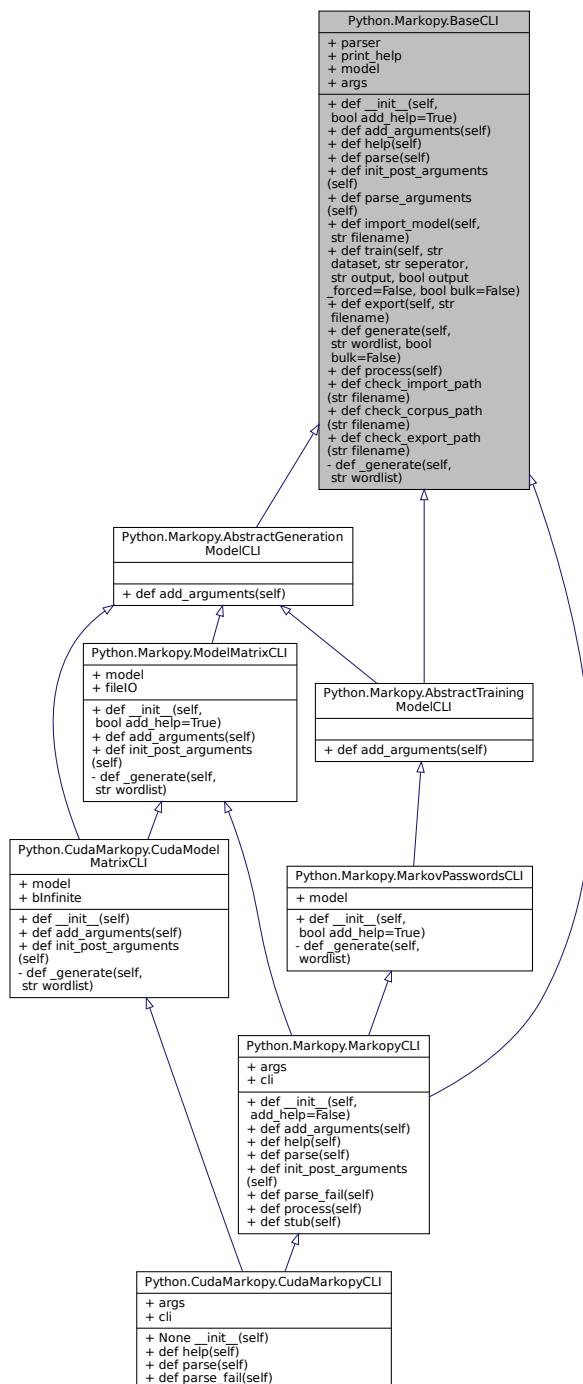
- [Markopy/MarkovAPICLI/src/argparse.h](#)

- [Markopy/MarkovAPICLI/src/argparse.cpp](#)

9.6 Python.Markopy.BaseCLI Class Reference

Base CLI class to handle user interactions

Inheritance diagram for Python.Markopy.BaseCLI:



Collaboration diagram for Python.Markopy.BaseCLI:

Python.Markopy.BaseCLI
+ parser + print_help + model + args + def __init__(self, bool add_help=True) + def add_arguments(self) + def help(self) + def parse(self) + def init_post_arguments (self) + def parse_arguments (self) + def import_model(self, str filename) + def train(self, str dataset, str seperator, str output, bool output _forced=False, bool bulk=False) + def export(self, str filename) + def generate(self, str wordlist, bool bulk=False) + def process(self) + def check_import_path (str filename) + def check_corpus_path (str filename) + def check_export_path (str filename) - def _generate(self, str wordlist)

Public Member Functions

- def `__init__` (self, bool add_help=True)
initialize base CLI
- def `add_arguments` (self)
- def `help` (self)
- def `parse` (self)
- def `init_post_arguments` (self)

- def `parse_arguments` (self)
- def `import_model` (self, str filename)

Import a model file.
- def `train` (self, str dataset, str separator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def `export` (self, str filename)

Export model to a file.
- def `generate` (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def `process` (self)

Process parameters for operation.

Static Public Member Functions

- def `check_import_path` (str filename)

check import path for validity
- def `check_corpus_path` (str filename)

check import path for validity
- def `check_export_path` (str filename)

check import path for validity

Public Attributes

- `parser`
- `print_help`
- `model`
- `args`

Private Member Functions

- def `_generate` (self, str wordlist)

wrapper for generate function.

9.6.1 Detailed Description

Base CLI class to handle user interactions

Definition at line 16 of file [base.py](#).

9.6.2 Constructor & Destructor Documentation

9.6.2.1 `__init__()`

```
def Python.Markopy.BaseCLI.__init__ (
    self,
    bool add_help = True )
initialize base CLI
```

Parameters

<code>add_help</code>	decide to overload the help function or not
-----------------------	---

Reimplemented in [Python.Markopy.MarkovPasswordsCLI](#), and [Python.Markopy.ModelMatrixCLI](#).

Definition at line 20 of file [base.py](#).

```

00020     def __init__(self, add_help : bool=True):
00021         """
00022             @brief initialize base CLI
00023             @param add_help decide to overload the help function or not
00024             """
00025             self.parser = argparse.ArgumentParser(description="Python wrapper for MarkovPasswords.",
00026             epilog=f"""{colored("Sample runs:", "yellow")}\n
00027             {__file__.split("/")[-1]} train untrained.mdl -d dataset.dat -s "\\t" -o trained.mdl
00028                 Import untrained.mdl, train it with dataset.dat which has tab delimited data, output
00029                 resulting model to trained.mdl\n
00030             {__file__.split("/")[-1]} generate trained.mdl -n 500 -w output.txt
00031                 Import trained.mdl, and generate 500 lines to output.txt
00032
00033             {__file__.split("/")[-1]} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt
00034                 Train and immediately generate 500 lines to output.txt. Do not export trained model.
00035
00036             {__file__.split("/")[-1]} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt
00037                 Train and immediately generate 500 lines to output.txt. Export trained model.
00038             """", add_help=add_help, formatter_class=argparse.RawTextHelpFormatter)
00039             self.print_help = self.parser.print_help
00040             self.model = MarkovModel()
00041

```

9.6.3 Member Function Documentation

9.6.3.1 `_generate()`

```
def Python.Markopy.BaseCLI._generate (
    self,
    str wordlist ) [private]
```

wrapper for generate function.

This can be overloaded by other models

Parameters

<code>wordlist</code>	filename to generate to
-----------------------	-------------------------

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

Definition at line 161 of file [base.py](#).

```

00161     def _generate(self, wordlist : str):
00162         """
00163             @brief wrapper for generate function. This can be overloaded by other models
00164             @param wordlist filename to generate to
00165             """
00166             self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
00167             int(self.args.threads))
00167

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.6.3.2 add_arguments()

```
def Python.Markopy.BaseCLI.add_arguments (
    self )
```

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), [Python.MarkopyMarkopyCLI](#), [Python.MarkopyAbstractTrainingModelCLI](#), [Python.MarkopyAbstractGenerationModelCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

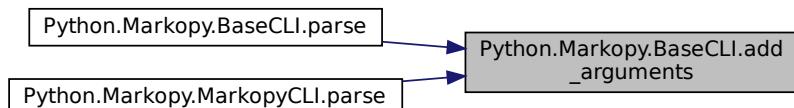
Definition at line 43 of file [base.py](#).

```
00043     def add_arguments(self):
00044         """! @brief Add command line arguments to the parser"""
00045         self.parser.add_argument("mode",
00046             'Train', 'Generate', or 'Combine' .)
00047         self.parser.add_argument("-t", "--threads", default=10,
00048             generate. Ignored in training mode.)
00049         self.parser.add_argument("-v", "--verbosity", action="count",
00050             self.parser.add_argument("-b", "--bulk", action="store_true",
00051             every corpus/model in the folder.)
```

References [Python.Markopy.BaseCLI.parser](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.MarkopyMarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.6.3.3 check_corpus_path()

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static]
check import path for validity
```

Parameters

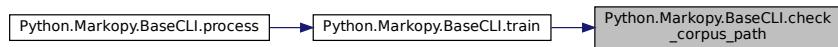
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.6.3.4 check_export_path()

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static]
check import path for validity
```

Parameters

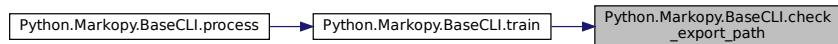
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.6.3.5 check_import_path()

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static]
check import path for validity
```

Parameters

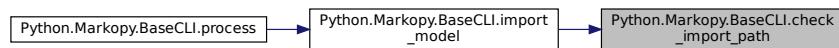
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.6.3.6 export()

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename )
```

Export model to a file.

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

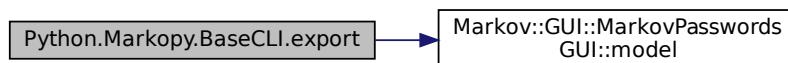
Definition at line 138 of file [base.py](#).

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

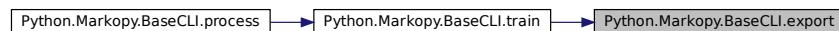
References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.6.3.7 generate()

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False )
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155         return False
00156
00157         if(bulk and os.path.isfile(wordlist)):
00158             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00159         self._generate(wordlist)
```

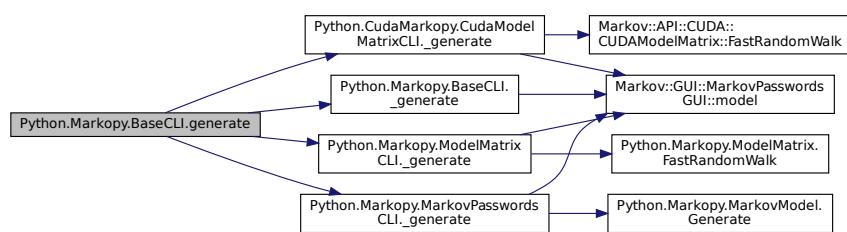
References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#),

[Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI._generate\(\)](#),

[Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.6.3.8 help()

```
def Python.Markopy.BaseCLI.help (
    self )
```

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

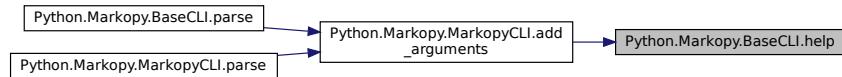
Definition at line 51 of file [base.py](#).

```
00051     def help(self):
00052         """! @brief Handle help strings. Defaults to argparse's help"""
00053         self.print_help()
00054
```

References [Python.Markopy.BaseCLI.print_help](#).

Referenced by [Python.Markopy.MarkopyCLI.add_arguments\(\)](#).

Here is the caller graph for this function:



9.6.3.9 import_model()

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename )
Import a model file.
```

Parameters

<code>filename</code>	filename to import
-----------------------	--------------------

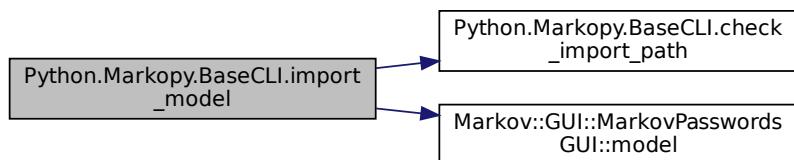
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

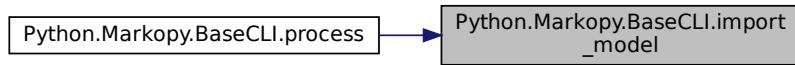
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.6.3.10 init_post_arguments()

```
def Python.Markopy.BaseCLI.init_post_arguments (
    self )
```

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

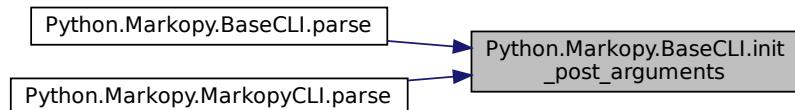
Definition at line 62 of file [base.py](#).

```
00062     def init_post_arguments(self):
00063         """! @brief set up stuff that is collected from command line arguments"""
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.args.verbosity:
00067                 logging.VERBOSITY = self.args.verbosity
00068                 logging pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
00070             pass
00071 
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.6.3.11 parse()

```
def Python.Markopy.BaseCLI.parse (
    self )
```

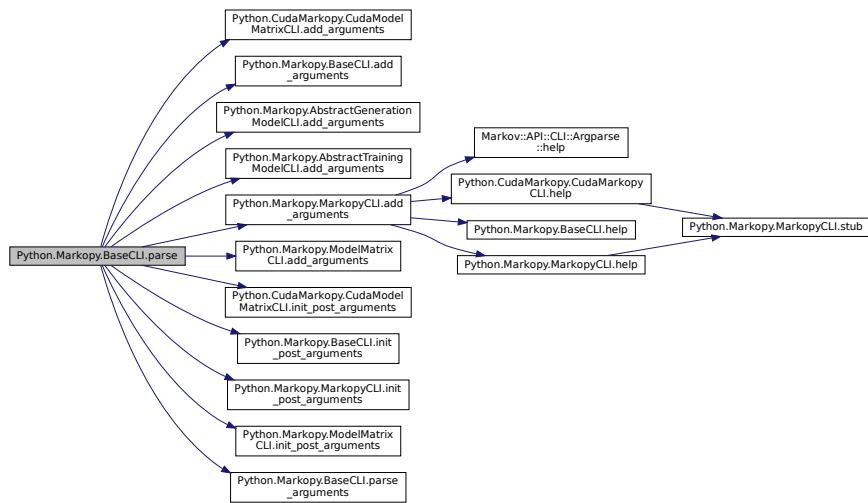
Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 55 of file [base.py](#).

```
00055     def parse(self):
00056         """! @brief add, parse and hook arguments"""
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060 
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.init_post_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.init_post_arguments\(\)](#), and [Python.Markopy.BaseCLI.parse_arguments\(\)](#).

Here is the call graph for this function:



9.6.3.12 parse_arguments()

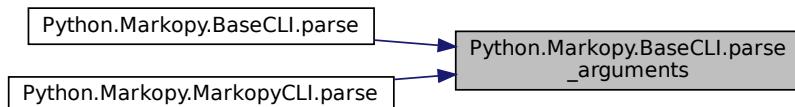
```
def Python.Markopy.BaseCLI.parse_arguments (
    self )
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """!
00075             @brief trigger parser
00076             self.args = self.parser.parse_known_args() [0]
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.6.3.13 process()

```
def Python.Markopy.BaseCLI.process (
    self )
```

Process parameters for operation.

Reimplemented in [Python.Markopy.MarkopyCLI](#).

Definition at line 202 of file [base.py](#).

```
00202     def process(self):
00203         """
00204             @brief Process parameters for operation
00205             """
00206             if(self.args.bulk):
00207                 logging pprint(f"Bulk mode operation chosen.", 4)
00208                 if (self.args.mode.lower() == "train"):
00209                     if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
00210                         (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
```

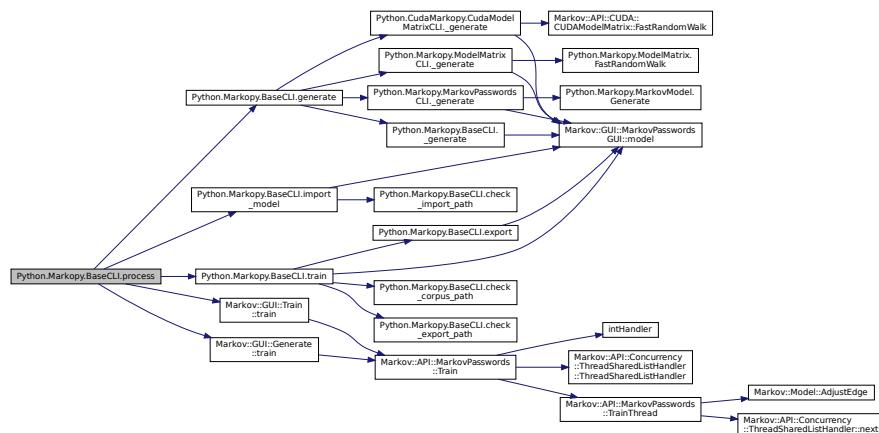
```

00210     corpus_list = os.listdir(self.args.dataset)
00211     for corpus in corpus_list:
00212         self.import_model(self.args.input)
00213         logging pprint(f"Training {self.args.input} with {corpus}", 2)
00214         output_file_name = corpus
00215         model_extension = ""
00216         if "." in self.args.input:
00217             model_extension = self.args.input.split(".")[-1]
00218             self.train(f"{self.args.dataset}/{corpus}", self.args.separator,
00219             f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00220         else:
00221             logging pprint("In bulk training, output and dataset should be a directory.")
00222             exit(1)
00223
00224         elif (self.args.mode.lower() == "generate"):
00225             if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
00226             (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00227                 model_list = os.listdir(self.args.input)
00228                 print(model_list)
00229                 for input in model_list:
00230                     logging pprint(f"Generating from {self.args.input}/{input} to
00231                     {self.args.wordlist}/{input}.txt", 2)
00232                     self.import_model(f"{self.args.input}/{input}")
00233                     model_base = input
00234                     if "." in self.args.input:
00235                         model_base = input.split(".")[1]
00236                         self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00237             else:
00238                 logging pprint("In bulk generation, input and wordlist should be directory.")
00239
00240         else:
00241             self.import_model(self.args.input)
00242             if (self.args.mode.lower() == "generate"):
00243                 self.generate(self.args.wordlist)
00244
00245             elif (self.args.mode.lower() == "train"):
00246                 self.train(self.args.dataset, self.args.separator, self.args.output,
00247                 output_forced=True)
00248
00249             elif (self.args.mode.lower() == "combine"):
00250                 self.train(self.args.dataset, self.args.separator, self.args.output)
00251                 self.generate(self.args.wordlist)
00252
00253             else:
00254                 logging pprint("Invalid mode arguement given.")
00255                 logging pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00256                 exit(5)

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), [Markov::GUI::Generate.train\(\)](#), [Markov::GUI::Train.train\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



9.6.3.14 train()

```
def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False )
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

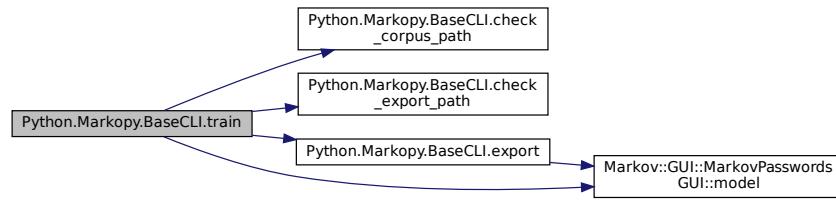
Definition at line 94 of file [base.py](#).

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104         """
00105         logging pprint("Training.")
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging pprint(f"Training mode requires -d/--dataset', '-o/--output' if output_forced
00108             else") and -s/-seperator parameters. Exiting.")
00109         return False
00110         if not bulk and not self.check_corpus_path(dataset):
00111             logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112             return False
00113         if not self.check_export_path(output):
00114             logging pprint(f"Cannot create output at {output}")
00115             return False
00116         if(seperator == '\\t'):
00117             logging pprint("Escaping seperator.", 3)
00118             seperator = '\t'
00119         if(len(seperator) != 1):
00120             logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00121             accepted.')
00122             exit(4)
00123         logging pprint(f'Starting training.', 3)
00124         self.model.Train(dataset, seperator, int(self.args.threads))
00125         logging pprint(f'Training completed.', 2)
00126         if(output):
00127             logging pprint(f'Exporting model to {output}', 2)
00128             self.export(output)
00129         else:
00130             logging pprint(f'Model will not be exported.', 1)
00131         return True
00132
00133
00134
00135
00136
00137
```

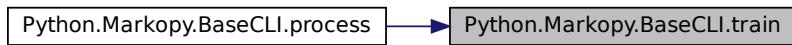
References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.6.4 Member Data Documentation

9.6.4.1 args

`Python.Markopy.BaseCLI.args`

Definition at line 75 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.MarkopyCLI.help\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.parse\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.parse_fail\(\)](#), [Python.Markopy.BaseCLI.process\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.6.4.2 model

`Python.Markopy.BaseCLI.model`

Definition at line 40 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.6.4.3 parser

`Python.Markopy.BaseCLI.parser`

Definition at line 25 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.print_help\(\)](#).

9.6.4.4 print_help

`Python.Markopy.BaseCLI.print_help`

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

The documentation for this class was generated from the following file:

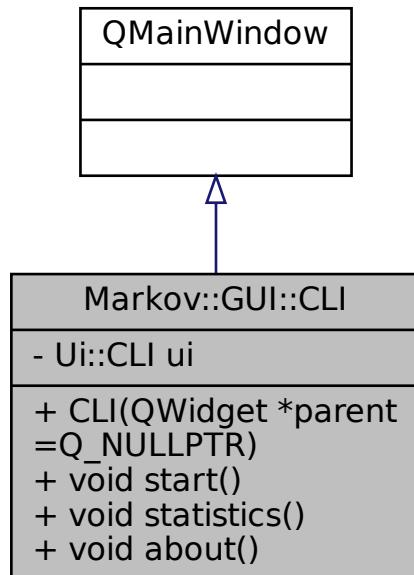
- [Markopy/Markopy/src/CLI/base.py](#)

9.7 Markov::GUI::CLI Class Reference

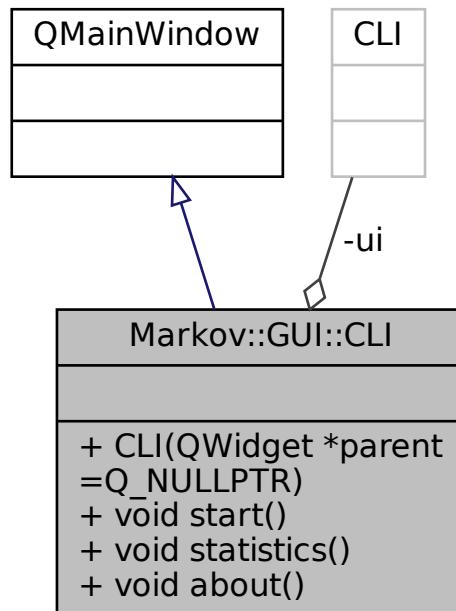
QT CLI Class.

```
#include <CLI.h>
```

Inheritance diagram for Markov::GUI::CLI:



Collaboration diagram for Markov::GUI::CLI:



Public Slots

- void [start \(\)](#)
- void [statistics \(\)](#)
- void [about \(\)](#)

Public Member Functions

- [CLI \(QWidget *parent=Q_NULLPTR\)](#)

Private Attributes

- [Ui::CLI ui](#)

9.7.1 Detailed Description

QT [CLI](#) Class.

Definition at line 14 of file [CLI.h](#).

9.7.2 Constructor & Destructor Documentation

9.7.2.1 CLI()

```
Markov::GUI::CLI::CLI (
    QWidget * parent = Q_NULLPTR )
Definition at line 15 of file CLI.cpp.
00016     : QMainWindow(parent)
```

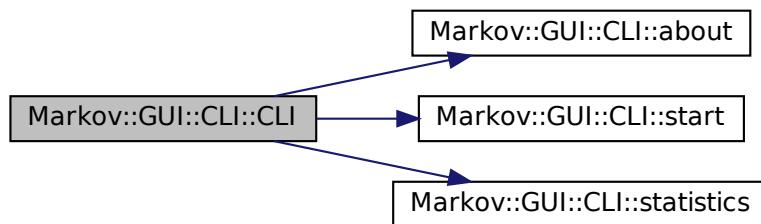
```

00017 {
00018     ui.setupUi(this);
00019
00020     QObject::connect(ui.startButton, &QPushButton::clicked, this, [this] { start(); });
00021     QObject::connect(ui.commandLinkButton_2, &QPushButton::clicked, this, [this] { statistics(); });
00022     QObject::connect(ui.commandLinkButton, &QPushButton::clicked, this, [this] { about(); });
00023
00024 }

```

References [about\(\)](#), [start\(\)](#), [statistics\(\)](#), and [ui](#).

Here is the call graph for this function:



9.7.3 Member Function Documentation

9.7.3.1 about

```

void Markov::GUI::CLI::about ( ) [slot]
Definition at line 36 of file CLI.cpp.
00036 {
00037     /*
00038     about button
00039     */
00040 }

```

Referenced by [CLI\(\)](#).

Here is the caller graph for this function:



9.7.3.2 start

```

void Markov::GUI::CLI::start ( ) [slot]
Definition at line 26 of file CLI.cpp.
00026 {
00027     Train* w = new Train;
00028     w->show();
00029     this->close();
00030 }

```

Referenced by [CLI\(\)](#).

Here is the caller graph for this function:



9.7.3.3 statistics

```
void Markov::GUI::CLI::statistics ( ) [slot]
Definition at line 31 of file CLI.cpp.
00031
00032     /*
00033     statistic will show
00034     */
00035 }
```

Referenced by [CLI\(\)](#).

Here is the caller graph for this function:



9.7.4 Member Data Documentation

9.7.4.1 ui

```
Ui::CLI Markov::GUI::CLI::ui [private]
```

Definition at line 20 of file [CLI.h](#).

Referenced by [CLI\(\)](#).

The documentation for this class was generated from the following files:

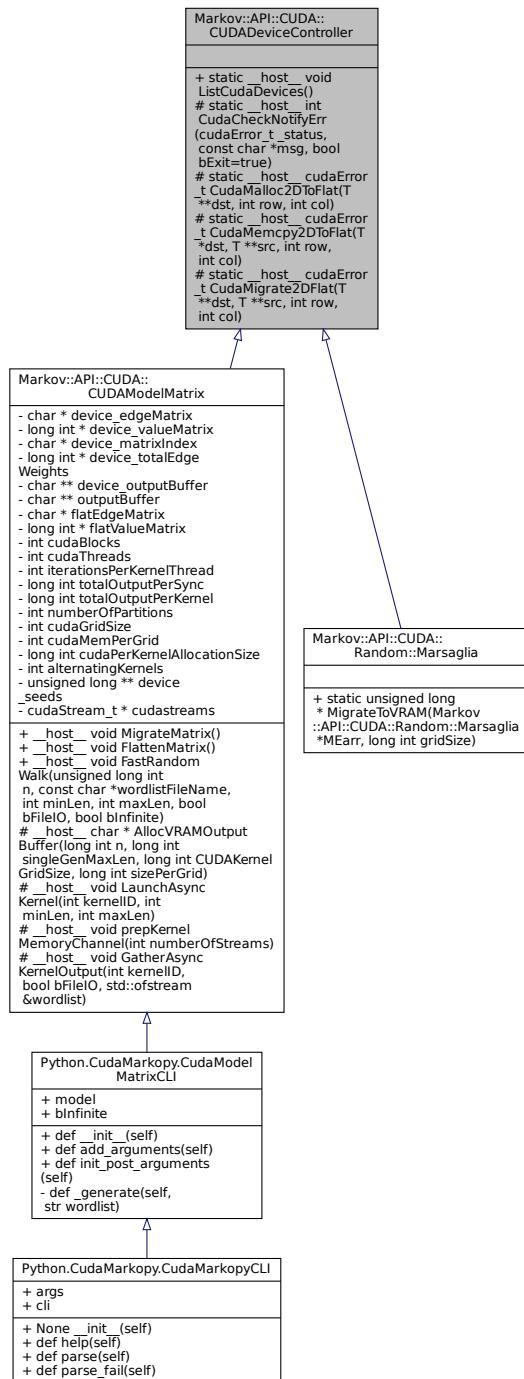
- [Markopy/MarkovPasswordsGUI/src/CLI.h](#)
- [Markopy/MarkovPasswordsGUI/src/CLI.cpp](#)

9.8 Markov::API::CUDA::CUDADeviceController Class Reference

Controller class for [CUDA](#) device.

```
#include <cudaDeviceController.h>
```

Inheritance diagram for Markov::API::CUDA::CUDADeviceController:



Collaboration diagram for Markov::API::CUDA::CUDADeviceController:

Markov::API::CUDA:: CUDADeviceController
<pre>+ static __host__ void ListCudaDevices() # static __host__ int CudaCheckNotifyErr (cudaError_t _status, const char *msg, bool bExit=true) # static __host__ cudaError _t CudaMalloc2DToFlat(T **dst, int row, int col) # static __host__ cudaError _t CudaMemcpy2DToFlat(T *dst, T **src, int row, int col) # static __host__ cudaError _t CudaMigrate2DFlat(T **dst, T **src, int row, int col)</pre>

Static Public Member Functions

- static __host__ void [ListCudaDevices \(\)](#)

List CUDA devices in the system.

Static Protected Member Functions

- static __host__ int [CudaCheckNotifyErr](#) (cudaError_t _status, const char *msg, bool bExit=true)

Check results of the last operation on GPU.
- template<typename T >
 static __host__ cudaError_t [CudaMalloc2DToFlat](#) (T **dst, int row, int col)

Malloc a 2D array in device space.
- template<typename T >
 static __host__ cudaError_t [CudaMemcpy2DToFlat](#) (T *dst, T **src, int row, int col)

Memcpy a 2D array in device space after flattening.
- template<typename T >
 static __host__ cudaError_t [CudaMigrate2DFlat](#) (T **dst, T **src, int row, int col)

Both malloc and memcpy a 2D array into device VRAM.

9.8.1 Detailed Description

Controller class for [CUDA](#) device.

This implementation only supports Nvidia devices.
Definition at line 18 of file [cudaDeviceController.h](#).

9.8.2 Member Function Documentation

9.8.2.1 CudaCheckNotifyErr()

```
__host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr (
    cudaError_t _status,
    const char * msg,
    bool bExit = true ) [static], [protected]
```

Check results of the last operation on GPU.

Check the status returned from cudaMalloc/cudaMemcpy to find failures.

If a failure occurs, its assumed beyond redemption, and exited.

Parameters

<code>_status</code>	Cuda error status to check
<code>msg</code>	Message to print in case of a failure

Returns

0 if successful, 1 if failure. **Example output:**

```
char *da, a = "test";
cudastatus = cudaMalloc((char **)&da, 5*sizeof(char*));
CudaCheckNotifyErr(cudastatus, "Failed to allocate VRAM for *da.\n");
```

Definition at line 32 of file [cudaDeviceController.cu](#).

```
00032
00033     {
00034         if (_status != cudaSuccess) {
00035             std::cerr << "\033[1;31m" << msg << " -> " << cudaGetErrorString(_status) << "(" << _status <
00036             ")" << "\033[0m" << "\n";
00037         if(bExit) {
00038             cudaDeviceReset();
00039             exit(1);
00040         }
00041     return 0;
00042 }
```

9.8.2.2 CudaMalloc2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat (
    T ** dst,
    int row,
    int col ) [inline], [static], [protected]
```

Malloc a 2D array in device space.

This function will allocate enough space on VRAM for flattened 2D array.

Parameters

<code>dst</code>	destination pointer
<code>row</code>	row size of the 2d array
<code>col</code>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

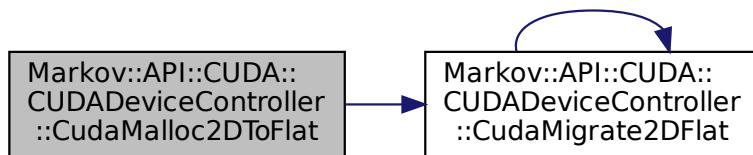
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
}
```

Definition at line 75 of file [cudaDeviceController.h](#).

```
00075     cudaError_t cudastatus = cudaMalloc((T **)dst, row*col*sizeof(T));
00076     CudaCheckNotifyErr(cudastatus, "cudaMalloc Failed.", false);
00077     return cudastatus;
00078 }
```

References [CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:

**9.8.2.3 CudaMemcpy2DToFlat()**

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat (
    T * dst,
    T ** src,
    int row,
    int col ) [inline], [static], [protected]
```

Memcpy a 2D array in device space after flattening.

Resulting buffer will not be true 2D array.

Parameters

<i>dst</i>	destination pointer
<i>rc</i>	source pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
CudaCheckNotifyErr(cudastatus, " CudaMemcpy2DToFlat Failed.", false);
```

Definition at line 103 of file [cudaDeviceController.h](#).

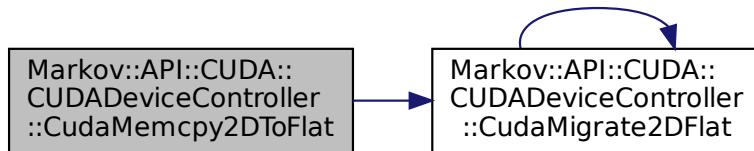
```
00103 {
```

```

00104     T* tempbuf = new T[row*col];
00105     for(int i=0;i<row;i++){
00106         memcpy(&(tempbuf[row*i]), src[i], col);
00107     }
00108     return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109 }
00110 }
```

References [CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:



9.8.2.4 CudaMigrate2DFlat()

```

template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat (
    T ** dst,
    T ** src,
    int row,
    int col ) [inline], [static], [protected]
```

Both malloc and memcpy a 2D array into device VRAM.

Resulting buffer will not be true 2D array.

Parameters

<i>dst</i>	destination pointer
<i>rc</i>	source pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

```

cudaError_t cudastatus;
char* dst;
cudastatus = CudaMigrate2DFlat<long int>(
    &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
CudaCheckNotifyErr(cudastatus, "  Cuda failed to initialize value matrix row.");

```

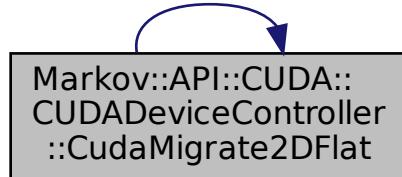
Definition at line 132 of file [cudaDeviceController.h](#).

```

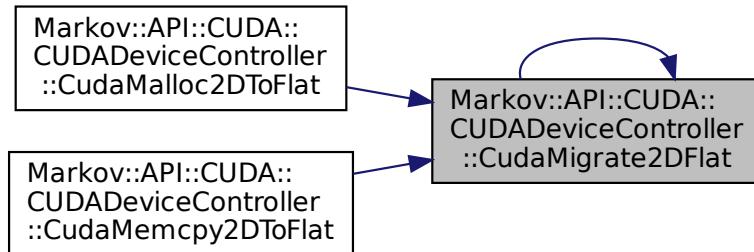
00132
00133     cudaError_t cudastatus;
00134     cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135     if(cudastatus!=cudaSuccess){
00136         CudaCheckNotifyErr(cudastatus, "  CudaMalloc2DToFlat Failed.", false);
00137         return cudastatus;
00138     }
00139     cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140     CudaCheckNotifyErr(cudastatus, "  CudaMemcpy2DToFlat Failed.", false);
00141     return cudastatus;
00142 }
```

References [CudaMigrate2DFlat\(\)](#).

Referenced by [CudaMalloc2DToFlat\(\)](#), [CudaMemcpy2DToFlat\(\)](#), and [CudaMigrate2DFlat\(\)](#).
Here is the call graph for this function:



Here is the caller graph for this function:



9.8.2.5 ListCudaDevices()

`__host__ void Markov::API::CUDA::CUDADeviceController::ListCudaDevices () [static]`
List [CUDA](#) devices in the system.

This function will print details of every [CUDA](#) capable device in the system.

Example output:

```

Device Number: 0
Device name: GeForce RTX 2070
Memory Clock Rate (KHz): 7001000
Memory Bus Width (bits): 256
Peak Memory Bandwidth (GB/s): 448.064
Max Linear Threads: 1024

```

Definition at line 16 of file [cudaDeviceController.cu](#).

```

00016     host.                                         { //list cuda Capable devices on
00017         int nDevices;
00018         cudaGetDeviceCount(&nDevices);
00019         for (int i = 0; i < nDevices; i++) {
00020             cudaDeviceProp prop;
00021             cudaGetDeviceProperties(&prop, i);
00022             std::cerr << "Device Number: " << i << "\n";
00023             std::cerr << "Device name: " << prop.name << "\n";
00024             std::cerr << "Memory Clock Rate (KHz): " << prop.memoryClockRate << "\n";
00025             std::cerr << "Memory Bus Width (bits): " << prop.memoryBusWidth << "\n";
00026             std::cerr << "Peak Memory Bandwidth (GB/s): " << 2.0 * prop.memoryClockRate *
(prop.memoryBusWidth / 8) / 1.0e6 << "\n";
00027             std::cerr << "Max Linear Threads: " << prop.maxThreadsDim[0] << "\n";
00028         }
00029     }
00030 }
```

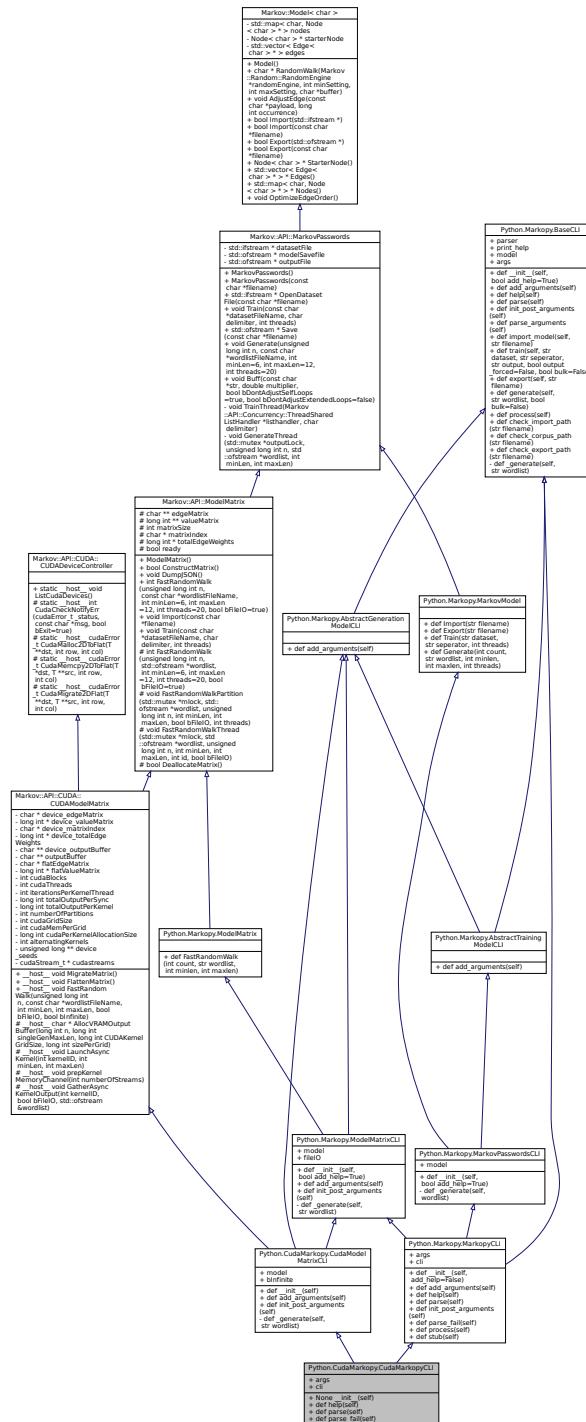
The documentation for this class was generated from the following files:

- [Markopy/CudaMarkovAPI/src/cudaDeviceController.h](#)
 - [Markopy/CudaMarkovAPI/src/cudaDeviceController.cu](#)

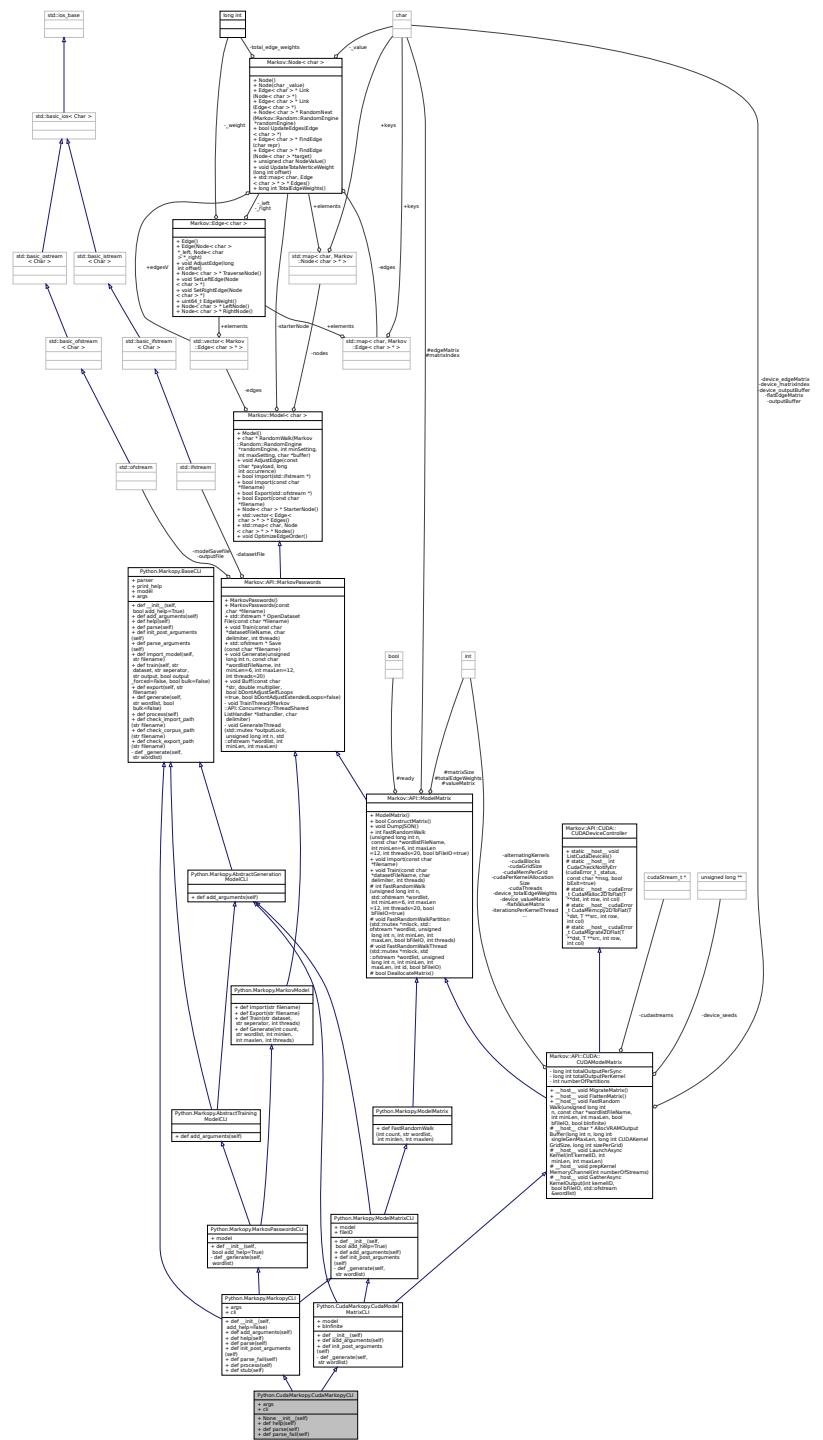
9.9 Python.CudaMarkopy.CudaMarkopyCLI Class Reference

CUDA extension to MarkopyCLI.

Inheritance diagram for Python.CudaMarkopy.CudaMarkopyCLI:



Collaboration diagram for Python.CudaMarkopy.CudaMarkopyCLI:



Public Member Functions

- None `_init_` (self)
 - def `help` (self)
overload help function to print submodel help
 - def `parse` (self)
 - def `parse_fail` (self)
 - def `add_arguments` (self)

add -mt/-model_type constructor

- def `init_post_arguments` (self)
- def `process` (self)

Process parameters for operation.
- def `stub` (self)
- def `parse_arguments` (self)
- def `parse_arguments` (self)
- def `parse_arguments` (self)
- def `parse_arguments` (self)
- def `import_model` (self, str filename)

Import a model file.
- def `import_model` (self, str filename)

Import a model file.
- def `import_model` (self, str filename)

Import a model file.
- def `import_model` (self, str filename)

Import a model file.
- def `import_model` (self, str filename)

Import a model file.
- def `train` (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def `train` (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def `train` (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def `train` (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def `export` (self, str filename)

Export model to a file.
- def `export` (self, str filename)

Export model to a file.
- def `export` (self, str filename)

Export model to a file.
- def `export` (self, str filename)

Export model to a file.
- def `generate` (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def `generate` (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def `generate` (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def `generate` (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def `FastRandomWalk` (int count, str wordlist, int minlen, int maxlen)
- int `FastRandomWalk` (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced `Markov::Model`.
- bool `ConstructMatrix` ()

Construct the related Matrix data for the model.
- void `DumpJSON` ()

Debug function to dump the model to a JSON file.
- void `Import` (const char *filename)

Open a file to import with filename, and call bool `Model::Import` with std::ifstream.

- bool **Import** (std::ifstream *)
Import a file to construct the model.
- def **Import** (str filename)
- bool **Import** (std::ifstream *)
Import a file to construct the model.
- void **Train** (const char *datasetFileName, char delimiter, int threads)
Train the model with the dataset file.
- def **Train** (str dataset, str seperator, int threads)
- std::ifstream * **OpenDatasetFile** (const char *filename)
Open dataset file and return the ifstream pointer.
- std::ifstream * **OpenDatasetFile** (const char *filename)
Open dataset file and return the ifstream pointer.
- std::ofstream * **Save** (const char *filename)
Export model to file.
- std::ofstream * **Save** (const char *filename)
Export model to file.
- void **Generate** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)
Call Markov::Model::RandomWalk n times, and collect output.
- def **Generate** (int count, str wordlist, int minlen, int maxlen, int threads)
- void **Generate** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)
Call Markov::Model::RandomWalk n times, and collect output.
- void **Buff** (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)
Buff expression of some characters in the model.
- void **Buff** (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)
Buff expression of some characters in the model.
- char * **RandomWalk** (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)
Do a random walk on this model.
- char * **RandomWalk** (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)
Do a random walk on this model.
- void **AdjustEdge** (const char *payload, long int occurrence)
Adjust the model with a single string.
- void **AdjustEdge** (const char *payload, long int occurrence)
Adjust the model with a single string.
- bool **Export** (std::ofstream *)
Export a file of the model.
- bool **Export** (const char *filename)
Open a file to export with filename, and call bool Model::Export with std::ofstream.
- def **Export** (str filename)
- bool **Export** (std::ofstream *)
Export a file of the model.
- bool **Export** (const char *filename)
Open a file to export with filename, and call bool Model::Export with std::ofstream.
- Node<char> * **StarterNode** ()
Return starter Node.
- Node<char> * **StarterNode** ()
Return starter Node.

- std::vector< Edge< char > * > * **Edges** ()

Return a vector of all the edges in the model.
- std::vector< Edge< char > * > * **Edges** ()

Return a vector of all the edges in the model.
- std::map< char, Node< char > * > * **Nodes** ()

Return starter Node.
- std::map< char, Node< char > * > * **Nodes** ()

Return starter Node.
- void **OptimizeEdgeOrder** ()

Sort edges of all nodes in the model ordered by edge weights.
- void **OptimizeEdgeOrder** ()

Sort edges of all nodes in the model ordered by edge weights.
- def **add_arguments** (self)
- def **init_post_arguments** (self)
- def **parse_arguments** (self)
- def **parse_arguments** (self)
- def **import_model** (self, str filename)

Import a model file.
- def **import_model** (self, str filename)

Import a model file.
- def **train** (self, str dataset, str separator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def **train** (self, str dataset, str separator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def **export** (self, str filename)

Export model to a file.
- def **export** (self, str filename)

Export model to a file.
- def **generate** (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def **generate** (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def **FastRandomWalk** (int count, str wordlist, int minlen, int maxlen)
- int **FastRandomWalk** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced `Markov::Model`.
- __host__ void **FastRandomWalk** (unsigned long int n, const char *wordlistFileName, int minLen, int maxLen, bool bFileIO, bool bInfinite)

Random walk on the Matrix-reduced `Markov::Model`.
- int **FastRandomWalk** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced `Markov::Model`.
- bool **ConstructMatrix** ()

Construct the related Matrix data for the model.
- bool **ConstructMatrix** ()

Construct the related Matrix data for the model.
- void **DumpJSON** ()

Debug function to dump the model to a JSON file.
- void **DumpJSON** ()

Debug function to dump the model to a JSON file.
- void **Import** (const char *filename)

- Open a file to import with filename, and call bool Model::Import with std::ifstream.*
- **bool Import** (std::ifstream *)
Import a file to construct the model.
 - **void Import** (const char *filename)
Open a file to import with filename, and call bool Model::Import with std::ifstream.
 - **bool Import** (std::ifstream *)
Import a file to construct the model.
 - **void Train** (const char *datasetFileName, char delimiter, int threads)
Train the model with the dataset file.
 - **void Train** (const char *datasetFileName, char delimiter, int threads)
Train the model with the dataset file.
 - **std::ifstream * OpenDatasetFile** (const char *filename)
Open dataset file and return the ifstream pointer.
 - **std::ifstream * OpenDatasetFile** (const char *filename)
Open dataset file and return the ifstream pointer.
 - **std::ofstream * Save** (const char *filename)
Export model to file.
 - **std::ofstream * Save** (const char *filename)
Export model to file.
 - **void Generate** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)
Call Markov::Model::RandomWalk n times, and collect output.
 - **void Generate** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)
Call Markov::Model::RandomWalk n times, and collect output.
 - **void Buff** (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)
Buff expression of some characters in the model.
 - **void Buff** (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)
Buff expression of some characters in the model.
 - **char * RandomWalk** (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)
Do a random walk on this model.
 - **char * RandomWalk** (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)
Do a random walk on this model.
 - **void AdjustEdge** (const char *payload, long int occurrence)
Adjust the model with a single string.
 - **void AdjustEdge** (const char *payload, long int occurrence)
Adjust the model with a single string.
 - **bool Export** (std::ofstream *)
Export a file of the model.
 - **bool Export** (const char *filename)
Open a file to export with filename, and call bool Model::Export with std::ofstream.
 - **bool Export** (std::ofstream *)
Export a file of the model.
 - **bool Export** (const char *filename)
Open a file to export with filename, and call bool Model::Export with std::ofstream.
 - **Node< char > * StarterNode** ()
Return starter Node.

- `Node< char > * StarterNode ()`
Return starter Node.
- `std::vector< Edge< char > * > * Edges ()`
Return a vector of all the edges in the model.
- `std::vector< Edge< char > * > * Edges ()`
Return a vector of all the edges in the model.
- `std::map< char, Node< char > * > * Nodes ()`
Return starter Node.
- `std::map< char, Node< char > * > * Nodes ()`
Return starter Node.
- `void OptimizeEdgeOrder ()`
Sort edges of all nodes in the model ordered by edge weights.
- `void OptimizeEdgeOrder ()`
Sort edges of all nodes in the model ordered by edge weights.
- `__host__ void MigrateMatrix ()`
Migrate the class members to the VRAM.
- `__host__ void FlattenMatrix ()`
Flatten migrated matrix from 2d to 1d.

Static Public Member Functions

- `def check_import_path (str filename)`
check import path for validity
- `def check_import_path (str filename)`
check import path for validity
- `def check_import_path (str filename)`
check import path for validity
- `def check_import_path (str filename)`
check import path for validity
- `def check_import_path (str filename)`
check import path for validity
- `def check_corpus_path (str filename)`
check import path for validity
- `def check_corpus_path (str filename)`
check import path for validity
- `def check_corpus_path (str filename)`
check import path for validity
- `def check_corpus_path (str filename)`
check import path for validity
- `def check_corpus_path (str filename)`
check import path for validity
- `def check_export_path (str filename)`
check import path for validity
- `def check_export_path (str filename)`
check import path for validity
- `def check_export_path (str filename)`
check import path for validity
- `def check_import_path (str filename)`
check import path for validity
- `def check_import_path (str filename)`
check import path for validity
- `def check_import_path (str filename)`
check import path for validity
- `def check_import_path (str filename)`
check import path for validity
- `def check_corpus_path (str filename)`
check import path for validity

- def `check_corpus_path` (str filename)
check import path for validity
- def `check_export_path` (str filename)
check import path for validity
- def `check_export_path` (str filename)
check import path for validity
- static __host__ void `ListCudaDevices` ()
List CUDA devices in the system.

Public Attributes

- args
- cli
- parser
- parser
- parser
- parser
- print_help
- print_help
- print_help
- print_help
- model
- model
- model
- fileIO
- model
- bInfinite
- fileIO
- parser
- parser
- print_help
- print_help

Protected Member Functions

- int `FastRandomWalk` (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)
Random walk on the Matrix-reduced Markov::Model.
- void `FastRandomWalkPartition` (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)
A single partition of FastRandomWalk event.
- void `FastRandomWalkThread` (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)
A single thread of a single partition of FastRandomWalk.
- bool `DeallocateMatrix` ()
Deallocate matrix and make it ready for re-construction.
- int `FastRandomWalk` (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)
Random walk on the Matrix-reduced Markov::Model.
- int `FastRandomWalk` (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)
Random walk on the Matrix-reduced Markov::Model.
- void `FastRandomWalkPartition` (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)
Random walk on the Matrix-reduced Markov::Model.

- A single partition of FastRandomWalk event.
- void **FastRandomWalkPartition** (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)
 - A single partition of FastRandomWalk event.
- void **FastRandomWalkThread** (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)
 - A single thread of a single partition of FastRandomWalk.
- void **FastRandomWalkThread** (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)
 - A single thread of a single partition of FastRandomWalk.
- bool **DeallocateMatrix** ()
 - Deallocate matrix and make it ready for re-construction.
- bool **DeallocateMatrix** ()
 - Deallocate matrix and make it ready for re-construction.
- __host__ char * **AllocVRAMOutputBuffer** (long int n, long int singleGenMaxLen, long int CUDAKernelGridSize, long int sizePerGrid)
 - Allocate the output buffer for kernel operation.
- __host__ void **LaunchAsyncKernel** (int kernelID, int minLen, int maxLen)
- __host__ void **prepKernelMemoryChannel** (int numberOfWorkStreams)
- __host__ void **GatherAsyncKernelOutput** (int kernelID, bool bFileIO, std::ofstream &wordlist)

Static Protected Member Functions

- static __host__ int **CudaCheckNotifyErr** (cudaError_t _status, const char *msg, bool bExit=true)
 - Check results of the last operation on GPU.
- template<typename T >
 - static __host__ cudaError_t **CudaMalloc2DToFlat** (T **dst, int row, int col)
 - Malloc a 2D array in device space.
 - static __host__ cudaError_t **CudaMemcpy2DToFlat** (T *dst, T **src, int row, int col)
 - Memcpy a 2D array in device space after flattening.
 - static __host__ cudaError_t **CudaMigrate2DFlat** (T **dst, T **src, int row, int col)
 - Both malloc and memcpy a 2D array into device VRAM.

Protected Attributes

- char ** **edgeMatrix**
 - 2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** **valueMatrix**
 - 2-d Integer array for the value Matrix (For the weights of Edges)
- int **matrixSize**
 - to hold Matrix size
- char * **matrixIndex**
 - to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * **totalEdgeWeights**
 - Array of the Total Edge Weights.
- bool **ready**
 - True when matrix is constructed. False if not.
- char ** **edgeMatrix**
 - 2-D Character array for the edge Matrix (The characters of Nodes)
- char ** **edgeMatrix**
 - 2-D Character array for the edge Matrix (The characters of Nodes)

- long int ** **valueMatrix**
2-d Integer array for the value Matrix (For the weights of Edges)
- long int ** **valueMatrix**
2-d Integer array for the value Matrix (For the weights of Edges)
- int **matrixSize**
to hold Matrix size
- int **matrixSize**
to hold Matrix size
- char * **matrixIndex**
to hold the Matrix index (To hold the orders of 2-D arrays')
- char * **matrixIndex**
to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * **totalEdgeWeights**
Array of the Total Edge Weights.
- long int * **totalEdgeWeights**
Array of the Total Edge Weights.
- bool **ready**
True when matrix is constructed. False if not.
- bool **ready**
True when matrix is constructed. False if not.

Private Member Functions

- def **_generate** (self, str wordlist)
wrapper for generate function.
- void **TrainThread** (Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char delimiter)
A single thread invoked by the Train function.
- void **GenerateThread** (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)
A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * **datasetFile**
- std::ofstream * **modelSavefile**
Dataset file input of our system
- std::ofstream * **outputFile**
File to save model of our system
- std::map< char, Node< char > * > **nodes**
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > * **starterNode**
Starter Node of this model.
- std::vector< Edge< char > * > **edges**
A list of all edges in this model.
- char * **device_edgeMatrix**
*VRAM Address pointer of edge matrix (from *modelMatrix.h*)*
- long int * **device_valueMatrix**
*VRAM Address pointer of value matrix (from *modelMatrix.h*)*
- char * **device_matrixIndex**
*VRAM Address pointer of matrixIndex (from *modelMatrix.h*)*

- long int * `device_totalEdgeWeights`
VRAM Address pointer of total edge weights (from `modelMatrix.h`)
- char ** `device_outputBuffer`
RandomWalk results in device.
- char ** `outputBuffer`
RandomWalk results in host.
- char * `flatEdgeMatrix`
Adding `Edge` matrix end-to-end and resize to 1-D array for better performance on traversing.
- long int * `flatValueMatrix`
Adding `Value` matrix end-to-end and resize to 1-D array for better performance on traversing.
- int `cudaBlocks`
- int `cudaThreads`
- int `iterationsPerKernelThread`
- long int `totalOutputPerSync`
- long int `totalOutputPerKernel`
- int `numberOfPartitions`
- int `cudaGridSize`
- int `cudaMemPerGrid`
- long int `cudaPerKernelAllocationSize`
- int `alternatingKernels`
- unsigned long ** `device_seeds`
- cudaStream_t * `cudastreams`

9.9.1 Detailed Description

CUDA extension to MarkopyCLI.

Adds `CudaModelMatrixCLI` to the interface.

Definition at line 46 of file `cudamarkopy.py`.

9.9.2 Constructor & Destructor Documentation

9.9.2.1 `__init__()`

```
None Python.CudaMarkopy.CudaMarkopyCLI.__init__ (
    self )
Reimplemented from Python.CudaMarkopy.CudaModelMatrixCLI.
Definition at line 53 of file cudamarkopy.py.
00053     def __init__(self) -> None:
00054         """ @brief initialize CLI selector"""
00055         markopy.MarkopyCLI.__init__(self, add_help=False)
00056         self.parser.epilog+=f"""
00057             {__file__.split("/")[-1]} -mt CUDA generate trained.mdl -n 500 -w output.txt
00058                 Import trained.mdl, and generate 500 lines to output.txt
00059             """
00060
References Python.Markopy.BaseCLI.parser.
```

9.9.3 Member Function Documentation

9.9.3.1 `_generate()`

```
def Python.Markopy.BaseCLI._generate (
    self,
    str wordlist ) [private], [inherited]
wrapper for generate function.
This can be overloaded by other models
```

Parameters

<code>wordlist</code>	filename to generate to
-----------------------	-------------------------

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

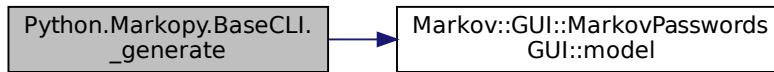
Definition at line 161 of file [base.py](#).

```
00161     def _generate(self, wordlist : str):
00162         """
00163             @brief wrapper for generate function. This can be overloaded by other models
00164             @param wordlist filename to generate to
00165         """
00166         self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
00167                         int(self.args.threads))
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.9.3.2 add_arguments() [1/2]**

```
def Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments (
    self) [inherited]
```

Reimplemented from [Python.Markopy.AbstractGenerationModelCLI](#).

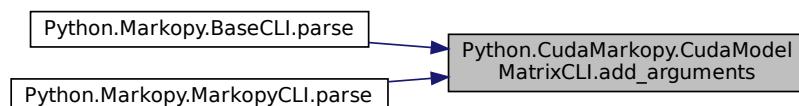
Definition at line 67 of file [cudammx.py](#).

```
00067     def add_arguments(self):
00068         super().add_arguments()
00069         self.parser.add_argument("-if", "--infinite", action="store_true", help="Infinite generation
mode")
```

References [Python.Markopy.BaseCLI.parser](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.9.3.3 add_arguments() [2/2]

```
def Python.Markopy.MarkopyCLI.add_arguments (
    self )  [inherited]
```

add -mt/-model_type constructor

Reimplemented from [Python.Markopy.BaseCLI](#).

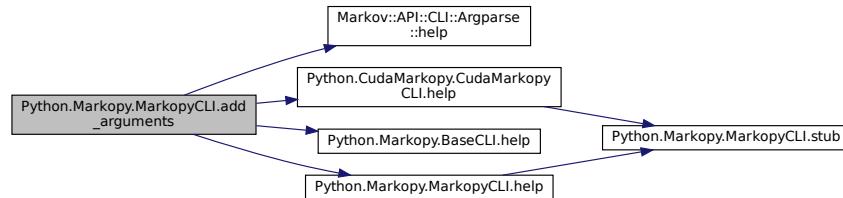
Definition at line 86 of file [markopy.py](#).

```
00086     def add_arguments(self):
00087         """
00088             @brief add -mt/--model_type constructor
00089             """
00090             self.parser.add_argument("-mt", "--model_type", default="_MMX", help="Model type to use.
Accepted values: MP, MMX")
00091             self.parser.add_argument("-h", "--help", action="store_true", help="Model type to use.
Accepted values: MP, MMX")
00092             self.parser.print_help = self.help
00093
```

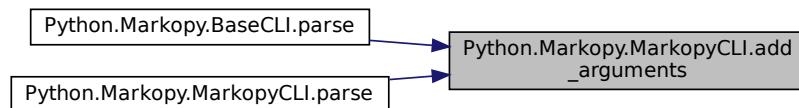
References [Markov::API::CLI::Argparse::help\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI::help\(\)](#), [Python.Markopy.BaseCLI::help\(\)](#), [Python.Markopy.MarkopyCLI::help\(\)](#), and [Python.Markopy.BaseCLI::parser](#).

Referenced by [Python.Markopy.BaseCLI::parse\(\)](#), and [Python.Markopy.MarkopyCLI::parse\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.4 AdjustEdge() [1/4]

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence )  [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
```

```
model.AdjustEdge(test, 15);
```

Parameters

<i>string</i>	- String that is passed from the training, and will be used to AdjustEdge the model with
<i>occurrence</i>	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```

00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.9.3.5 AdjustEdge() [2/4]

```
void Markov::Model<char>::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

<i>string</i>	- String that is passed from the training, and will be used to AdjustEdge the model with
<i>occurrence</i>	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```

00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.9.3.6 AdjustEdge() [3/4]

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.9.3.7 AdjustEdge() [4/4]

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
```

```

00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[+i];
00350     }
00351     e = curnode->FindEdge('\xff');
00352     e->AdjustEdge(occurrence);
00353     return;
00354 }
00355 }
```

9.9.3.8 AllocVRAMOutputBuffer()

```

__host__ char* Markov::API::CUDA::CUDAModelMatrix::AllocVRAMOutputBuffer (
    long int n,
    long int singleGenMaxLen,
    long int CUDAKernelGridSize,
    long int sizePerGrid ) [protected], [inherited]
```

Allocate the output buffer for kernel operation.

TODO

Parameters

<i>n</i>	- Number of passwords to generate.
<i>singleGenMaxLen</i>	- maximum string length for a single generation
<i>CUDAKernelGridSize</i>	- Total number of grid members in CUDA kernel
<i>sizePerGrid</i>	- Size to allocate per grid member

Returns

pointer to the allocation on VRAM

9.9.3.9 Buff() [1/4]

```

void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]
```

Buff expression of some characters in the model.

Parameters

<i>str</i>	A string containing all the characters to be buffed
<i>multiplier</i>	A constant value to buff the nodes with.
<i>bDontAdjustSelfEdges</i>	Do not adjust weights if target node is same as source node
<i>bDontAdjustExtendedLoops</i>	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

```

00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
```

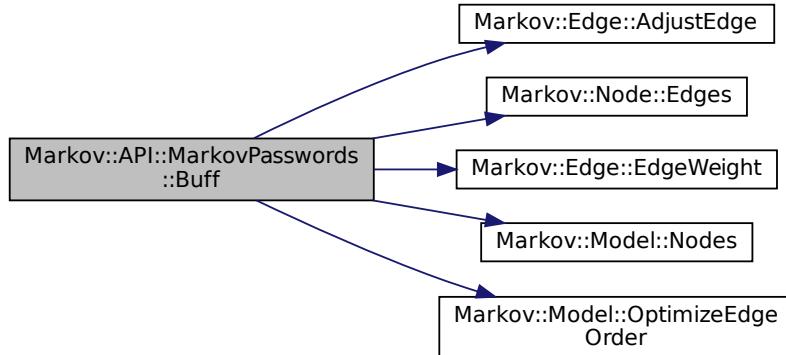
```

00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges) {
00162             if(buffstr.find(targetrepr)!= std::string::npos) {
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops) {
00165                     if(buffstr.find(repr)!= std::string::npos) {
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173         }
00174         i++;
00175     }
00176 }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.10 Buff() [2/4]

```
void Markov::API::MarkovPasswords::Buff (
    const char * str,
```

```

    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]

```

Buff expression of some characters in the model.

Parameters

<i>str</i>	A string containing all the characters to be buffed
<i>multiplier</i>	A constant value to buff the nodes with.
<i>bDontAdjustSelfEdges</i>	Do not adjust weights if target node is same as source node
<i>bDontAdjustExtendedLoops</i>	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

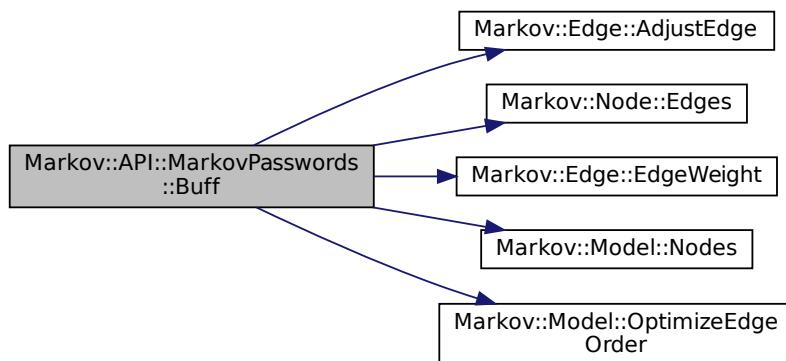
```

00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges) {
00162             if(buffstr.find(targetrepr)!= std::string::npos) {
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops) {
00165                     if(buffstr.find(repr)!= std::string::npos) {
00166                         continue;
00167                     }
00168                     long int weight = edge->EdgeWeight ();
00169                     weight = weight*multiplier;
00170                     edge->AdjustEdge(weight);
00171                 }
00172             }
00173         }
00174         i++;
00175     }
00176     this->OptimizeEdgeOrder ();
00177
00178
00179 }
```

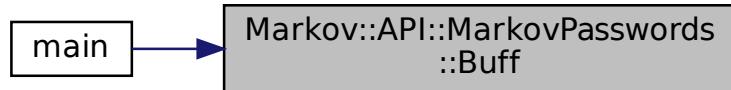
References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.11 Buff() [3/4]

```

void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]
  
```

Buff expression of some characters in the model.

Parameters

<i>str</i>	A string containing all the characters to be buffed
<i>multiplier</i>	A constant value to buff the nodes with.
<i>bDontAdjustSelfEdges</i>	Do not adjust weights if target node is same as source node
<i>bDontAdjustExtendedLoops</i>	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

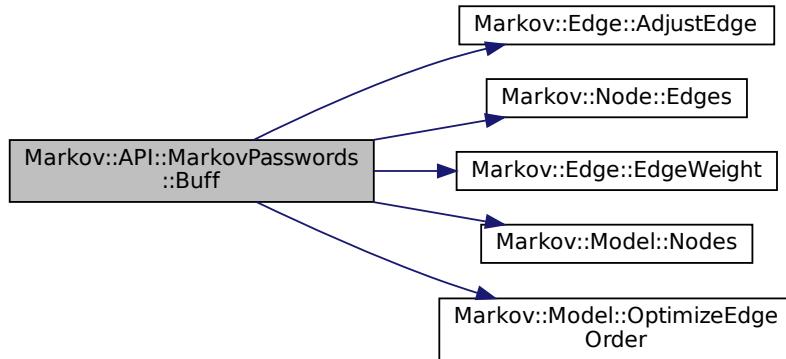
```

00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos) {
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops) {
00165                     if(buffstr.find(repr)!= std::string::npos) {
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173         }
00174     }
00175     i++;
00176 }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.12 Buff() [4/4]

```

void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false )  [inherited]
  
```

Buff expression of some characters in the model.

Parameters

<code>str</code>	A string containing all the characters to be buffed
<code>multiplier</code>	A constant value to buff the nodes with.
<code>bDontAdjustSelfEdges</code>	Do not adjust weights if target node is same as source node
<code>bDontAdjustExtendedLoops</code>	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

```

00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
  
```

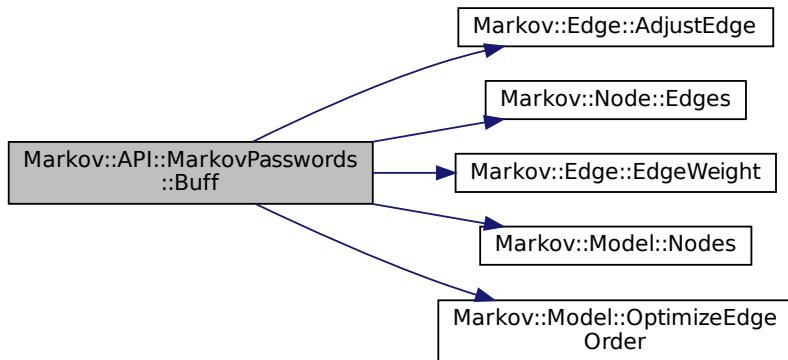
```

00160     edges = node->Edges();
00161     for (auto const& [targetrepr, edge] : *edges){
00162         if (buffstr.find(targetrepr) != std::string::npos) {
00163             if (bDontAdjustSelfLoops && repr==targetrepr) continue;
00164             if (bDontAdjustExtendedLoops) {
00165                 if (buffstr.find(repr) != std::string::npos) {
00166                     continue;
00167                 }
00168                 long int weight = edge->EdgeWeight();
00169                 weight = weight*multiplier;
00170                 edge->AdjustEdge(weight);
00171             }
00172         }
00173     }
00174     i++;
00175 }
00176 }
00177 this->OptimizeEdgeOrder();
00179 }
```

References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.13 check_corpus_path() [1/6]

```

def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

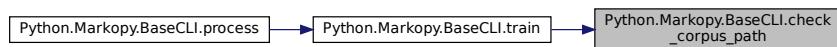
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:

**9.9.3.14 check_corpus_path() [2/6]**

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:

**9.9.3.15 check_corpus_path() [3/6]**

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.9.3.16 check_corpus_path() [4/6]

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

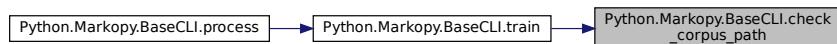
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.9.3.17 check_corpus_path() [5/6]

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

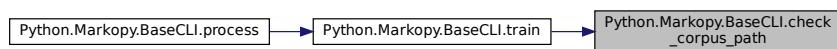
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:

**9.9.3.18 check_corpus_path() [6/6]**

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:

**9.9.3.19 check_export_path() [1/6]**

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

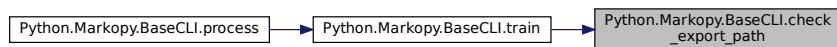
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.9.3.20 check_export_path() [2/6]

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

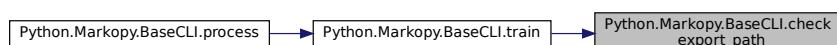
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.9.3.21 check_export_path() [3/6]

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

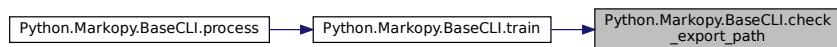
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:

**9.9.3.22 check_export_path() [4/6]**

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

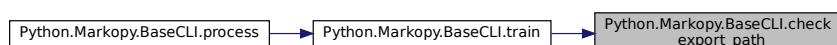
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:

**9.9.3.23 check_export_path() [5/6]**

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

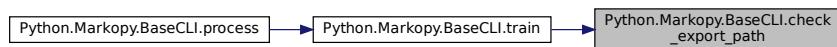
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:

**9.9.3.24 check_export_path() [6/6]**

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:

**9.9.3.25 check_import_path() [1/6]**

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

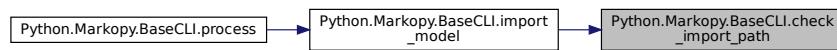
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:

**9.9.3.26 check_import_path() [2/6]**

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

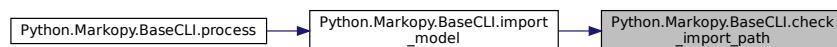
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:

**9.9.3.27 check_import_path() [3/6]**

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

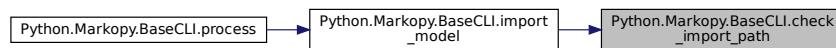
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:

**9.9.3.28 check_import_path() [4/6]**

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:

**9.9.3.29 check_import_path() [5/6]**

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

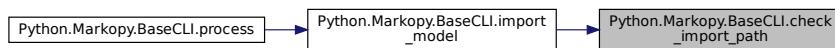
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.9.3.30 check_import_path() [6/6]

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

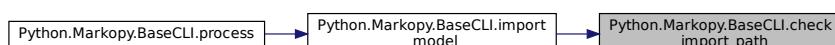
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.9.3.31 ConstructMatrix() [1/3]

```
bool Markov::API::ModelMatrix::ConstructMatrix ( ) [inherited]
```

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content.

this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

True if constructed. False if already construced.

Definition at line 31 of file `modelMatrix.cpp`.

```

00031         {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072         }
00073     }
00074     i++;
00075   }
00076   this->ready = true;
00077   return true;
00078 //this->DumpJSON();
00079 }
```

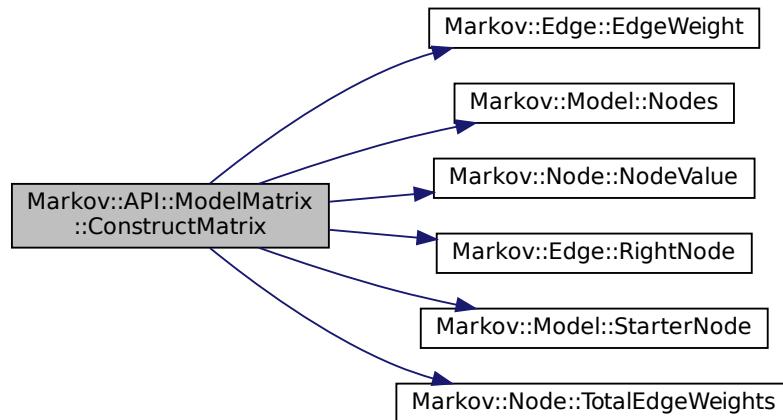
References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::API::ModelMatrix::Markov::API::ModelMatrix::matrixSize](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), [Markov::Node< storageType >::NodeValue\(\)](#)

[Markov::API::ModelMatrix::ready](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::Starter\(\)](#)

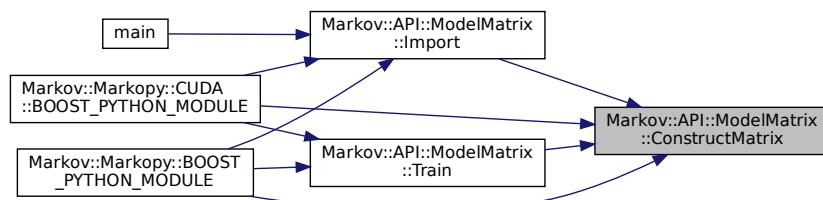
[Markov::API::ModelMatrix::totalEdgeWeights](#), [Markov::Node< storageType >::TotalEdgeWeights\(\)](#), and [Markov::API::ModelMatrix::valueMatrix](#)

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.32 `ConstructMatrix()` [2/3]

```
bool Markov::API::ModelMatrix::ConstructMatrix ( ) [inherited]
Construct the related Matrix data for the model.
```

This operation can be used after importing/training to allocate and populate the matrix content. This will initialize: `char** edgeMatrix` -> a 2D array of mapping left and right connections of each edge. `long int **valueMatrix` -> a 2D array representing the edge weights. `int matrixSize` -> Size of the matrix, aka total number of nodes. `char* matrixIndex` -> order of nodes in the model `long int *totalEdgeWeights` -> total edge weights of each Node.

Returns

True if constructed. False if already constructed.

Definition at line 31 of file [modelMatrix.cpp](#).

```

00031
00032     if(this->ready)  return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
  
```

```

00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes) {
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072         }
00073     }
00074     i++;
00075 }
00076 this->ready = true;
00077 return true;
00078 //this->DumpJSON();
00079 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::API::ModelMatrix::ModelMatrix](#),

[Markov::API::ModelMatrix::matrixSize](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), [Markov::Node< storageType >::NodeValue\(\)](#)

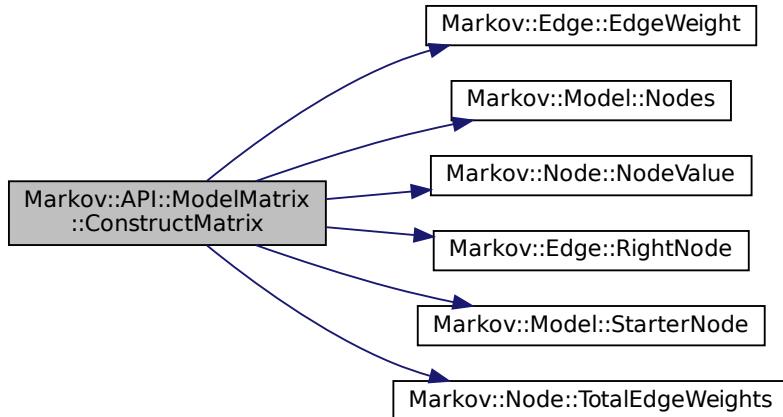
[Markov::API::ModelMatrix::ready](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::StarterNode](#)

[Markov::API::ModelMatrix::totalEdgeWeights](#), [Markov::Node< storageType >::TotalEdgeWeights\(\)](#), and [Markov::API::ModelMatrix::valueMatrix](#)

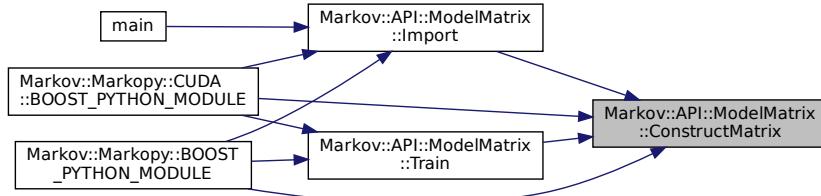
Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#),

[Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.33 ConstructMatrix() [3/3]

```
bool Markov::API::ModelMatrix::ConstructMatrix () [inherited]
```

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content.

this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

True if constructed. False if already constructed.

Definition at line 31 of file [modelMatrix.cpp](#).

```

00031     if(this->ready)  return false;
00032
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042
00043     this->valueMatrix = new long int*[this->matrixSize];
00044     for(int i=0;i<this->matrixSize;i++){
00045         this->valueMatrix[i] = new long int[this->matrixSize];
00046     }
00047     std::map< char, Node< char > * > *nodes;
00048     nodes = this->Nodes();
00049     int i=0;
00050     for (auto const& [repr, node] : *nodes){
00051         if(repr!=0) this->matrixIndex[i] = repr;
00052         else this->matrixIndex[i] = 199;
00053         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00054         for(int j=0;j<this->matrixSize;j++){
00055             char val = node->NodeValue();
00056             if(val < 0){
00057                 for(int k=0;k<this->matrixSize;k++){
00058                     this->valueMatrix[i][k] = 0;
00059                     this->edgeMatrix[i][k] = 255;
00060                 }
00061                 break;
00062             }
00063             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)) {
00064                 this->valueMatrix[i][j] = 0;
00065                 this->edgeMatrix[i][j] = 255;
00066             }else if(j==(this->matrixSize-1)) {
00067                 this->valueMatrix[i][j] = 0;
00068                 this->edgeMatrix[i][j] = 255;
00069             }else{
00070                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00071                 this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();
00072             }
00073         }
00074         i++;
00075     }
00076 }
```

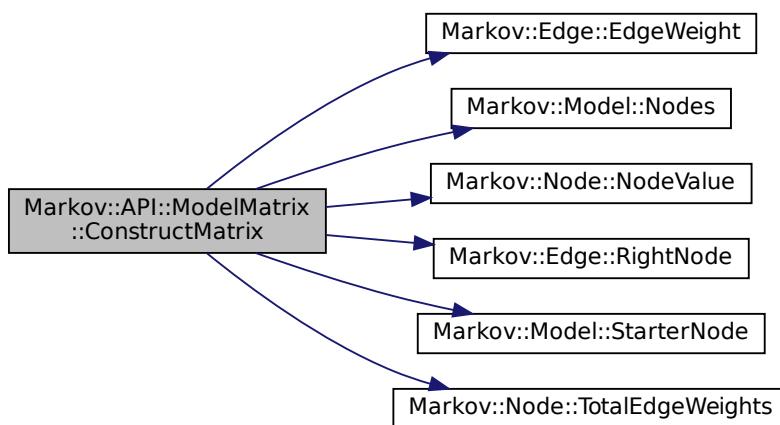
```

00075     }
00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

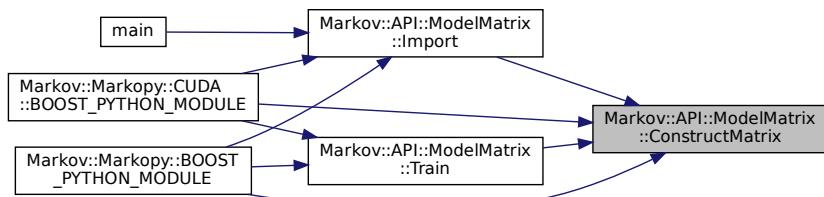
References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::API::ModelMatrix::Markov::API::ModelMatrix::matrixSize](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), [Markov::Node< storageType >::NodeValue\(\)](#), [Markov::API::ModelMatrix::ready](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::StarterNode\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), [Markov::Node< storageType >::TotalEdgeWeights\(\)](#), and [Markov::API::ModelMatrix::value](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.34 CudaCheckNotifyErr()

```

__host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr (
    cudaError_t _status,
    const char * msg,
    bool bExit = true) [static], [protected], [inherited]
```

Check results of the last operation on GPU.

Check the status returned from `cudaMalloc/cudaMemcpy` to find failures.

If a failure occurs, its assumed beyond redemption, and exited.

Parameters

<i>_status</i>	Cuda error status to check
<i>msg</i>	Message to print in case of a failure

Returns0 if successful, 1 if failure. **Example output:**

```
char *da, a = "test";
cudastatus = cudaMalloc((char **)&da, 5*sizeof(char *));
CudaCheckNotifyErr(cudastatus, "Failed to allocate VRAM for *da.\n");
```

Definition at line 32 of file [cudaDeviceController.cu](#).

```
00032
00033     {
00034         if (_status != cudaSuccess) {
00035             std::cerr << "\033[1;31m" << msg << " -> " << cudaGetErrorString(_status) << "(" << _status <
00036             ")" << "\033[0m" << "\n";
00037         if(bExit) {
00038             cudaDeviceReset();
00039             exit(1);
00040         }
00041     return 0;
00042 }
```

9.9.3.35 CudaMalloc2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat (
    T ** dst,
    int row,
    int col ) [inline], [static], [protected], [inherited]
```

Malloc a 2D array in device space.

This function will allocate enough space on VRAM for flattened 2D array.

Parameters

<i>dst</i>	destination pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

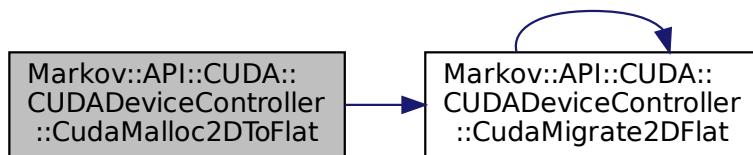
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
}
```

Definition at line 75 of file [cudaDeviceController.h](#).

```
00075
00076     cudaError_t cudastatus = cudaMalloc((T **)dst, row*col*sizeof(T));
00077     CudaCheckNotifyErr(cudastatus, "cudaMalloc Failed.", false);
00078     return cudastatus;
00079 }
```

References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:



9.9.3.36 CudaMemcpy2DToFlat()

```

template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat (
    T * dst,
    T ** src,
    int row,
    int col ) [inline], [static], [protected], [inherited]
Memcpy a 2D array in device space after flattening.
Resulting buffer will not be true 2D array.
  
```

Parameters

<i>dst</i>	destination pointer
<i>rc</i>	source pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

```

cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
CudaCheckNotifyErr(cudastatus, " CudaMemcpy2DToFlat Failed.", false);
  
```

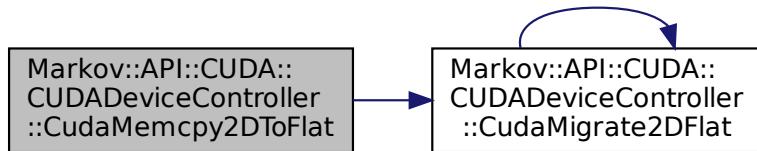
Definition at line 103 of file [cudaDeviceController.h](#).

```

00103
00104      T* tempbuf = new T[row*col];
00105      for(int i=0;i<row;i++){
00106          memcpy(&(tempbuf[row*i]), src[i], col);
00107      }
00108      return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109
00110  }
  
```

References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:



9.9.3.37 CudaMigrate2DFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDADeviceController::CudaMigrate2DFlat (
    T ** dst,
    T ** src,
    int row,
    int col ) [inline], [static], [protected], [inherited]
```

Both malloc and memcpy a 2D array into device VRAM.

Resulting buffer will not be true 2D array.

Parameters

<i>dst</i>	destination pointer
<i>rc</i>	source pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMigrate2DFlat<long int>(
    &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
```

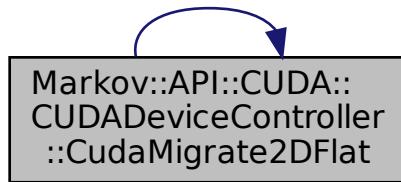
Definition at line 132 of file [cudaDeviceController.h](#).

```
00132
00133     cudaError_t cudastatus;
00134     cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135     if(cudastatus!=cudaSuccess){
00136         CudaCheckNotifyErr(cudastatus, "    CudaMalloc2DToFlat Failed.", false);
00137         return cudastatus;
00138     }
00139     cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140     CudaCheckNotifyErr(cudastatus, "    CudaMemcpy2DToFlat Failed.", false);
00141     return cudastatus;
00142 }
```

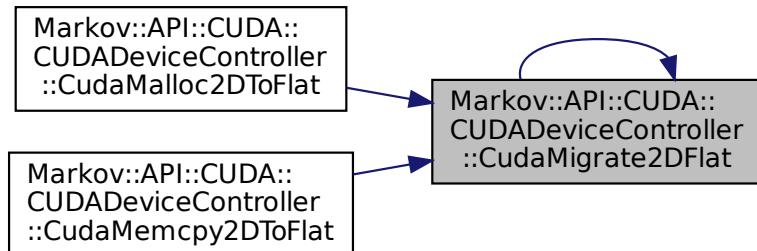
References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Referenced by [Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat\(\)](#), [Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat\(\)](#) and [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.38 DeallocateMatrix() [1/3]

```
bool Markov::API::ModelMatrix::DeallocateMatrix ( ) [protected], [inherited]
Deallocate matrix and make it ready for re-construction.
```

Returns

True if deallocated. False if matrix was not initialized

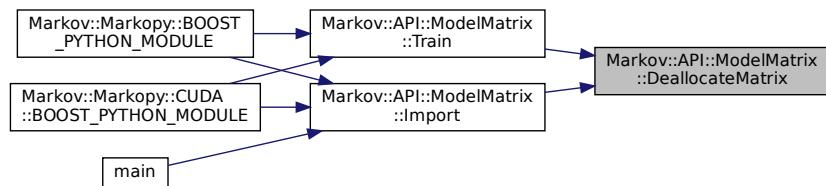
Definition at line 81 of file [modelMatrix.cpp](#).

```

00081
00082     if(!this->ready)  return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085
00086     for(int i=0;i<this->matrixSize;i++) {
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090
00091     for(int i=0;i<this->matrixSize;i++) {
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::API::ModelMatrix::ready](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).
 Here is the caller graph for this function:



9.9.3.39 DeallocateMatrix() [2/3]

`bool Markov::API::ModelMatrix::DeallocateMatrix ()` [protected], [inherited]
 Deallocate matrix and make it ready for re-construction.

Returns

True if deallocated. False if matrix was not initialized

Definition at line 81 of file [modelMatrix.cpp](#).

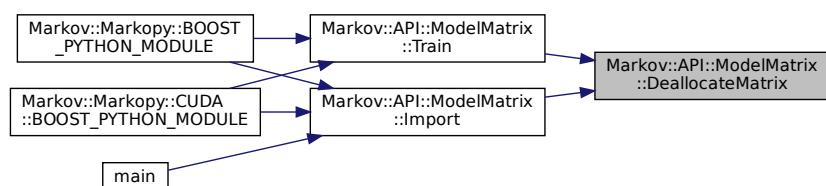
```

00081     if(!this->ready) return false;
00082     {
00083         delete[] this->matrixIndex;
00084         delete[] this->totalEdgeWeights;
00085
00086         for(int i=0;i<this->matrixSize;i++) {
00087             delete[] this->edgeMatrix[i];
00088         }
00089         delete[] this->edgeMatrix;
00090
00091         for(int i=0;i<this->matrixSize;i++) {
00092             delete[] this->valueMatrix[i];
00093         }
00094         delete[] this->valueMatrix;
00095
00096         this->matrixSize = -1;
00097         this->ready = false;
00098         return true;
00099     }
  
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::API::ModelMatrix::ready](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the caller graph for this function:



9.9.3.40 DeallocateMatrix() [3/3]

`bool Markov::API::ModelMatrix::DeallocateMatrix ()` [protected], [inherited]
 Deallocate matrix and make it ready for re-construction.

Returns

True if deallocated. False if matrix was not initialized

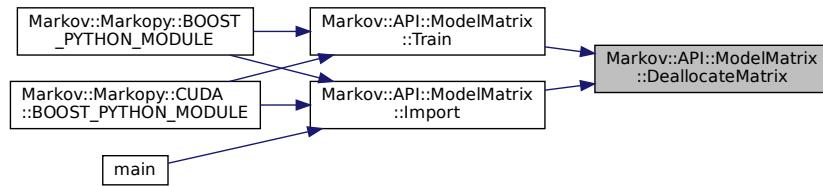
Definition at line 81 of file [modelMatrix.cpp](#).

```
00081     if(!this->ready) return false;
00082     delete[] this->matrixIndex;
00083     delete[] this->totalEdgeWeights;
00084
00085     for(int i=0;i<this->matrixSize;i++) {
00086         delete[] this->edgeMatrix[i];
00087     }
00088     delete[] this->edgeMatrix;
00089
00090     for(int i=0;i<this->matrixSize;i++) {
00091         delete[] this->valueMatrix[i];
00092     }
00093     delete[] this->valueMatrix;
00094
00095     this->matrixSize = -1;
00096     this->ready = false;
00097     return true;
00098 }
00099 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::API::ModelMatrix::ready](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the caller graph for this function:

**9.9.3.41 DumpJSON() [1/3]**

`void Markov::API::ModelMatrix::DumpJSON() [inherited]`

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

Definition at line 101 of file [modelMatrix.cpp](#).

```
00101     {
00102
00103     std::cout << "{\n      \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++) {
00105         if(this->matrixIndex[i]=='\r') std::cout << "\\\\";";
00106         else if(this->matrixIndex[i]=='\n') std::cout << "\\\\\\";";
00107         else if(this->matrixIndex[i]==0) std::cout << "\\\\\x00";
00108         else if(i==0) std::cout << "\\\\\xff";
00109         else if(this->matrixIndex[i]=='\r\n') std::cout << "\\n";
00110         else std::cout << this->matrixIndex[i];
00111     }
00112     std::cout <<
00113     "\",\n      \"edgemap\": {\n";
00114
00115     for(int i=0;i<this->matrixSize;i++) {
00116         if(this->matrixIndex[i]=='\r') std::cout << "        \\\\\\"": "[";
00117         else if(this->matrixIndex[i]=='\n') std::cout << "        \\\\\\"": "[";
00118         else if(this->matrixIndex[i]==0) std::cout << "        \\\\\x00": "[";
00119         else if(this->matrixIndex[i]<0) std::cout << "        \\\\\x0f": "[";
00120         else std::cout << "        \"": "[";
00121         for(int j=0;j<this->matrixSize;j++) {
00122             if(this->edgeMatrix[i][j]=='\r') std::cout << "          \\\\\\"";
00123             else if(this->edgeMatrix[i][j]=='\n') std::cout << "          \\\\\\"";
00124             else if(this->edgeMatrix[i][j]==0) std::cout << "          \\\\\x00";
00125             else if(this->edgeMatrix[i][j]<0) std::cout << "          \\\\\x0f";
00126             else if(this->edgeMatrix[i][j]=='\r\n') std::cout << "          \\\\\n";
```

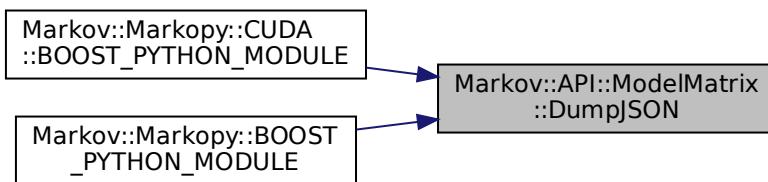
```

00128     else std::cout << "\\" << this->edgeMatrix[i][j] << "\\";
00129     if(j!=this->matrixSize-1) std::cout << ", ";
00130   }
00131   std::cout << "],\n";
00132 }
00133 std::cout << "},\n";
00134
00135 std::cout << "\" weightmap\": {\n";
00136 for(int i=0;i<this->matrixSize;i++){
00137   if(this->matrixIndex[i]=='"') std::cout << "      \"\\\\\\\"": [";
00138   else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\\\\\\\"": [";
00139   else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\\\\\\\x00\\\"": [";
00140   else if(this->matrixIndex[i]<0) std::cout << "      \"\\\\\\\\\\xff\\\"": [";
00141   else std::cout << "      \" \" << this->matrixIndex[i] << "\": [";
00142
00143   for(int j=0;j<this->matrixSize;j++){
00144     std::cout << this->valueMatrix[i][j];
00145     if(j!=this->matrixSize-1) std::cout << ", ";
00146   }
00147   std::cout << "],\n";
00148 }
00149 std::cout << " }\n}\n";
00150 }

```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the caller graph for this function:



9.9.3.42 DumpJSON() [2/3]

void Markov::API::ModelMatrix::DumpJSON () [inherited]

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

Definition at line 101 of file [modelMatrix.cpp](#).

```

00101
00102
00103   std::cout << "{\n     \"index\": \"";
00104   for(int i=0;i<this->matrixSize;i++){
00105     if(this->matrixIndex[i]=='"') std::cout << "      \"\\\\\\\"";
00106     else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\\\\\\\"";
00107     else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\\\\\\\x00\"";
00108     else if(i==0) std::cout << "      \"\\\\\\\\\\xff\"";
00109     else if(this->matrixIndex[i]=='\n') std::cout << "      \"\\n\"";
00110     else std::cout << this->matrixIndex[i];
00111   }
00112   std::cout <<
00113   "     ,\n"
00114   "     \"edgemap\": {\n";
00115
00116   for(int i=0;i<this->matrixSize;i++){
00117     if(this->matrixIndex[i]=='"') std::cout << "       \"\\\\\\\"": [";
00118     else if(this->matrixIndex[i]=='\\') std::cout << "       \"\\\\\\\\\\\"": [";
00119     else if(this->matrixIndex[i]==0) std::cout << "       \"\\\\\\\\\\x00\\\"": [";
00120     else if(this->matrixIndex[i]<0) std::cout << "       \"\\\\\\\\\\xff\\\"": [";
00121     else std::cout << "       \" \" << this->matrixIndex[i] << "\": [";
00122     for(int j=0;j<this->matrixSize;j++){
00123       if(this->edgeMatrix[i][j]=='"') std::cout << "         \"\\\\\\\"";
00124       else if(this->edgeMatrix[i][j]=='\\') std::cout << "         \"\\\\\\\\\\\"";
00125       else if(this->edgeMatrix[i][j]==0) std::cout << "         \"\\\\\\\\\\x00\"";
00126       else if(this->edgeMatrix[i][j]<0) std::cout << "         \"\\\\\\\\\\xff\"";
00127     }
00128   }
00129 }

```

```

00127         else if(this->matrixIndex[i]=='\n') std::cout << "\\\n\"";  

00128         else std::cout << "\"" << this->edgeMatrix[i][j] << "\"";  

00129         if(j!=this->matrixSize-1) std::cout << ", ";  

00130     }  

00131     std::cout << "],\n";  

00132 }  

00133 std::cout << "},\n";  

00134  

00135 std::cout << "\" weightmap\": {\n";  

00136 for(int i=0;i<this->matrixSize;i++){  

00137     if(this->matrixIndex[i]=='"') std::cout << "      \"\\\\\"\\": [";  

00138     else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\\\\\\": [";  

00139     else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\\\\\x00\\": [";  

00140     else if(this->matrixIndex[i]<0) std::cout << "      \"\\\\\\\\xff\\": [";  

00141     else std::cout << "      \" \" << this->matrixIndex[i] << "\": [";  

00142  

00143     for(int j=0;j<this->matrixSize;j++){  

00144         std::cout << this->valueMatrix[i][j];  

00145         if(j!=this->matrixSize-1) std::cout << ", ";  

00146     }  

00147     std::cout << "],\n";  

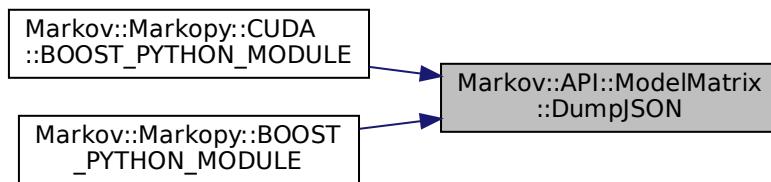
00148 }  

00149 std::cout << "  }\n}\n";
00150 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the caller graph for this function:



9.9.3.43 DumpJSON() [3/3]

void Markov::API::ModelMatrix::DumpJSON () [inherited]

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

Definition at line 101 of file [modelMatrix.cpp](#).

```

00101                                     {  

00102  

00103     std::cout << "{\n      \"index\": \"";  

00104     for(int i=0;i<this->matrixSize;i++){  

00105         if(this->matrixIndex[i]=='"') std::cout << "      \"\\\\\"\\": [";  

00106         else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\\\\\\": [";  

00107         else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\\\\\x00\"";  

00108         else if(i==0) std::cout << "      \"\\\\\\\\xff\"";  

00109         else if(this->matrixIndex[i]=='\n') std::cout << "      \"\\n\"";  

00110         else std::cout << this->matrixIndex[i];  

00111     }  

00112     std::cout <<  

00113     "\",\n      \"edgemap\": {\n";  

00114  

00115     for(int i=0;i<this->matrixSize;i++){  

00116         if(this->matrixIndex[i]=='"') std::cout << "      \"\\\\\"\\": [";  

00117         else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\\\\\\": [";  

00118         else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\\\\\x00\\": [";  

00119         else if(this->matrixIndex[i]<0) std::cout << "      \"\\\\\\\\xff\\": [";  

00120         else std::cout << "      \" \" << this->matrixIndex[i] << "\": [";  

00121         for(int j=0;j<this->matrixSize;j++){  

00122             if(this->edgeMatrix[i][j]=='"') std::cout << "        \"\\\\\"\\\"";  

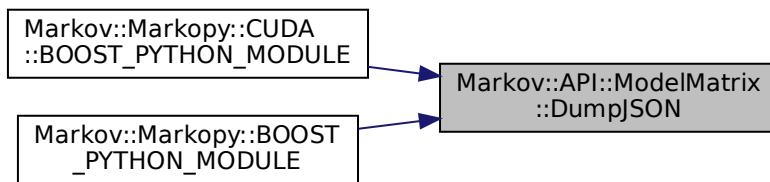
00123             else if(this->edgeMatrix[i][j]=='\\') std::cout << "        \"\\\\\\\\\\\"";  

00124             else if(this->edgeMatrix[i][j]==0) std::cout << "        \"\\\\\\\\x00\\\"";
00125         }
```

References `Markov::API::ModelMatrix::edgeMatrix`, `Markov::API::ModelMatrix::matrixIndex`, `Markov::API::ModelMatrix::matrixSize`, and `Markov::API::ModelMatrix::valueMatrix`.

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the caller graph for this function:



9.9.3.44 Edges() [1/4]

```
std::vector<Edge<char>*>* Markov::Model<char>::Edges() [inline], [inherited]  
Return a vector of all the edges in the model.
```

Returns

vector of edges

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.9.3.45 Edges() [2/4]

`std::vector<Edge<char>*>* Markov::Model<char>::Edges() [inline], [inherited]`
Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.9.3.46 Edges() [3/4]

```
std::vector<Edge<char>*>* Markov::Model< char >::Edges () [inline], [inherited]
Return a vector of all the edges in the model.
```

Returns

vector of edges

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.9.3.47 Edges() [4/4]

```
std::vector<Edge<char>*>* Markov::Model< char >::Edges () [inline], [inherited]
Return a vector of all the edges in the model.
```

Returns

vector of edges

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.9.3.48 Export() [1/9]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
Open a file to export with filename, and call bool Model::Export with std::ofstream.
```

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.9.3.49 Export() [2/9]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
Open a file to export with filename, and call bool Model::Export with std::ofstream.
```

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.9.3.50 Export() [3/9]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
```

```
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.9.3.51 Export() [4/9]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
```

```
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.9.3.52 export() [1/6]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

Definition at line 138 of file [base.py](#).

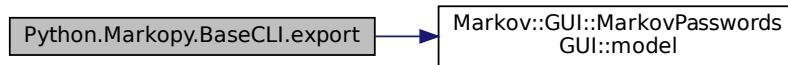
```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """
00143         self.model.Export(filename)
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.mo](#)

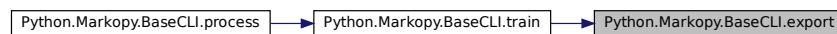
[Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.53 export() [2/6]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

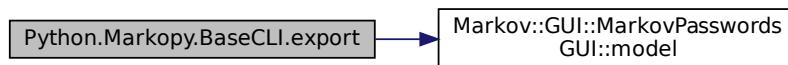
Definition at line 138 of file [base.py](#).

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142             """
00143         self.model.Export(filename)
00144 
```

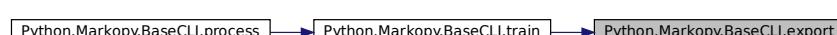
References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.54 `export()` [3/6]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<code>filename</code>	filename to export to
-----------------------	-----------------------

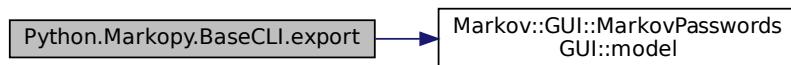
Definition at line 138 of file [base.py](#).

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142             """
00143         self.model.Export(filename)
00144
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.55 `export()` [4/6]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<code>filename</code>	filename to export to
-----------------------	-----------------------

Definition at line 138 of file [base.py](#).

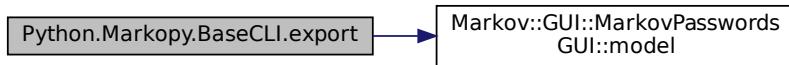
```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142             """
00143         self.model.Export(filename)
```

00144

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.56 export() [5/6]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

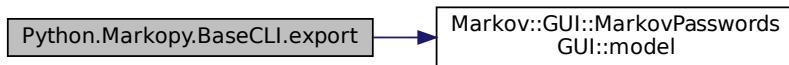
Definition at line 138 of file [base.py](#).

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.57 export() [6/6]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<code>filename</code>	filename to export to
-----------------------	-----------------------

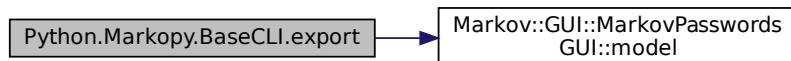
Definition at line 138 of file [base.py](#).

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144 
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.mo](#)
[Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.58 Export() [5/9]

```
bool Markov::Model< char >::Export (
    std::ofstream * f ) [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight:right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file model.h.

```
00288     {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <
00294         "\n";
00295     }
00296     return true;
00297 }
```

9.9.3.59 Export() [6/9]

```
bool Markov::Model< char >::Export (
    std::ofstream * f) [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file model.h.

```
00288     {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <
00294         "\n";
00295     }
00296     return true;
00297 }
```

9.9.3.60 Export() [7/9]

```
bool Markov::Model< char >::Export (
    std::ofstream * f) [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file model.h.

```

00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
00294         "\n";
00295     }
00296     return true;
00297 }
```

9.9.3.61 Export() [8/9]

```
bool Markov::Model< char >::Export (
    std::ofstream * f) [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file [model.h](#).

```

00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
00294         "\n";
00295     }
00296     return true;
00297 }
```

9.9.3.62 Export() [9/9]

```
def Python.Markopy.MarkovModel.Export (
    str filename) [inherited]
```

Definition at line 26 of file [mm.py](#).

```
00026     def Export(filename : str):
00027         pass
00028
```

9.9.3.63 FastRandomWalk() [1/9]

```
def Python.Markopy.ModelMatrix.FastRandomWalk (
    int count,
    str wordlist,
    int minlen,
    int maxlen) [inherited]
```

Definition at line 48 of file [mm.py](#).

```
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

Referenced by [Python.Markopy.ModelMatrixCLI._generate\(\)](#).

Here is the caller graph for this function:



9.9.3.64 FastRandomWalk() [2/9]

```
def Python.Markopy.ModelMatrix.FastRandomWalk (
    int count,
    str wordlist,
    int minlen,
    int maxlen ) [inherited]
```

Definition at line 48 of file mm.py.

```
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

Referenced by [Python.Markopy.ModelMatrixCLI._generate\(\)](#).

Here is the caller graph for this function:



9.9.3.65 FastRandomWalk() [3/9]

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen,
    int maxLen,
    bool bFileIO,
    bool bInfinite ) [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

TODO

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

Definition at line 58 of file [cudaModelMatrix.cu](#).

```
00058
00059     cudaDeviceProp prop;
00060     int device=0;
00061     cudaGetDeviceProperties(&prop, device);
00062     cudaChooseDevice(&device, &prop);
00063     //std::cout << "Flattening matrix." << std::endl;
```

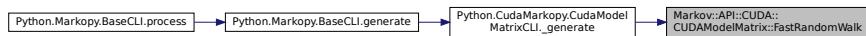
```

00064     this->FlattenMatrix();
00065     //std::cout << "Migrating matrix." << std::endl;
00066     this->MigrateMatrix();
00067     //std::cout << "Migrated matrix." << std::endl;
00068     std::ofstream wordlist;
00069     if(bFileIO)
00070         wordlist.open(wordlistFileName);
00071
00072
00073     cudaBlocks = 1024;
00074     cudaThreads = 256;
00075     iterationsPerKernelThread = 100;
00076     alternatingKernels = 2;
00077     totalOutputPerKernel = (long int)cudaBlocks*(long int)cudaThreads*iterationsPerKernelThread;
00078     totalOutputPerSync= totalOutputPerKernel*alternatingKernels;
00079     numberofPartitions = n/totalOutputPerSync;
00080     cudaGridSize = cudaBlocks*cudaThreads;
00081     cudaMemPerGrid = (maxLen+2)*iterationsPerKernelThread;
00082     cudaPerKernelAllocationSize = cudaGridSize*cudaMemPerGrid;
00083     this->prepKernelMemoryChannel(alternatingKernels);
00084
00085     unsigned long int leftover = n - (totalOutputPerSync*numberofPartitions);
00086
00087     if(bInfinite && !numberofPartitions) numberofPartitions=5;
00088     std::cerr << cudaPerKernelAllocationSize << "\n";
00089
00090     if(n%totalOutputPerSync) std::cerr << "For optimization, request outputs muliples of "<<
00091     totalOutputPerSync << ".\n";
00092
00093     //start kernelID 1
00094     this->LaunchAsyncKernel(1, minLen, maxLen);
00095
00096     for(int i=1;i<numberofPartitions;i++) {
00097         if(bInfinite) i=0;
00098
00099         //wait kernelID1 to finish, and start kernelID 0
00100         cudaStreamSynchronize(this->cudastreams[1]);
00101         this->LaunchAsyncKernel(0, minLen, maxLen);
00102
00103         //start memcpy from kernel 1 (block until done)
00104         this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00105
00106         //wait kernelID 0 to finish, then start kernelID1
00107         cudaStreamSynchronize(this->cudastreams[0]);
00108         this->LaunchAsyncKernel(1, minLen, maxLen);
00109
00110         //start memcpy from kernel 0 (block until done)
00111         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00112     }
00113
00114     //wait kernelID1 to finish, and start kernelID 0
00115     cudaStreamSynchronize(this->cudastreams[1]);
00116     this->LaunchAsyncKernel(0, minLen, maxLen);
00117     this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00118     cudaStreamSynchronize(this->cudastreams[0]);
00119     this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00120
00121
00122     if(!leftover) return;
00123     alternatingKernels=1;
00124     std::cerr << "Remaining line count (" << leftover << ") is lower than partition. Adjusting CUDA
workload..\n";
00125     this->iterationsPerKernelThread = leftover/cudaGridSize;
00126     this->LaunchAsyncKernel(0, minLen, maxLen);
00127     cudaStreamSynchronize(this->cudastreams[0]);
00128     this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00129
00130     leftover -= this->iterationsPerKernelThread*cudaGridSize;
00131     if(!leftover) return;
00132
00133     std::cerr << "Remaining line count (" << leftover << ") is lower than minimum possible. Handing
over to CPU generation.\n";
00134     this->iterationsPerKernelThread = leftover/cudaGridSize;
00135
00136     leftover -= this->iterationsPerKernelThread;
00137
00138     if(!leftover) return;
00139     std::cerr << "Remaining " << leftover << " lines are absolutely not worth printing.\n";
00140     Markov::API::ModelMatrix::ConstructMatrix();
00141     Markov::API::ModelMatrix::FastRandomWalk(leftover, &wordlist, minLen, maxLen, 1, bFileIO);
00142
00143 }
```

References [Markov::API::CUDA::CUDAModelMatrix::alternatingKernels](#), [Markov::API::CUDA::CUDAModelMatrix::cudaBlocks](#), [Markov::API::CUDA::CUDAModelMatrix::cudaGridSize](#), [Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid](#), [Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize](#), [Markov::API::CUDA::CUDAModelMatrix::cudaThreads](#),

`Markov::API::CUDA::CUDAModelMatrix::iterationsPerKernelThread`, `Markov::API::CUDA::CUDAModelMatrix::numberOfPartitions`, `Markov::API::CUDA::CUDAModelMatrix::totalOutputPerKernel`, and `Markov::API::CUDA::CUDAModelMatrix::totalOutputPerSync`. Referenced by `Python.CudaMarkopy.CudaModelMatrixCLI::_generate()`.

Here is the caller graph for this function:



9.9.3.66 FastRandomWalk() [4/9]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [inherited]
```

Random walk on the Matrix-reduced `Markov::Model`.

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using `Markov::API::ModelMatrix::FastRandomWalkPartition`.

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces `Markov::API::MarkovPasswords::Generate` runtime by %96.5

This function has deprecated `Markov::API::MarkovPasswords::Generate`, and will eventually replace it.

Parameters

<code>n</code>	- Number of passwords to generate.
<code>wordlistFileName</code>	- Filename to write to
<code>minLen</code>	- Minimum password length to generate
<code>maxLen</code>	- Maximum password length to generate
<code>threads</code>	- number of OS threads to spawn
<code>bFileIO</code>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

Definition at line 217 of file `modelMatrix.cpp`.

```
00217
{
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

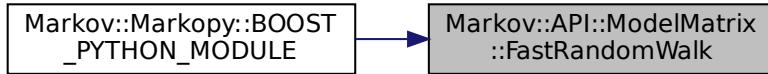
References `Markov::API::ModelMatrix::FastRandomWalk()`.

Referenced by `Markov::Markopy::BOOST_PYTHON_MODULE()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.67 FastRandomWalk() [5/9]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

Definition at line 217 of file [modelMatrix.cpp](#).

```
00217
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

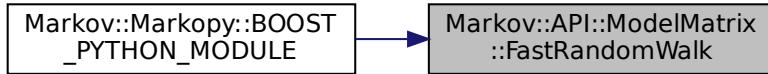
References [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.68 FastRandomWalk() [6/9]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

Definition at line 217 of file [modelMatrix.cpp](#).

```
00217
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

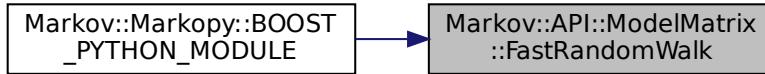
References [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.69 FastRandomWalk() [7/9]

```

int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [protected], [inherited]

```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by 996.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```

Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);

```

Definition at line 204 of file [modelMatrix.cpp](#).

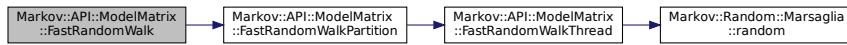
```

00204
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
00209     threads);
00210     else{
00211         int numberOfPartitions = n/50000000ull;
00212         for(int i=0;i<numberOfPartitions;i++)
00213             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
00214             threads);
00215     }
00216     return 0;
00217 }
```

References [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.70 FastRandomWalk() [8/9]

```

int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [protected], [inherited]
  
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```

Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
  
```

Definition at line 204 of file [modelMatrix.cpp](#).

```

00204
00205
00206
00207      std::mutex mlock;
00208      if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
00209      threads);
00210      else{
00211          int numberOfPartitions = n/50000000ull;
00212          for(int i=0;i<numberOfPartitions;i++)
00213              this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
00214          threads);
00215      }
  
```

```
00214     return 0;
00215 }
```

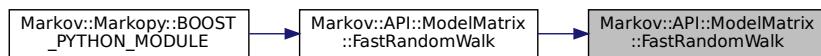
References [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.71 FastRandomWalk() [9/9]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [protected], [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

Definition at line 204 of file [modelMatrix.cpp](#).

```
00204
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
threads);
```

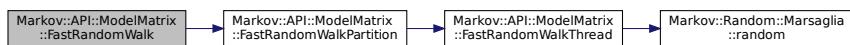
```

00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
00213             threads);
00214     }
00215 }
```

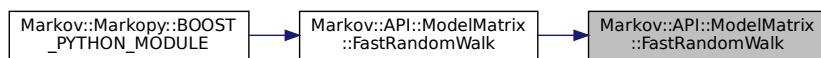
References [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.72 FastRandomWalkPartition() [1/3]

```

void Markov::API::ModelMatrix::FastRandomWalkPartition (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    bool bFileIO,
    int threads ) [protected], [inherited]
```

A single partition of FastRandomWalk event.

Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation
- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 225 of file [modelMatrix.cpp](#).

```

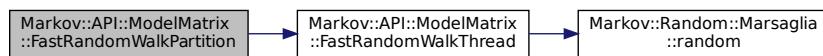
00225
00226     int iterationsPerThread = n/threads;
00227     int iterationsPerThreadCarryOver = n%threads;
00228
00229     std::vector<std::thread*> threadsV;
00230
00231     int id = 0;
00232     for(int i=0;i<threads;i++) {
00233         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00234                                         mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00239                                         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00240
00241     for(int i=0;i<threads;i++) {
00242         threadsV[i]->join();
00243     }

```

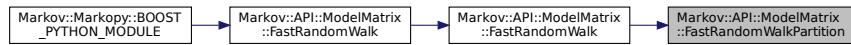
References [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.73 FastRandomWalkPartition() [2/3]

```

void Markov::API::ModelMatrix::FastRandomWalkPartition (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    bool bFileIO,
    int threads ) [protected], [inherited]

```

A single partition of FastRandomWalk event.

Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation
- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 225 of file [modelMatrix.cpp](#).

```

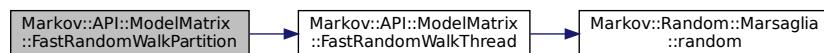
00225
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++) {
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00235                                         mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00236         id++;
00237     }
00238
00239     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00240                                         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00241     for(int i=0;i<threads;i++) {
00242         threadsV[i]->join();
00243     }

```

References [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.74 FastRandomWalkPartition() [3/3]

```

void Markov::API::ModelMatrix::FastRandomWalkPartition (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    bool bFileIO,
    int threads ) [protected], [inherited]

```

A single partition of FastRandomWalk event.

Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation
- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 225 of file [modelMatrix.cpp](#).

```

00225
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++) {
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00235                                         mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00236         id++;
00237     }
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00239                                         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00240     for(int i=0;i<threads;i++) {
00241         threadsV[i]->join();
00242     }
00243 }
```

References [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.75 FastRandomWalkThread() [1/3]

```
void Markov::API::ModelMatrix::FastRandomWalkThread (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    int id,
    bool bFileIO ) [protected], [inherited]
```

A single thread of a single partition of FastRandomWalk.

A FastRandomWalkPartition will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>id</i>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

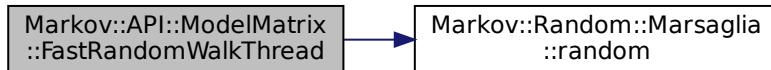
Definition at line 153 of file [modelMatrix.cpp](#).

```
00153
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179             if (len >= maxLen) break;
00180             else if ((next < 0) && (len < minLen)) continue;
00181             else if (next < 0) break;
00182             cur = next;
00183             res[bufferctr + len++] = cur;
00184         }
00185     }
00186     res[bufferctr + len++] = '\n';
00187     bufferctr+=len;
00188
00189 }
00190 if(bFileIO){
00191     mlock->lock();
00192     *wordlist << res;
00193     mlock->unlock();
00194 }else{
00195     mlock->lock();
00196     std::cout << res;
00197     mlock->unlock();
00198 }
```

```
00199     delete res;
00200
00201 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::Random::Marsaglia::random\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#). Referenced by [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.76 FastRandomWalkThread() [2/3]

```
void Markov::API::ModelMatrix::FastRandomWalkThread (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    int id,
    bool bFileIO ) [protected], [inherited]
```

A single thread of a single partition of [FastRandomWalk](#).

A [FastRandomWalkPartition](#) will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>id</i>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 153 of file [modelMatrix.cpp](#).

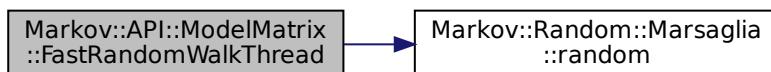
```
00153
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
```

```

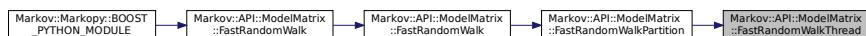
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179             if (len >= maxLen) break;
00180             else if ((next < 0) && (len < minLen)) continue;
00181             else if (next < 0) break;
00182             cur = next;
00183             res[bufferctr + len++] = cur;
00184         }
00185         res[bufferctr + len++] = '\n';
00186         bufferctr+=len;
00187     }
00188 }
00189 if(bFileIO){
00190     mlock->lock();
00191     *wordlist << res;
00192     mlock->unlock();
00193 }else{
00194     mlock->lock();
00195     std::cout << res;
00196     mlock->unlock();
00197 }
00198 }
00199 delete res;
00200
00201 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::Random::Marsaglia::random\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#). Referenced by [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.77 FastRandomWalkThread() [3/3]

```

void Markov::API::ModelMatrix::FastRandomWalkThread (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
```

```

    int maxLen,
    int id,
    bool bFileIO ) [protected], [inherited]

```

A single thread of a single partition of FastRandomWalk.

A FastRandomWalkPartition will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>id</i>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 153 of file [modelMatrix.cpp](#).

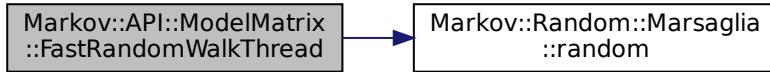
```

00153
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++) {
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0) {
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179             if (len >= maxLen) break;
00180             else if ((next < 0) && (len < minLen)) continue;
00181             else if (next < 0) break;
00182             cur = next;
00183             res[bufferctr + len++] = cur;
00184         }
00185         res[bufferctr + len++] = '\n';
00186         bufferctr+=len;
00187
00188     }
00189
00190     if(bFileIO){
00191         mlock->lock();
00192         *wordlist << res;
00193         mlock->unlock();
00194     }else{
00195         mlock->lock();
00196         std::cout << res;
00197         mlock->unlock();
00198     }
00199     delete res;
00200 }

```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::Random::Marsaglia::random\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#). Referenced by [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.78 FlattenMatrix()

`__host__ void Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix () [inherited]`
Flatten migrated matrix from 2d to 1d.

Definition at line 261 of file [cudaModelMatrix.cu](#).

```

00261
00262     this->flatEdgeMatrix = new char[this->matrixSize*this->matrixSize];
00263
00264     this->flatValueMatrix = new long int[this->matrixSize*this->matrixSize];
00265     for(int i=0;i<this->matrixSize;i++){
00266         memcpy(&this->flatEdgeMatrix[i*this->matrixSize], this->edgeMatrix[i], this->matrixSize );
00267         memcpy(this->flatValueMatrix[i*this->matrixSize], this->valueMatrix[i],
00268                this->matrixSize*sizeof(long int) );
00269     }

```

References [Markov::API::CUDA::CUDAModelMatrix::flatEdgeMatrix](#), [Markov::API::CUDA::CUDAModelMatrix::flatValueMatrix](#), and [Markov::API::ModelMatrix::matrixSize](#).

9.9.3.79 GatherAsyncKernelOutput()

`__host__ void Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput (`
 `int kernelID,`
 `bool bFileIO,`
 `std::ofstream & wordlist) [protected], [inherited]`

Definition at line 180 of file [cudaModelMatrix.cu](#).

```

00180
00181 {
00182     cudaMemcpy(this->outputBuffer[kernelID],this->device_outputBuffer[kernelID],cudaPerKernelAllocationSize,
00183     cudaMemcpyDeviceToHost);
00184     //std::cerr << "Kernel" << kernelID << " output copied\n";
00185     if(bFileIO){
00186         for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00187             wordlist << &this->outputBuffer[kernelID][j];
00188         }
00189     }else{
00190         for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00191             std::cout << &this->outputBuffer[kernelID][j];
00192         }
00193     }

```

References [Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid](#), [Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize](#), and [Markov::API::CUDA::CUDAModelMatrix::outputBuffer](#).

9.9.3.80 Generate() [1/5]

```
def Python.Markopy.MarkovModel.Generate (
    int count,
    str wordlist,
    int minlen,
    int maxlen,
    int threads ) [inherited]
```

Definition at line 34 of file [mm.py](#).

```
00034     def Generate(count : int, wordlist : str, minlen : int, maxlen: int, threads : int):
00035         pass
00036
00037
```

Referenced by [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#).

Here is the caller graph for this function:



9.9.3.81 generate() [1/6]

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

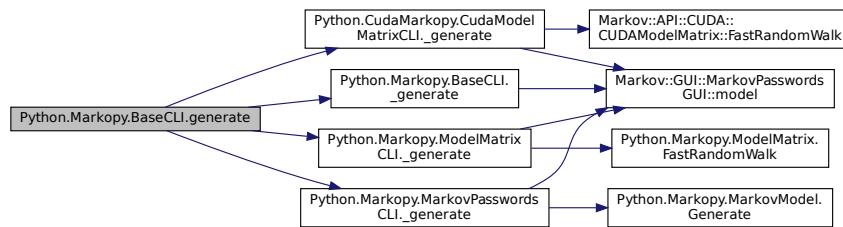
Definition at line 145 of file [base.py](#).

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156
00157         if(bulk and os.path.isfile(wordlist)):
00158             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00159             self._generate(wordlist)
```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopy](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.82 generate() [2/6]

```

def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
  
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

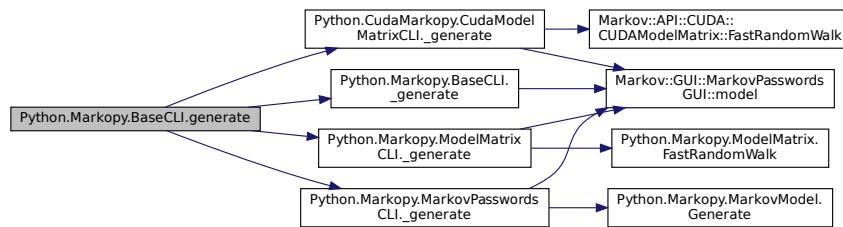
```

00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159
  
```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI._generate\(\)](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.83 generate() [3/6]

```

def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
  
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

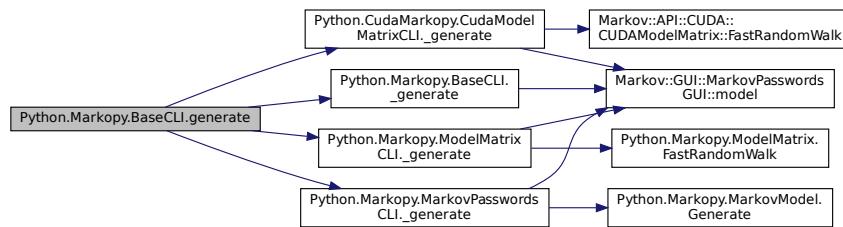
```

00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159
  
```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopyBaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.84 generate() [4/6]

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

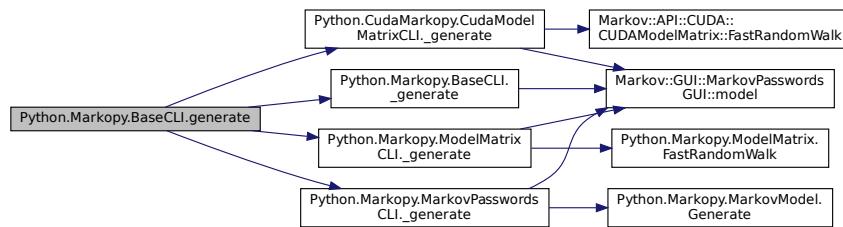
Definition at line 145 of file [base.py](#).

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159
```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI._generate\(\)](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.85 generate() [5/6]

```

def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
  
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

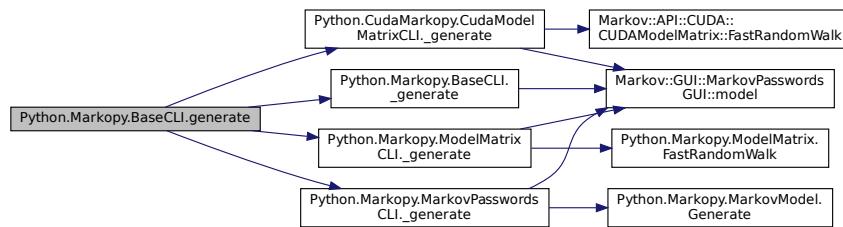
```

00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159
  
```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopyBaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.86 generate() [6/6]

```

def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
  
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

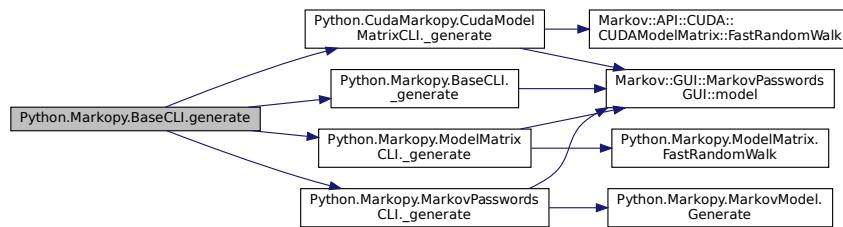
```

00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159
  
```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI._generate\(\)](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.87 Generate() [2/5]

```

void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]
  
```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

```

00118
00119     {
00120     char* res;
00121     char print[100];
00122     std::ofstream wordlist;
00123     wordlist.open(wordlistFileName);
00124     std::mutex mlock;
00125     int iterationsPerThread = n/threads;
00126     int iterationsCarryOver = n%threads;
00127     std::vector<std::thread*> threadsV;
00128     for(int i=0;i<threads;i++) {
00129         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00130                                     &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
  
```

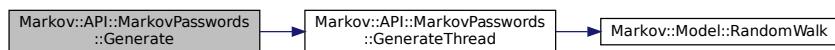
```

00129     }
00130
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

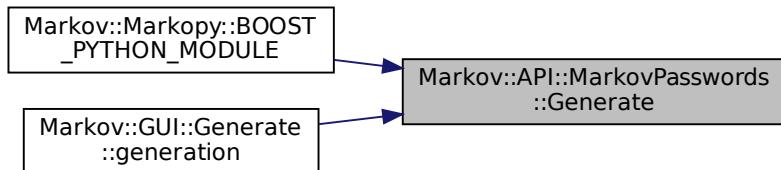
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.88 Generate() [3/5]

```
void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]
```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

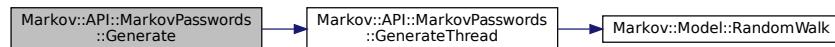
```

00118
00119     {
00120     char* res;
00121     char print[100];
00122     std::ofstream wordlist;
00123     wordlist.open(wordlistFileName);
00124     std::mutex mlock;
00125     int iterationsPerThread = n/threads;
00126     int iterationsCarryOver = n%threads;
00127     std::vector<std::thread*> threadsV;
00128     for(int i=0;i<threads;i++) {
00129         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00130                                     &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00131     }
00132     for(int i=0;i<threads;i++) {
00133         threadsV[i]->join();
00134         delete threadsV[i];
00135     }
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

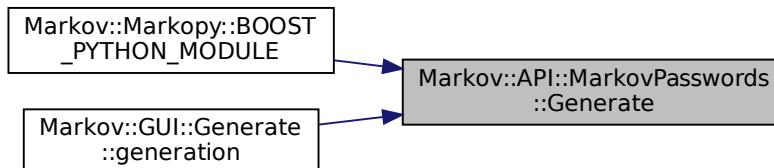
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.89 Generate() [4/5]

```

void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]
```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. [FastRandomWalk](#) will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

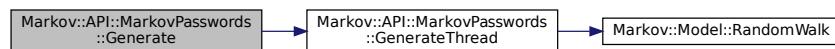
```

00118
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++) {
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129                                         &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130     }
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00136
00137
00138 }
```

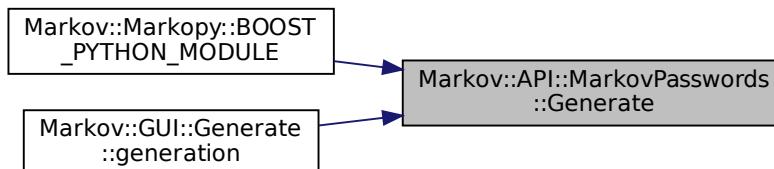
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.90 Generate() [5/5]

```

void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
```

```
int maxLen = 12,
int threads = 20 ) [inherited]
```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

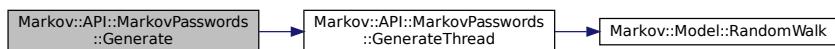
Definition at line 118 of file [markovPasswords.cpp](#).

```
00118     {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++) {
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129             &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130     }
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00136 }
00137
00138 }
```

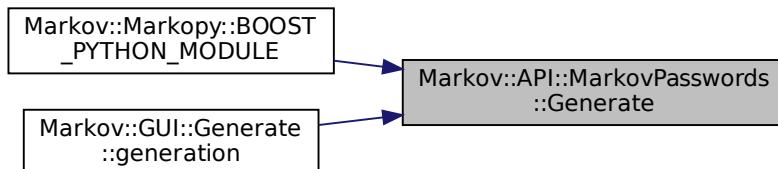
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.91 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
    std::mutex * outputLock,
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen,
    int maxLen ) [private], [inherited]
```

A single thread invoked by the Generate function.

DEPRECATED: See `Markov::API::MatrixModel::FastRandomWalkThread` for more information. This has been replaced with a much more performance-optimized method. `FastRandomWalk` will reduce the runtime by %96.5 on average.

Parameters

<code>outputLock</code>	- shared mutex lock to lock during output operation. Prevents race condition on write.
<code>n</code>	number of lines to be generated by this thread
<code>wordlist</code>	wordlistfile
<code>minLen</code>	- Minimum password length to generate
<code>maxLen</code>	- Maximum password length to generate

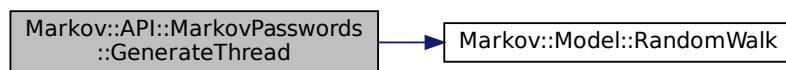
Definition at line 140 of file `markovPasswords.cpp`.

```
00140
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist << res << "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

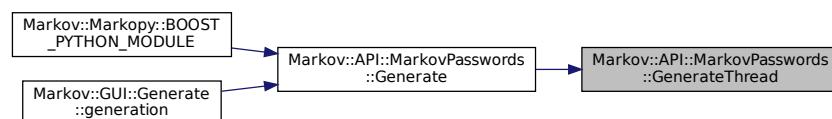
References `Markov::Model< NodeStorageType >::RandomWalk()`.

Referenced by `Markov::API::MarkovPasswords::Generate()`.

Here is the call graph for this function:



Here is the caller graph for this function:



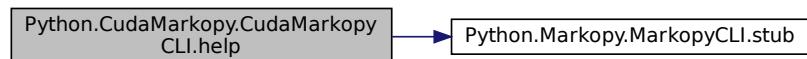
9.9.3.92 help()

```
def Python.CudaMarkopy.CudaMarkopyCLI.help (
    self )
overload help function to print submodel helps
Reimplemented from Python.Markopy.MarkopyCLI.
Definition at line 61 of file cudamarkopy.py.
00061     def help(self):
00062         """! @brief overload help string"""
00063         self.parser.print_help = self.stub
00064         self.args = self.parser.parse_known_args()[0]
00065         if(self.args.model_type!="_MMX"):
00066             if(self.args.model_type=="MP"):
00067                 mp = MarkovPasswordsCLI()
00068                 mp.add_arguments()
00069                 mp.parser.print_help()
00070             elif(self.args.model_type=="MMX"):
00071                 mp = ModelMatrixCLI()
00072                 mp.add_arguments()
00073                 mp.parser.print_help()
00074             elif(self.args.model_type == "CUDA"):
00075                 mp = CudaModelMatrixCLI()
00076                 mp.add_arguments()
00077                 mp.parser.print_help()
00078         else:
00079             print(colored("Model Mode selection choices:", "green"))
00080             self.print_help()
00081             print(colored("Following are applicable for -mt MP mode:", "green"))
00082             mp = MarkovPasswordsCLI()
00083             mp.add_arguments()
00084             mp.parser.print_help()
00085             print(colored("Following are applicable for -mt MMX mode:", "green"))
00086             mp = ModelMatrixCLI()
00087             mp.add_arguments()
00088             mp.parser.print_help()
00089             print(colored("Following are applicable for -mt CUDA mode:", "green"))
00090             mp = CudaModelMatrixCLI()
00091             mp.add_arguments()
00092             mp.parser.print_help()
00093         exit()
00094
```

References [Python.Markopy.BaseCLI.parser](#), and [Python.Markopy.MarkopyCLI.stub\(\)](#).

Referenced by [Python.Markopy.MarkopyCLI.add_arguments\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.93 Import() [1/8]

```
void Markov::API::ModelMatrix::Import (
    const char * filename ) [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.

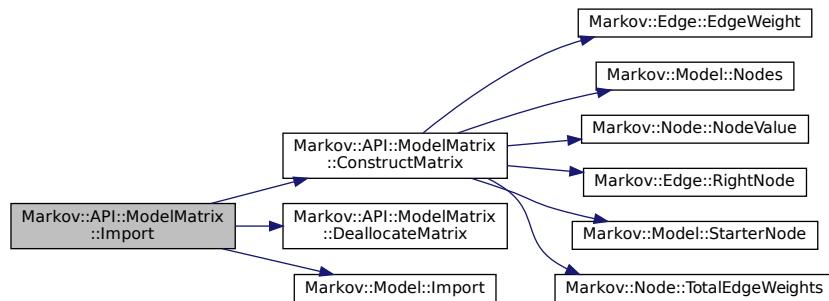
Definition at line 19 of file [modelMatrix.cpp](#).

```
00019
00020     this->DeallocateMatrix\(\);
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix\(\);
00023 }
```

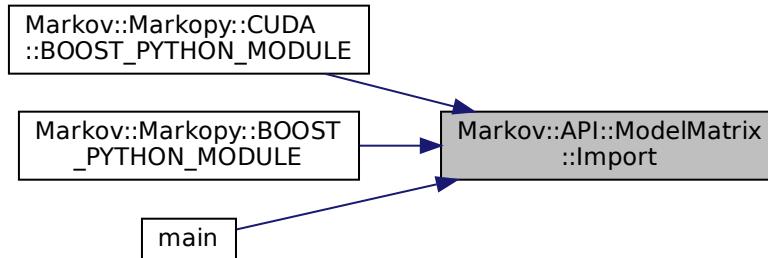
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.9.3.94 Import() [2/8]**

```
void Markov::API::ModelMatrix::Import (
    const char * filename ) [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.

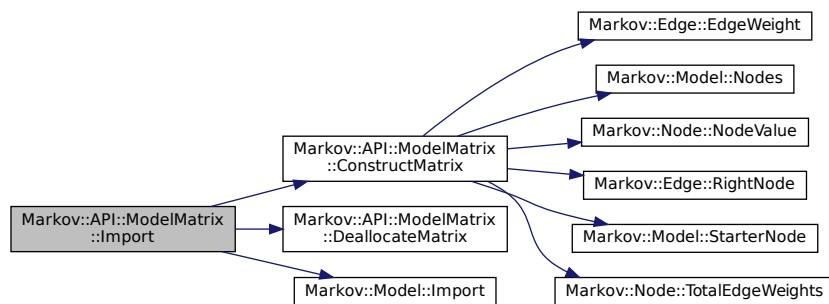
Definition at line 19 of file [modelMatrix.cpp](#).

```
00019
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

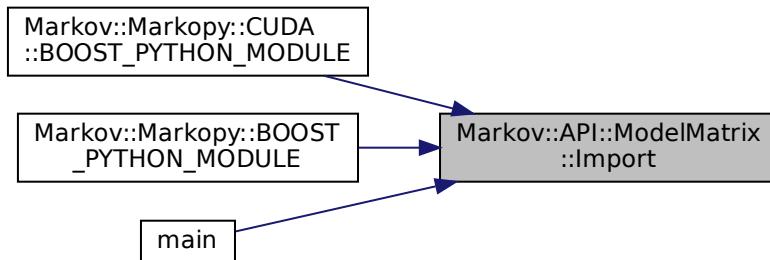
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.9.3.95 Import() [3/8]**

```
void Markov::API::ModelMatrix::Import (
    const char * filename ) [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.

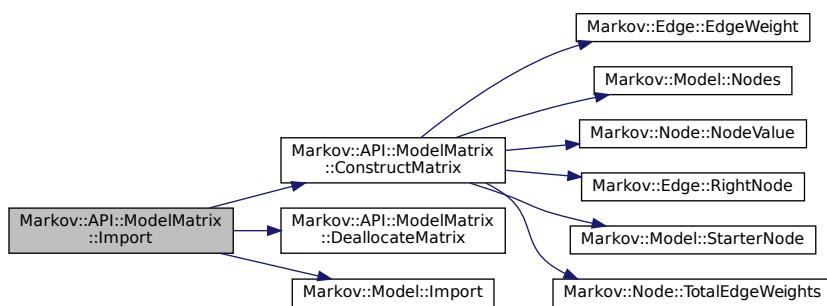
Definition at line 19 of file [modelMatrix.cpp](#).

```
00019
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

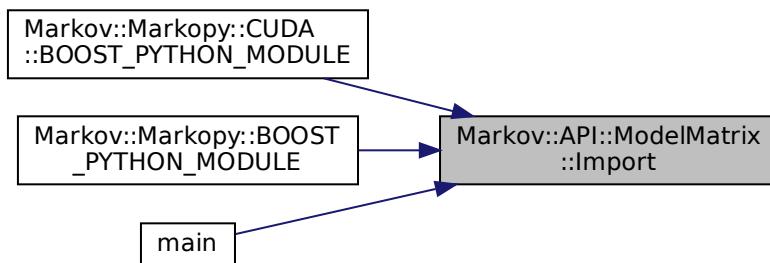
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.9.3.96 Import() [4/8]**

```
bool Markov::Model< char >::Import (
    std::ifstream * f )  [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);

Definition at line 126 of file model.h.
00216     std::string cell;                                {
00217     std::string cell;
00218     char src;
00219     char target;
00220     long int oc;
00221
00222     while (std::getline(*f, cell)) {
00223         //std::cout << "cell: " << cell << std::endl;
00224         src = cell[0];
00225         target = cell[cell.length() - 1];
00226         char* j;
00227         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00228         //std::cout << "oc << "\n";
00229         Markov::Node<NodeStorageType>* srcN;
00230         Markov::Node<NodeStorageType>* targetN;
00231         Markov::Edge<NodeStorageType>* e;
00232         if (this->nodes.find(src) == this->nodes.end()) {
00233             srcN = new Markov::Node<NodeStorageType>(src);
00234             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00235             //std::cout << "Creating new node at start.\n";
00236         }
00237         else {
00238             srcN = this->nodes.find(src)->second;
00239         }
00240
00241         if (this->nodes.find(target) == this->nodes.end()) {
00242             targetN = new Markov::Node<NodeStorageType>(target);
00243             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00244             //std::cout << "Creating new node at end.\n";
00245         }
00246         else {
00247             targetN = this->nodes.find(target)->second;
00248         }
00249         e = srcN->Link(targetN);
00250         e->AdjustEdge(oc);
00251         this->edges.push_back(e);
00252
00253         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00254         int(targetN->NodeValue()) << "\n";
00255
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.9.3.97 Import() [5/8]

```
bool Markov::Model< char >::Import (
    std::ifstream * f )  [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file model.h.

```
00216     std::string cell;                                {
00217     std::string cell;
00218
```

```

00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout << "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00255         int(targetN->NodeValue()) << "\n";
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.9.3.98 Import() [6/8]

```
bool Markov::Model< char >::Import (
    std::ifstream * f )  [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file [model.h](#).

```

00216
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
```

```

00232     Markov::Edge<NodeStorageType>* e;
00233     if (this->nodes.find(src) == this->nodes.end()) {
00234         srcN = new Markov::Node<NodeStorageType>(src);
00235         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00236         //std::cout << "Creating new node at start.\n";
00237     }
00238     else {
00239         srcN = this->nodes.find(src)->second;
00240     }
00241
00242     if (this->nodes.find(target) == this->nodes.end()) {
00243         targetN = new Markov::Node<NodeStorageType>(target);
00244         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00245         //std::cout << "Creating new node at end.\n";
00246     }
00247     else {
00248         targetN = this->nodes.find(target)->second;
00249     }
00250     e = srcN->Link(targetN);
00251     e->AdjustEdge(oc);
00252     this->edges.push_back(e);
00253
00254     //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "-->" <<
00255     int(targetN->NodeValue()) << "\n";
00256
00257 }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.9.3.99 Import() [7/8]

```
bool Markov::Model< char >::Import (
    std::ifstream * f )  [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file [model.h](#).

```

00216     std::string cell;
00217
00218     char src;
00219     char target;
00220     long int oc;
00221
00222     while (std::getline(*f, cell)) {
00223         //std::cout << "cell: " << cell << std::endl;
00224         src = cell[0];
00225         target = cell[cell.length() - 1];
00226         char* j;
00227         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00228         //std::cout << oc << "\n";
00229         Markov::Node<NodeStorageType>* srcN;
00230         Markov::Node<NodeStorageType>* targetN;
00231         Markov::Edge<NodeStorageType>* e;
00232
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00245         }
00246     }
00247 }
```

```

00245         //std::cout << "Creating new node at end.\n";
00246     }
00247     else {
00248         targetN = this->nodes.find(target)->second;
00249     }
00250     e = srcN->Link(targetN);
00251     e->AdjustEdge(oc);
00252     this->edges.push_back(e);
00253
00254     //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00255     int(targetN->NodeValue()) << "\n";
00256
00257 }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.9.3.100 Import() [8/8]

```
def Python.Markopy.MarkovModel.Import (
    str filename ) [inherited]
Definition at line 22 of file mm.py.
00022     def Import(filename : str):
00023         pass
00024
```

9.9.3.101 import_model() [1/6]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
Import a model file.
```

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

Definition at line 77 of file base.py.

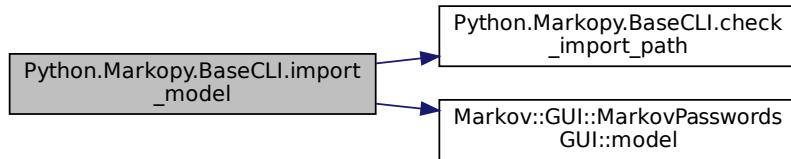
```

00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
00086             directory")
00087             return False
00088
00089         self.model.Import(filename)
00090         logging pprint("Model imported successfully.", 2)
00091
00092
00093
```

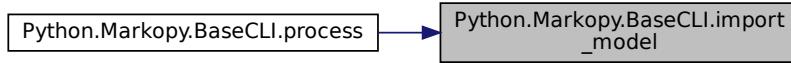
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.102 import_model() [2/6]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

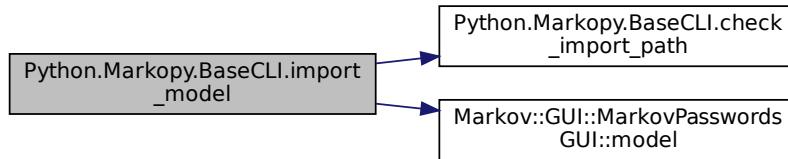
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

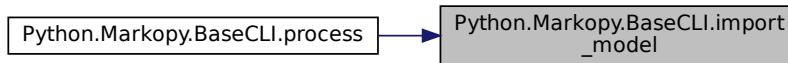
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model\(\)](#), [Python.Markopy.BaseCLI.model\(\)](#), [Python.Markopy.ModelMatrixCLI.model\(\)](#), [Python.Markopy.MarkovPasswordsCLI.model\(\)](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.103 import_model() [3/6]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

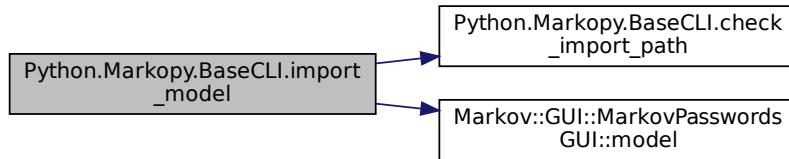
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

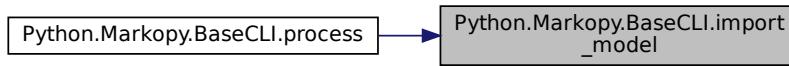
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model\(\)](#), [Python.Markopy.BaseCLI.model\(\)](#), [Python.Markopy.ModelMatrixCLI.model\(\)](#), [Python.Markopy.MarkovPasswordsCLI.model\(\)](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.104 import_model() [4/6]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

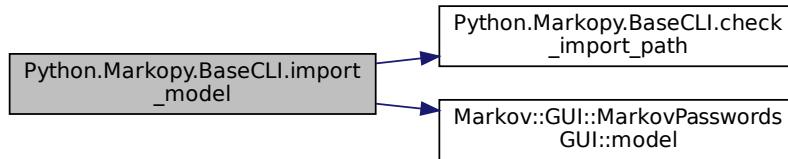
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

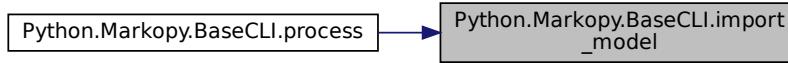
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.105 import_model() [5/6]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

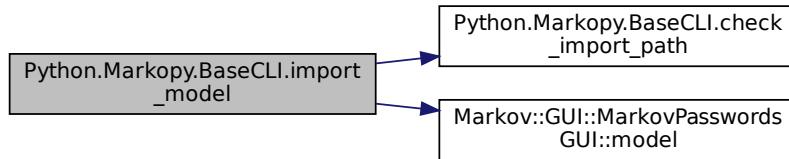
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

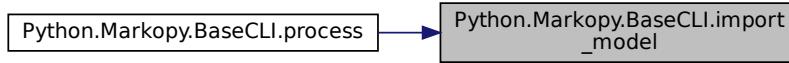
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.106 import_model() [6/6]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

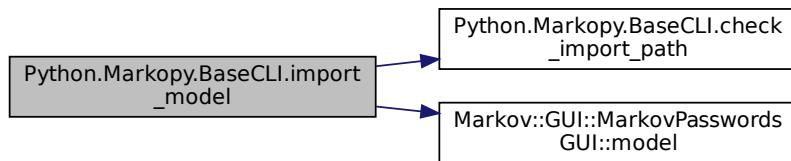
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

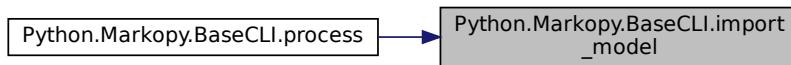
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.107 init_post_arguments() [1/2]

```
def Python.CudaMarkopy.CudaModelMatrixCLI.init_post_arguments (
    self ) [inherited]
```

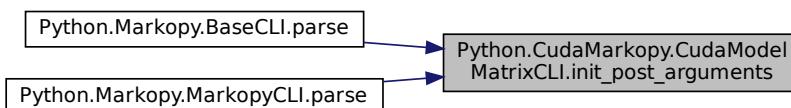
Reimplemented from [Python.Markopy.ModelMatrixCLI](#).

Definition at line 71 of file [cudammx.py](#).

```
00071     def init_post_arguments(self):
00072         super().init_post_arguments()
00073         self.bInfinite = self.args.infinite
00074
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.9.3.108 init_post_arguments() [2/2]

```
def Python.Markopy.MarkopyCLI.init_post_arguments (
    self ) [inherited]
```

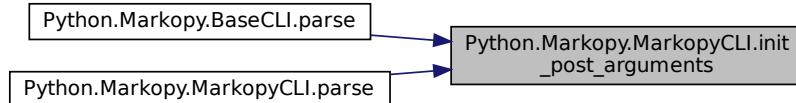
Reimplemented from [Python.Markopy.BaseCLI](#).

Definition at line 143 of file [markopy.py](#).

```
00143     def init_post_arguments(self):
00144         pass
00145
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.9.3.109 LaunchAsyncKernel()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::LaunchAsyncKernel (
    int kernelID,
    int minLen,
    int maxLen ) [protected], [inherited]
```

Definition at line 171 of file [cudaModelMatrix.cu](#).

```
00171
00172
00173     //if(kernelID == 0); // cudaStreamSynchronize(this->cudastreams[2]);
00174     //else cudaStreamSynchronize(this->cudastreams[kernelID-1]);
00175     FastRandomWalkCUDAKernel<<cudaBlocks,cudaThreads,0,
00176     this->cudastreams[kernelID]>>(iterationsPerKernelThread, minLen, maxLen,
00177     this->device_outputBuffer[kernelID], this->device_matrixIndex,
00178     this->device_totalEdgeWeights, this->device_valueMatrix, this->device_edgeMatrix,
00179     this->matrixSize, cudaMemPerGrid, this->device_seeds[kernelID]);
00177     //std::cerr << "Started kernel" << kernelID << "\n";
00178 }
```

9.9.3.110 ListCudaDevices()

```
__host__ void Markov::API::CUDA::CUDADeviceController::ListCudaDevices () [static], [inherited]
```

List CUDA devices in the system.

This function will print details of every CUDA capable device in the system.

Example output:

```
Device Number: 0
Device name: GeForce RTX 2070
Memory Clock Rate (KHz): 7001000
Memory Bus Width (bits): 256
Peak Memory Bandwidth (GB/s): 448.064
Max Linear Threads: 1024
```

Definition at line 16 of file [cudaDeviceController.cu](#).

```
00016
00016     host.
00017         int nDevices;
00018         cudaGetDeviceCount(&nDevices);
00019         for (int i = 0; i < nDevices; i++) {
00020             cudaDeviceProp prop;
00021             cudaGetDeviceProperties(&prop, i);
00022             std::cerr << "Device Number: " << i << "\n";
00023             std::cerr << "Device name: " << prop.name << "\n";
00024             std::cerr << "Memory Clock Rate (KHz): " << prop.memoryClockRate << "\n";
00025             std::cerr << "Memory Bus Width (bits): " << prop.memoryBusWidth << "\n";
00026             std::cerr << "Peak Memory Bandwidth (GB/s): " << 2.0 * prop.memoryClockRate *
00027             (prop.memoryBusWidth / 8) / 1.0e6 << "\n";
00028             std::cerr << "Max Linear Threads: " << prop.maxThreadsDim[0] << "\n";
00029         }
00030     }
```

9.9.3.111 MigrateMatrix()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::MigrateMatrix () [inherited]
```

Migrate the class members to the VRAM.

Cannot be used without calling [Markov::API::ModelMatrix::ConstructMatrix](#) at least once. This function will manage the memory allocation and data transfer from CPU RAM to GPU VRAM.

Newly allocated VRAM pointers are set in the class member variables.

Definition at line 20 of file [cudaModelMatrix.cu](#).

```

00020         {
00021             cudastatus;
00022
00023             cudastatus = cudaMalloc((char**) &(this->device_matrixIndex),
00024                 this->matrixSize*sizeof(char));
00025             CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_matrixIndex.");
00026
00027             cudastatus = cudaMalloc((long int **) &(this->device_totalEdgeWeights),
00028                 this->matrixSize*sizeof(long int));
00029             CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_totalEdgeWeights.");
00030
00031             cudastatus = cudaMemcpy(this->device_matrixIndex, this->matrixIndex,
00032                 this->matrixSize*sizeof(char), cudaMemcpyHostToDevice);
00033             CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Index)");
00034
00035             cudastatus = cudaMemcpy(this->device_totalEdgeWeights, this->totalEdgeWeights,
00036                 this->matrixSize*sizeof(long int), cudaMemcpyHostToDevice);
00037             CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Total Edge Values)");
00038
00039             cudastatus = CudaMigrate2DFlat<char>(
00040                 &(this->device_edgeMatrix), this->edgeMatrix, this->matrixSize, this->matrixSize);
00041             CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize edge matrix.");
00042
00043             cudastatus = CudaMigrate2DFlat<long int>(
00044                 &(this->device_valueMatrix), this->valueMatrix, this->matrixSize, this->matrixSize);
00045             CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize value matrix row.");
00046         }

```

9.9.3.112 Nodes() [1/4]

`std::map<char, Node<char>*>* Markov::Model< char >::Nodes () [inline], [inherited]`
Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.9.3.113 Nodes() [2/4]

`std::map<char, Node<char>*>* Markov::Model< char >::Nodes () [inline], [inherited]`
Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.9.3.114 Nodes() [3/4]

`std::map<char, Node<char>*>* Markov::Model< char >::Nodes () [inline], [inherited]`
Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.9.3.115 Nodes() [4/4]

```
std::map<char , Node<char >*>* Markov::Model< char >::Nodes ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.9.3.116 OpenDatasetFile() [1/4]

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename ) [inherited]
```

Open dataset file and return the ifstream pointer.

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

`ifstream*` to the the dataset file

Definition at line 51 of file [markovPasswords.cpp](#).

```
00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060
00061 }
```

{

References [Markov::Model< NodeStorageType >::Import\(\)](#).

Here is the call graph for this function:



9.9.3.117 OpenDatasetFile() [2/4]

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename ) [inherited]
```

Open dataset file and return the ifstream pointer.

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

`ifstream*` to the dataset file

Definition at line 51 of file `markovPasswords.cpp`.

```
00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060
00061 }
```

{

References `Markov::Model< NodeStorageType >::Import()`.

Here is the call graph for this function:

**9.9.3.118 OpenDatasetFile() [3/4]**

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename ) [inherited]
```

Open dataset file and return the `ifstream` pointer.

Parameters

<code>filename</code>	- Filename to open
-----------------------	--------------------

Returns

`ifstream*` to the dataset file

Definition at line 51 of file `markovPasswords.cpp`.

```
00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060
00061 }
```

{

References `Markov::Model< NodeStorageType >::Import()`.

Here is the call graph for this function:



9.9.3.119 OpenDatasetFile() [4/4]

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename ) [inherited]
```

Open dataset file and return the ifstream pointer.

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

`ifstream*` to the the dataset file

Definition at line 51 of file [markovPasswords.cpp](#).

```
00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References [Markov::Model< NodeStorageType >::Import\(\)](#).

Here is the call graph for this function:



9.9.3.120 OptimizeEdgeOrder() [1/4]

```
void Markov::Model< char >::OptimizeEdgeOrder [inherited]
```

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file [model.h](#).

```
00265
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType*>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
00269             Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight() > rhs->EdgeWeight();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00274     }
00275     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00276     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00277 }
```

9.9.3.121 OptimizeEdgeOrder() [2/4]

```
void Markov::Model< char >::OptimizeEdgeOrder [inherited]
```

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file [model.h](#).

```
00265     {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
00269             Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight() > rhs->EdgeWeight();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00278 }
```

9.9.3.122 OptimizeEdgeOrder() [3/4]

[void Markov::Model< char >::OptimizeEdgeOrder](#) [inherited]

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file [model.h](#).

```
00265     {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
00269             Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight() > rhs->EdgeWeight();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00278 }
```

9.9.3.123 OptimizeEdgeOrder() [4/4]

[void Markov::Model< char >::OptimizeEdgeOrder](#) [inherited]

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file [model.h](#).

```
00265     {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
00269             Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight() > rhs->EdgeWeight();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00278 }
```

9.9.3.124 parse()

```
def Python.CudaMarkopy.CudaMarkopyCLI.parse (
    self)
```

Reimplemented from [Python.Markopy.MarkopyCLI](#).

Definition at line 95 of file [cudamarkopy.py](#).

```
00095     def parse(self):
00096         markopy.MarkopyCLI.parse(self)
00097
```

9.9.3.125 parse_arguments() [1/6]

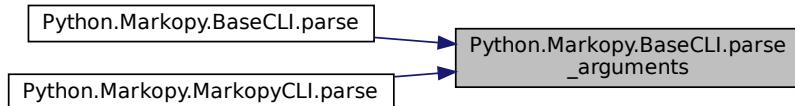
```
def Python.Markopy.BaseCLI.parse_arguments (
    self) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.9.3.126 parse_arguments() [2/6]

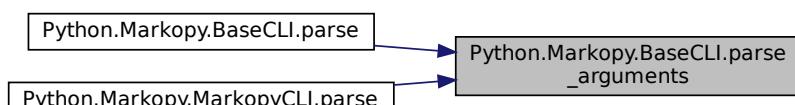
```
def Python.Markopy.BaseCLI.parse_arguments (
    self ) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.9.3.127 parse_arguments() [3/6]

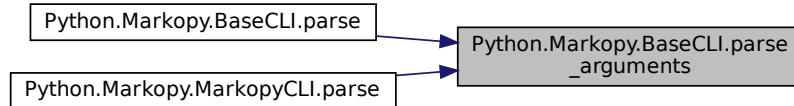
```
def Python.Markopy.BaseCLI.parse_arguments (
    self ) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.9.3.128 parse_arguments() [4/6]

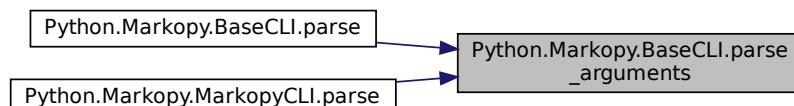
```
def Python.Markopy.BaseCLI.parse_arguments ( self ) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.9.3.129 parse_arguments() [5/6]

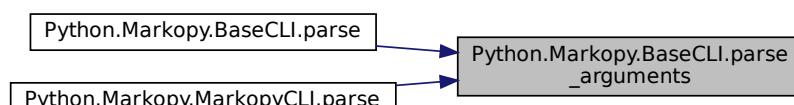
```
def Python.Markopy.BaseCLI.parse_arguments ( self ) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.9.3.130 parse_arguments() [6/6]

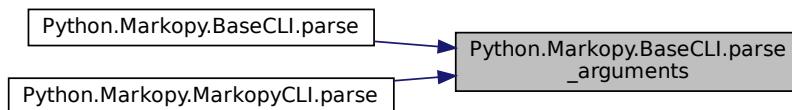
```
def Python.Markopy.BaseCLI.parse_arguments (
    self )  [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.9.3.131 parse_fail()

```
def Python.CudaMarkopy.CudaMarkopyCLI.parse_fail (
    self )
```

Reimplemented from [Python.Markopy.MarkopyCLI](#).

Definition at line 98 of file [cudamarkopy.py](#).

```
00098     def parse_fail(self):
00099         """! @brief Not a valid model type"""
00100         if(self.args.model_type == "CUDA"):
00101             self.cli = CudaModelMatrixCLI()
00102         else:
00103             markopy.MarkopyCLI.parse_fail(self)
00104
00105
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

9.9.3.132 prepKernelMemoryChannel()

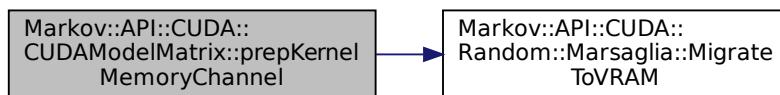
```
__host__ void Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel (
    int number_of_streams )  [protected], [inherited]
```

Definition at line 145 of file [cudaModelMatrix.cu](#).

```
00145
00146
00147     this->cudastreams = new cudaStream_t[number_of_streams];
00148     for(int i=0;i<number_of_streams;i++)
00149         cudaStreamCreate(&this->cudastreams[i]);
00150
00151     this->outputBuffer = new char*[number_of_streams];
00152     for(int i=0;i<number_of_streams;i++)
00153         this->outputBuffer[i]= new char[cudaPerKernelAllocationSize];
00154
00155     cudaError_t cudastatus;
00156     this->device_outputBuffer = new char*[number_of_streams];
00157     for(int i=0;i<number_of_streams;i++){
00158         cudastatus = cudaMalloc((char**) &(device_outputBuffer[i]),
00159                                 cudaPerKernelAllocationSize);
00160         CudaCheckNotifyErr(cudastatus, "Failed to establish memory channel. Possibly out of
VRAM?");
00161     }
00162
00163     this->device_seeds = new unsigned long*[number_of_streams];
00164     for(int i=0;i<number_of_streams;i++){
00165         Markov::API::CUDA::Random::Marsaglia *MEarr = new
Markov::API::CUDA::Random::Marsaglia[cudaGridSize];
00166         this->device_seeds[i] = Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(MEarr,
cudaGridSize);
00167         delete[] MEarr;
```

```
00167         }
00168
00169 }
```

References [Markov::API::CUDA::CUDAModelMatrix::cudaGridSize](#), [Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocation](#), [Markov::API::CUDA::CUDAModelMatrix::device_outputBuffer](#), [Markov::API::CUDA::CUDAModelMatrix::device_seeds](#), [Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM\(\)](#), and [Markov::API::CUDA::CUDAModelMatrix::outputBuffer](#).
Here is the call graph for this function:



9.9.3.133 process()

```
def Python.Markopy.MarkopyCLI.process (
    self ) [inherited]
```

Process parameters for operation.

Reimplemented from [Python.Markopy.BaseCLI](#).

Definition at line 152 of file [markopy.py](#).

```
00152     def process(self):
00153         """! @brief pass the process request to selected submodel"""
00154         return self.cli.process()
00155
```

References [Python.CudaMarkopy.CudaMarkopyCLI.cli](#), and [Python.Markopy.MarkopyCLI.cli](#).

9.9.3.134 RandomWalk() [1/4]

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```

00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319         else if (temp_node == NULL) {
00320             break;
00321         }
00322         n = temp_node;
00323
00324         buffer[len++] = n->NodeValue();
00325
00326     //null terminate the string
00327     buffer[len] = 0x00;
00328
00329     //do something with the generated string
00330     return buffer; //for now
00331
00332
00333
00334 }
```

9.9.3.135 RandomWalk() [2/4]

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
```

```
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```

00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319         else if (temp_node == NULL) {
00320             break;
00321         }
00322         n = temp_node;
00323
00324         buffer[len++] = n->NodeValue();
00325
00326     //null terminate the string
00327     buffer[len] = 0x00;
00328
00329     //do something with the generated string
00330     return buffer; //for now
00331
00332
00333
00334 }
```

9.9.3.136 RandomWalk() [3/4]

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
```

```
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```

00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319         else if (temp_node == NULL) {
00320             break;
00321         }
00322         n = temp_node;
00323
00324         buffer[len++] = n->NodeValue();
00325
00326     //null terminate the string
00327     buffer[len] = 0x00;
00328
00329     //do something with the generated string
00330     return buffer; //for now
00331
00332
00333
00334 }
```

9.9.3.137 RandomWalk() [4/4]

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
```

```
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```

00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext (randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL) {
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue ();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

9.9.3.138 Save() [1/4]

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
```

Export model to file.

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

`std::ofstream*` of the exported file.

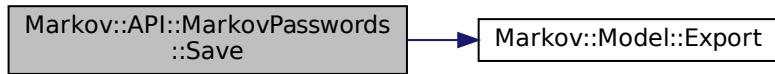
Definition at line 106 of file [markovPasswords.cpp](#).

```

00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export (exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.9.3.139 Save() [2/4]

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
Export model to file.
```

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

```
00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.9.3.140 Save() [3/4]

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
Export model to file.
```

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

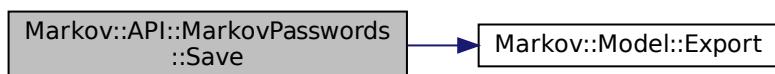
`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

```
00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:

**9.9.3.141 Save() [4/4]**

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
```

Export model to file.

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

```
00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.9.3.142 StarterNode() [1/4]

```
Node< char >* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.9.3.143 StarterNode() [2/4]

```
Node< char >* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.9.3.144 StarterNode() [3/4]

```
Node< char >* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.9.3.145 StarterNode() [4/4]

```
Node< char >* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

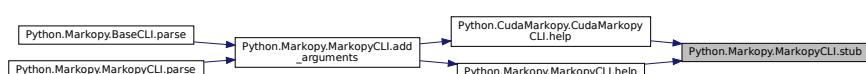
```
00171 { return starterNode; }
```

9.9.3.146 stub()

```
def Python.Markopy.MarkopyCLI.stub (
    self) [inherited]
Definition at line 156 of file markopy.py.
00156     def stub(self):
00157         """! @brief stub function to hack help requests"""
00158         return
00159
00160
```

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

Here is the caller graph for this function:



9.9.3.147 Train() [1/4]

```
void Markov::API::ModelMatrix::Train (
    const char * datasetFileName,
    char delimiter,
    int threads ) [inherited]
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

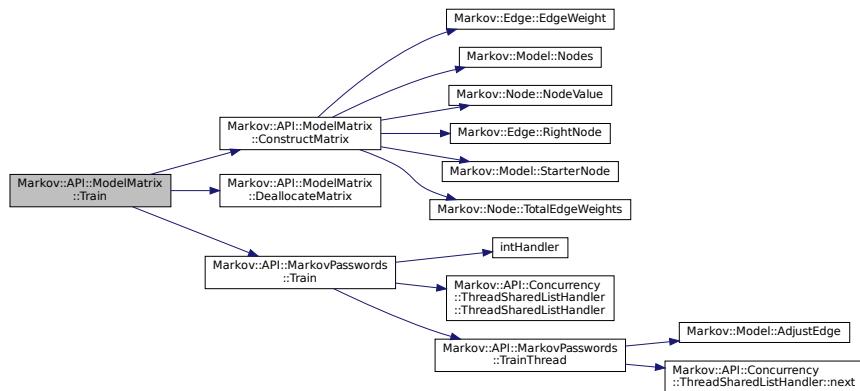
Construct the matrix when done.

Definition at line 25 of file [modelMatrix.cpp](#).

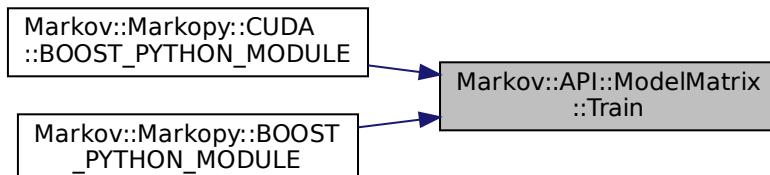
```
00025
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.148 Train() [2/4]

```
void Markov::API::ModelMatrix::Train (
    const char * datasetFileName,
    char delimiter,
    int threads ) [inherited]
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

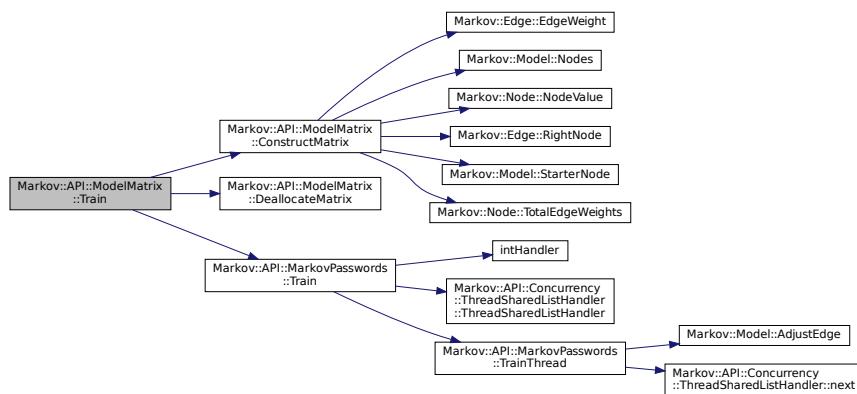
Construct the matrix when done.

Definition at line 25 of file [modelMatrix.cpp](#).

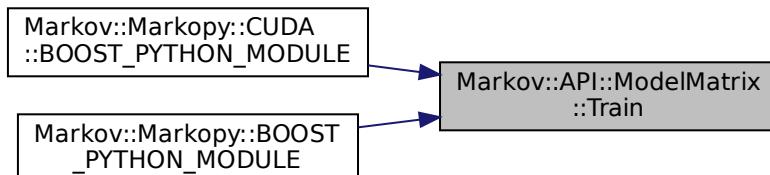
```
00025
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.149 Train() [3/4]

```
void Markov::API::ModelMatrix::Train (
    const char * datasetFileName,
    char delimiter,
    int threads ) [inherited]
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

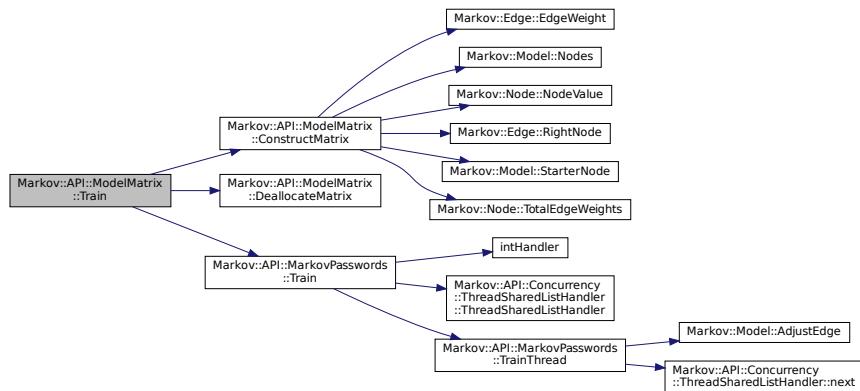
Construct the matrix when done.

Definition at line 25 of file [modelMatrix.cpp](#).

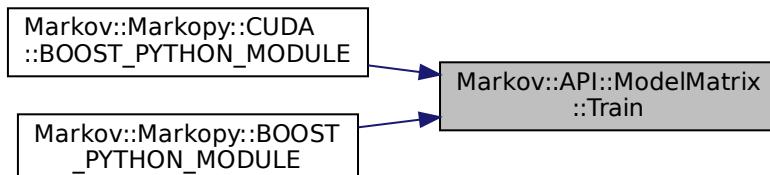
```
00025
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.150 train() [1/6]

```
def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False ) [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

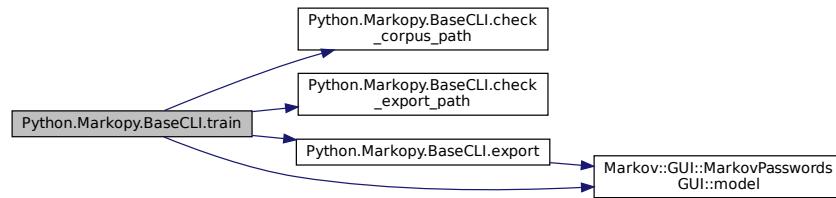
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104         """
00105         logging pprint("Training.")
00106
00107         if not (dataset and seperator and (output or not output_forced)):
00108             logging pprint(f"Training mode requires -d/--dataset', -o/--output' if output_forced
00109             else") and -s/--seperator parameters. Exiting."
00110             return False
00111
00112         if not bulk and not self.check_corpus_path(dataset):
00113             logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00114             return False
00115
00116         if not self.check_export_path(output):
00117             logging pprint(f"Cannot create output at {output}")
00118             return False
00119
00120         if(seperator == '\\t'):
00121             logging pprint("Escaping seperator.", 3)
00122             seperator = '\t'
00123
00124         if(len(seperator)!=1):
00125             logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00126             accepted.')
00127             exit(4)
00128
00129         if(output):
00130             logging pprint(f'Starting training.', 3)
00131             self.model.Train(dataset,seperator, int(self.args.threads))
00132             logging pprint(f'Training completed.', 2)
00133
00134         if(output):
00135             logging pprint(f'Exporting model to {output}', 2)
00136             self.export(output)
00137         else:
00138             logging pprint(f'Model will not be exported.', 1)
00139
00140     return True
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#),

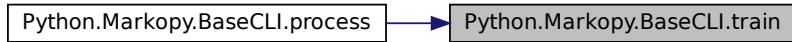
`Python.Markopy.MarkovPasswordsCLI.model`, and `Markov::GUI::MarkovPasswordsGUI.model()`.

Referenced by `Python.Markopy.BaseCLI.process()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.151 train() [2/6]

```
def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False ) [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file `base.py`.

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104
00105             logging pprint("Training.")
00106             if not (dataset and seperator and (output or not output_forced)):
00107                 raise ValueError("Dataset and seperator must be provided")
00108
00109             if output_forced:
00110                 self._model = self._model.load()
00111             else:
00112                 self._model = self._model.train(dataset, seperator, output)
00113
00114             if bulk:
00115                 self._model = self._model.bulk_train(dataset, seperator, output)
00116             else:
00117                 self._model = self._model.train(dataset, seperator, output)
00118
00119             self._model.save()
00120
00121             return self._model
```

```

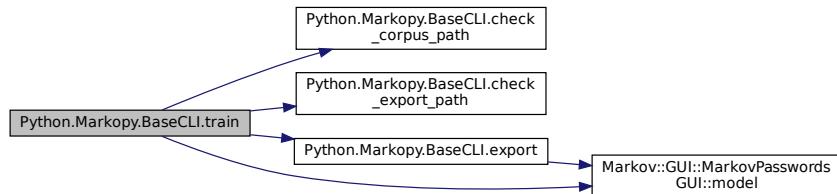
00107         logging pprint(f"Training mode requires -d/--dataset', -o/--output' if output_forced
00108     else") and -s/--seperator parameters. Exiting.")
00109     return False
00110
00111     if not bulk and not self.check_corpus_path(dataset):
00112         logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00113         return False
00114
00115     if not self.check_export_path(output):
00116         logging pprint(f"Cannot create output at {output}")
00117         return False
00118
00119     if(seperator == '\\t'):
00120         logging pprint("Escaping seperator.", 3)
00121         seperator = '\t'
00122
00123     if(len(seperator) !=1):
00124         logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00125 accepted.')
00126         exit(4)
00127
00128     logging pprint(f'Starting training.', 3)
00129     self.model.Train(dataset,seperator, int(self.args.threads))
00130     logging pprint(f'Training completed.', 2)
00131
00132     if(output):
00133         logging pprint(f'Exporting model to {output}', 2)
00134         self.export(output)
00135     else:
00136         logging pprint(f'Model will not be exported.', 1)
00137
00138     return True

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.152 train() [3/6]

```

def Python.Markopy.BaseCLI.train (
        self,
        str dataset,
        str seperator,
        str output,

```

```
    bool output_forced = False,
    bool bulk = False ) [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	separator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

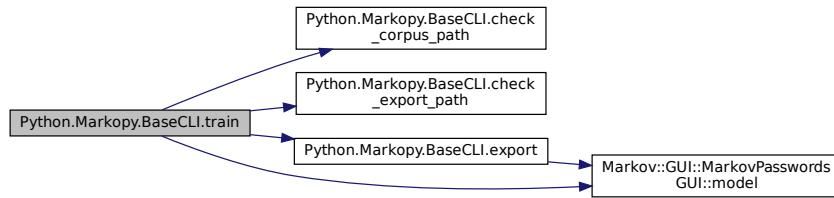
Definition at line 94 of file [base.py](#).

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096             """
00097                 @brief Train a model via CLI parameters
00098                 @param model Model instance
00099                 @param dataset filename for the dataset
00100                 @param seperator separator used with the dataset
00101                 @param output output filename
00102                 @param output_forced force overwrite
00103                 @param bulk marks bulk operation with directories
00104             """
00105             logging pprint("Training.")
00106             if not (dataset and seperator and (output or not output_forced)):
00107                 logging pprint(f"Training mode requires -d/--dataset{'', -o/--output' if output_forced
00108 else'} and -s/-seperator parameters. Exiting.")
00109             return False
00110             if not bulk and not self.check_corpus_path(dataset):
00111                 logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112                 return False
00113             if not self.check_export_path(output):
00114                 logging pprint(f"Cannot create output at {output}")
00115                 return False
00116             if(seperator == '\\t'):
00117                 logging pprint("Escaping seperator.", 3)
00118                 seperator = '\t'
00119             if(len(seperator) !=1):
00120                 logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00121 accepted.')
00122                 exit(4)
00123             logging pprint(f'Starting training.', 3)
00124             self.model.Train(dataset, seperator, int(self.args.threads))
00125             logging pprint(f'Training completed.', 2)
00126             if(output):
00127                 logging pprint(f'Exporting model to {output}', 2)
00128                 self.export(output)
00129             else:
00130                 logging pprint(f'Model will not be exported.', 1)
00131             return True
00132
00133
00134
00135
00136
00137
```

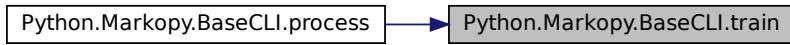
References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.153 train() [4/6]

```
def Python.Markopy.BaseCLI.train (
        self,
        str dataset,
        str seperator,
        str output,
        bool output_forced = False,
        bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104         """
00105         logging pprint("Training.")
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging pprint(f"Training mode requires -d/--dataset(',-o/--output' if output_forced
00108 else') and -s/--seperator parameters. Exiting.")
00109         return False
```

```

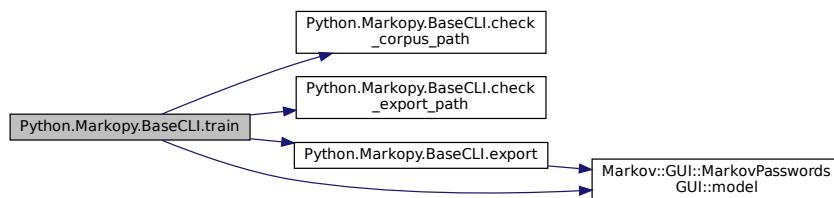
00110     if not bulk and not self.check_corpus_path(dataset):
00111         logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112         return False
00113
00114     if not self.check_export_path(output):
00115         logging pprint(f"Cannot create output at {output}")
00116         return False
00117
00118     if(seperator == '\\t'):
00119         logging pprint("Escaping seperator.", 3)
00120         seperator = '\t'
00121
00122     if(len(seperator) !=1):
00123         logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00124 accepted.')
00125         exit(4)
00126
00127     logging pprint(f'Starting training.', 3)
00128     self.model.Train(dataset,seperator, int(self.args.threads))
00129     logging pprint(f'Training completed.', 2)
00130
00131     if(output):
00132         logging pprint(f'Exporting model to {output}', 2)
00133         self.export(output)
00134     else:
00135         logging pprint(f'Model will not be exported.', 1)
00136
00137     return True
00138

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.154 train() [5/6]

```

def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False )  [inherited]

```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

```

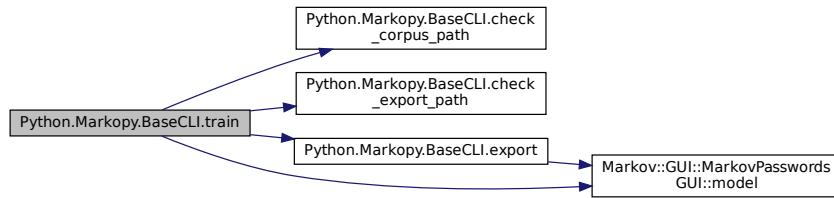
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096             """!
00097                 @brief Train a model via CLI parameters
00098                 @param model Model instance
00099                 @param dataset filename for the dataset
00100                 @param seperator seperator used with the dataset
00101                 @param output output filename
00102                 @param output_forced force overwrite
00103                 @param bulk marks bulk operation with directories
00104             """
00105             logging pprint("Training.")
00106             if not (dataset and seperator and (output or not output_forced)):
00107                 logging pprint(f"Training mode requires -d/--dataset', -o/--output' if output_forced
00108                 else") and -s/-seperator parameters. Exiting.")
00109             return False
00110             if not bulk and not self.check_corpus_path(dataset):
00111                 logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112                 return False
00113             if not self.check_export_path(output):
00114                 logging pprint(f"Cannot create output at {output}")
00115                 return False
00116             if(seperator == '\\t'):
00117                 logging pprint("Escaping seperator.", 3)
00118                 seperator = '\t'
00119             if(len(seperator)!=1):
00120                 logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00121                 accepted.')
00122                 exit(4)
00123             logging pprint(f'Starting training.', 3)
00124             self.model.Train(dataset, seperator, int(self.args.threads))
00125             logging pprint(f'Training completed.', 2)
00126             if(output):
00127                 logging pprint(f'Exporting model to {output}', 2)
00128                 self.export(output)
00129             else:
00130                 logging pprint(f'Model will not be exported.', 1)
00131             return True
00132
00133
00134
00135
00136
00137

```

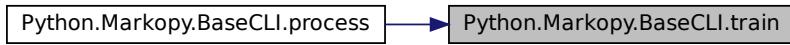
References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.155 train() [6/6]

```
def Python.Markopy.BaseCLI.train (
        self,
        str dataset,
        str seperator,
        str output,
        bool output_forced = False,
        bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104         """
00105         logging pprint("Training.")
00106         if not (dataset and seperator and (output or not output_forced)):
00107             logging pprint(f"Training mode requires -d/--dataset(',-o/--output' if output_forced
00108 else') and -s/--seperator parameters. Exiting.")
00109         return False
```

```

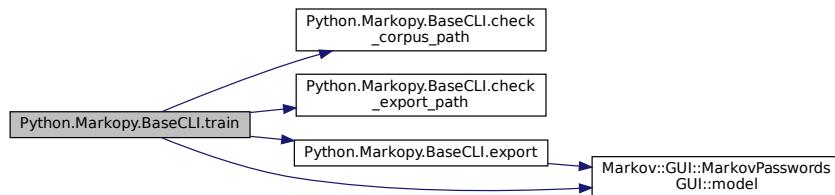
00110     if not bulk and not self.check_corpus_path(dataset):
00111         logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112         return False
00113
00114     if not self.check_export_path(output):
00115         logging pprint(f"Cannot create output at {output}")
00116         return False
00117
00118     if(seperator == '\\t'):
00119         logging pprint("Escaping seperator.", 3)
00120         seperator = '\t'
00121
00122     if(len(seperator) != 1):
00123         logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00124 accepted.')
00125         exit(4)
00126
00127     logging pprint(f'Starting training.', 3)
00128     self.model.Train(dataset, seperator, int(self.args.threads))
00129     logging pprint(f'Training completed.', 2)
00130
00131     if(output):
00132         logging pprint(f'Exporting model to {output}', 2)
00133         self.export(output)
00134     else:
00135         logging pprint(f'Model will not be exported.', 1)
00136
00137     return True
00138

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.3.156 Train() [4/4]

```

def Python.Markopy.MarkovModel.Train (
        str dataset,
        str seperator,
        int threads ) [inherited]

```

Definition at line 30 of file [mm.py](#).

```

00030     def Train(dataset: str, seperator : str, threads : int):
00031         pass
00032

```

9.9.3.157 TrainThread()

```
void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private], [inherited]
```

A single thread invoked by the Train function.

Parameters

<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

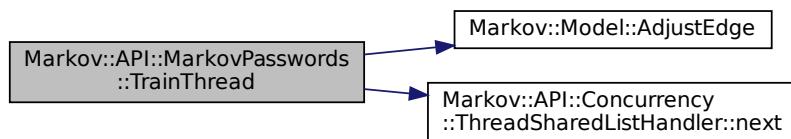
Definition at line 85 of file markovPasswords.cpp.

```
00085     {
00086     char format_str[] = "%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to-
00097         "%ld,%s"
00098     #else
00099         sscanf(line.c_str(), format_str, &oc, linebuf);
00100     #endif
00101     this->AdjustEdge((const char*)linebuf, oc);
00102     delete linebuf;
00103 }
```

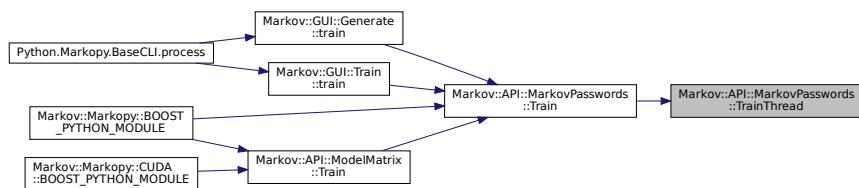
References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), [keepRunning](#), and [Markov::API::Concurrency::ThreadSharedListHandler::next\(\)](#).

Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.9.4 Member Data Documentation

9.9.4.1 alternatingKernels

```
int Markov::API::CUDA::CUDAModelMatrix::alternatingKernels [private], [inherited]
Definition at line 135 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\).
```

9.9.4.2 args

```
Python.CudaMarkopy.CudaMarkopyCLI.args
Definition at line 64 of file cudamarkopy.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI.\_generate\(\), Python.Markopy.BaseCLI.\_generate\(\),
Python.Markopy.ModelMatrixCLI.\_generate\(\), Python.Markopy.MarkovPasswordsCLI.\_generate\(\), Python.Markopy.BaseCLI.generate\(\)
Python.Markopy.MarkopyCLI.help\(\), Python.Markopy.BaseCLI.init\_post\_arguments\(\), Python.Markopy.MarkopyCLI.parse\(\),
Python.CudaMarkopy.CudaMarkopyCLI.parse\_fail\(\), Python.Markopy.BaseCLI.process\(\), and Python.Markopy.BaseCLI.train\(\).
```

9.9.4.3 bInfinite

```
Python.CudaMarkopy.CudaModelMatrixCLI.bInfinite [inherited]
Definition at line 73 of file cudammx.py.
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI.\_generate\(\).
```

9.9.4.4 cli

```
Python.CudaMarkopy.CudaMarkopyCLI.cli
Definition at line 101 of file cudamarkopy.py.
Referenced by Python.Markopy.MarkopyCLI.process\(\).
```

9.9.4.5 cudaBlocks

```
int Markov::API::CUDA::CUDAModelMatrix::cudaBlocks [private], [inherited]
Definition at line 125 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\).
```

9.9.4.6 cudaGridSize

```
int Markov::API::CUDA::CUDAModelMatrix::cudaGridSize [private], [inherited]
Definition at line 131 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\), and Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel\(\).
```

9.9.4.7 cudaMemPerGrid

```
int Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid [private], [inherited]
Definition at line 132 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\), and Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelMemoryChannel\(\).
```

9.9.4.8 cudaPerKernelAllocationSize

```
long int Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize [private], [inherited]
Definition at line 133 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\), Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelMemoryChannel\(\), and Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel\(\).
```

9.9.4.9 cudastreams

cudaStream_t* Markov::API::CUDA::CUDAModelMatrix::cudastreams [private], [inherited]
Definition at line 139 of file [cudaModelMatrix.h](#).

9.9.4.10 cudaThreads

int Markov::API::CUDA::CUDAModelMatrix::cudaThreads [private], [inherited]
Definition at line 126 of file [cudaModelMatrix.h](#).
Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#).

9.9.4.11 datasetFile

std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]
Definition at line 123 of file [markovPasswords.h](#).

9.9.4.12 device_edgeMatrix

char* Markov::API::CUDA::CUDAModelMatrix::device_edgeMatrix [private], [inherited]
VRAM Address pointer of edge matrix (from [modelMatrix.h](#))
Definition at line 88 of file [cudaModelMatrix.h](#).

9.9.4.13 device_matrixIndex

char* Markov::API::CUDA::CUDAModelMatrix::device_matrixIndex [private], [inherited]
VRAM Address pointer of matrixIndex (from [modelMatrix.h](#))
Definition at line 98 of file [cudaModelMatrix.h](#).

9.9.4.14 device_outputBuffer

char** Markov::API::CUDA::CUDAModelMatrix::device_outputBuffer [private], [inherited]
RandomWalk results in device.
Definition at line 108 of file [cudaModelMatrix.h](#).
Referenced by [Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel\(\)](#).

9.9.4.15 device_seeds

unsigned long** Markov::API::CUDA::CUDAModelMatrix::device_seeds [private], [inherited]
Definition at line 137 of file [cudaModelMatrix.h](#).
Referenced by [Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel\(\)](#).

9.9.4.16 device_totalEdgeWeights

long int* Markov::API::CUDA::CUDAModelMatrix::device_totalEdgeWeights [private], [inherited]
VRAM Address pointer of total edge weights (from [modelMatrix.h](#))
Definition at line 103 of file [cudaModelMatrix.h](#).

9.9.4.17 device_valueMatrix

long int* Markov::API::CUDA::CUDAModelMatrix::device_valueMatrix [private], [inherited]
VRAM Address pointer of value matrix (from [modelMatrix.h](#))
Definition at line 93 of file [cudaModelMatrix.h](#).

9.9.4.18 edgeMatrix [1/3]

`char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]`

2-D Character array for the edge Matrix (The characters of Nodes)

Definition at line 175 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.9.4.19 edgeMatrix [2/3]

`char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]`

2-D Character array for the edge Matrix (The characters of Nodes)

Definition at line 175 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.9.4.20 edgeMatrix [3/3]

`char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]`

2-D Character array for the edge Matrix (The characters of Nodes)

Definition at line 175 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.9.4.21 edges

`std::vector<Edge<char>>*> Markov::Model< char >::edges [private], [inherited]`

A list of all edges in this model.

Definition at line 204 of file [model.h](#).

9.9.4.22 fileIO [1/2]

`Python.Markopy.ModelMatrixCLI.fileIO [inherited]`

Definition at line 38 of file [mmx.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), and [Python.Markopy.ModelMatrixCLI._generate\(\)](#).

9.9.4.23 fileIO [2/2]

`Python.Markopy.ModelMatrixCLI.fileIO [inherited]`

Definition at line 38 of file [mmx.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), and [Python.Markopy.ModelMatrixCLI._generate\(\)](#).

9.9.4.24 flatEdgeMatrix

`char* Markov::API::CUDA::CUDAModelMatrix::flatEdgeMatrix [private], [inherited]`

Adding Edge matrix end-to-end and resize to 1-D array for better performance on traversing.

Definition at line 118 of file [cudaModelMatrix.h](#).

Referenced by [Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix\(\)](#).

9.9.4.25 flatValueMatrix

```
long int* Markov::API::CUDA::CUDAModelMatrix::flatValueMatrix [private], [inherited]
```

Adding Value matrix end-to-end and resize to 1-D array for better performance on traversing.

Definition at line 123 of file [cudaModelMatrix.h](#).

Referenced by [Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix\(\)](#).

9.9.4.26 iterationsPerKernelThread

```
int Markov::API::CUDA::CUDAModelMatrix::iterationsPerKernelThread [private], [inherited]
```

Definition at line 127 of file [cudaModelMatrix.h](#).

Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#).

9.9.4.27 matrixIndex [1/3]

```
char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]
```

to hold the Matrix index (To hold the orders of 2-D arrays')

Definition at line 190 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.9.4.28 matrixIndex [2/3]

```
char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]
```

to hold the Matrix index (To hold the orders of 2-D arrays')

Definition at line 190 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.9.4.29 matrixIndex [3/3]

```
char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]
```

to hold the Matrix index (To hold the orders of 2-D arrays')

Definition at line 190 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.9.4.30 matrixSize [1/3]

```
int Markov::API::ModelMatrix::matrixSize [protected], [inherited]
```

to hold Matrix size

Definition at line 185 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#), and [Markov::API::CUDA::CUDAMoo](#)

9.9.4.31 matrixSize [2/3]

```
int Markov::API::ModelMatrix::matrixSize [protected], [inherited]
```

to hold Matrix size

Definition at line 185 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#), and [Markov::API::CUDA::CUDAMoo](#)

9.9.4.32 matrixSize [3/3]

`int Markov::API::ModelMatrix::matrixSize [protected], [inherited]`
to hold Matrix size

Definition at line 185 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#), and [Markov::API::CUDA::CUDAMoo](#).

9.9.4.33 model [1/4]

`Python.CudaMarkopy.CudaModelMatrixCLI.model [inherited]`
Definition at line 65 of file [cudammx.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.9.4.34 model [2/4]

`Python.Markopy.BaseCLI.model [inherited]`
Definition at line 40 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.9.4.35 model [3/4]

`Python.Markopy.ModelMatrixCLI.model [inherited]`
Definition at line 30 of file [mmx.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.9.4.36 model [4/4]

`Python.Markopy.MarkovPasswordsCLI.model [inherited]`
Definition at line 29 of file [mp.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.9.4.37 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`
Dataset file input of our system

Definition at line 124 of file [markovPasswords.h](#).

9.9.4.38 nodes

`std::map<char , Node<char *>> Markov::Model< char >::nodes [private], [inherited]`
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.

Definition at line 193 of file [model.h](#).

9.9.4.39 `numberOfPartitions`

```
int Markov::API::CUDA::CUDAModelMatrix::numberOfPartitions [private], [inherited]
Definition at line 130 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\).
```

9.9.4.40 `outputBuffer`

```
char** Markov::API::CUDA::CUDAModelMatrix::outputBuffer [private], [inherited]
RandomWalk results in host.
Definition at line 113 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput\(\), and Markov::API::CUDA::CUDAModelMatrix::pr
```

9.9.4.41 `outputFile`

```
std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]
File to save model of our system
```

Definition at line 125 of file [markovPasswords.h](#).

9.9.4.42 `parser` [1/6]

```
Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.\_\_init\_\_\(\), Python.CudaMarkopy.CudaModelMatrixCLI.add\_arguments\(\),
Python.Markopy.BaseCLI.add\_arguments\(\), Python.Markopy.AbstractGenerationModelCLI.add\_arguments\(\),
Python.Markopy.AbstractTrainingModelCLI.add\_arguments\(\), Python.Markopy.MarkopyCLI.add\_arguments\(\),
Python.Markopy.ModelMatrixCLI.add\_arguments\(\), Python.CudaMarkopy.CudaMarkopyCLI.help\(\), and Python.Markopy.MarkopyCLI.
```

9.9.4.43 `parser` [2/6]

```
Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.\_\_init\_\_\(\), Python.CudaMarkopy.CudaModelMatrixCLI.add\_arguments\(\),
Python.Markopy.BaseCLI.add\_arguments\(\), Python.Markopy.AbstractGenerationModelCLI.add\_arguments\(\),
Python.Markopy.AbstractTrainingModelCLI.add\_arguments\(\), Python.Markopy.MarkopyCLI.add\_arguments\(\),
Python.Markopy.ModelMatrixCLI.add\_arguments\(\), Python.CudaMarkopy.CudaMarkopyCLI.help\(\), and Python.Markopy.MarkopyCLI.
```

9.9.4.44 `parser` [3/6]

```
Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.\_\_init\_\_\(\), Python.CudaMarkopy.CudaModelMatrixCLI.add\_arguments\(\),
Python.Markopy.BaseCLI.add\_arguments\(\), Python.Markopy.AbstractGenerationModelCLI.add\_arguments\(\),
Python.Markopy.AbstractTrainingModelCLI.add\_arguments\(\), Python.Markopy.MarkopyCLI.add\_arguments\(\),
Python.Markopy.ModelMatrixCLI.add\_arguments\(\), Python.CudaMarkopy.CudaMarkopyCLI.help\(\), and Python.Markopy.MarkopyCLI.
```

9.9.4.45 `parser` [4/6]

```
Python.Markopy.BaseCLI.parser [inherited]
Definition at line 25 of file base.py.
Referenced by Python.CudaMarkopy.CudaMarkopyCLI.\_\_init\_\_\(\), Python.CudaMarkopy.CudaModelMatrixCLI.add\_arguments\(\),
Python.Markopy.BaseCLI.add\_arguments\(\), Python.Markopy.AbstractGenerationModelCLI.add\_arguments\(\),
```

[Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#),
[Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.](#)

9.9.4.46 parser [5/6]

Python.Markopy.BaseCLI.parser [inherited]

Definition at line 25 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#),
[Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#),
[Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#),
[Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.](#)

9.9.4.47 parser [6/6]

Python.Markopy.BaseCLI.parser [inherited]

Definition at line 25 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#),
[Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#),
[Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#),
[Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.](#)

9.9.4.48 print_help [1/6]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.9.4.49 print_help [2/6]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.9.4.50 print_help [3/6]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.9.4.51 print_help [4/6]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.9.4.52 print_help [5/6]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.9.4.53 print_help [6/6]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.9.4.54 ready [1/3]

bool Markov::API::ModelMatrix::ready [protected], [inherited]

True when matrix is constructed. False if not.

Definition at line 200 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::ModelMatrix\(\)](#).

9.9.4.55 ready [2/3]

bool Markov::API::ModelMatrix::ready [protected], [inherited]

True when matrix is constructed. False if not.

Definition at line 200 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::ModelMatrix\(\)](#).

9.9.4.56 ready [3/3]

bool Markov::API::ModelMatrix::ready [protected], [inherited]

True when matrix is constructed. False if not.

Definition at line 200 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::ModelMatrix\(\)](#).

9.9.4.57 starterNode

Node<char *>* Markov::Model< char >::starterNode [private], [inherited]

Starter Node of this model.

Definition at line 198 of file [model.h](#).

9.9.4.58 totalEdgeWeights [1/3]

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]

Array of the Total Edge Weights.

Definition at line 195 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.9.4.59 totalEdgeWeights [2/3]

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]

Array of the Total Edge Weights.

Definition at line 195 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.9.4.60 totalEdgeWeights [3/3]

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]

Array of the Total Edge Weights.

Definition at line 195 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.9.4.61 totalOutputPerKernel

long int Markov::API::CUDA::CUDAModelMatrix::totalOutputPerKernel [private], [inherited]

Definition at line 129 of file [cudaModelMatrix.h](#).

Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#).

9.9.4.62 totalOutputPerSync

long int Markov::API::CUDA::CUDAModelMatrix::totalOutputPerSync [private], [inherited]

Definition at line 128 of file [cudaModelMatrix.h](#).

Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#).

9.9.4.63 valueMatrix [1/3]

long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]

2-d Integer array for the value Matrix (For the weights of Edges)

Definition at line 180 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.9.4.64 valueMatrix [2/3]

long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]

2-d Integer array for the value Matrix (For the weights of Edges)

Definition at line 180 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.9.4.65 valueMatrix [3/3]

long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]

2-d Integer array for the value Matrix (For the weights of Edges)

Definition at line 180 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

The documentation for this class was generated from the following file:

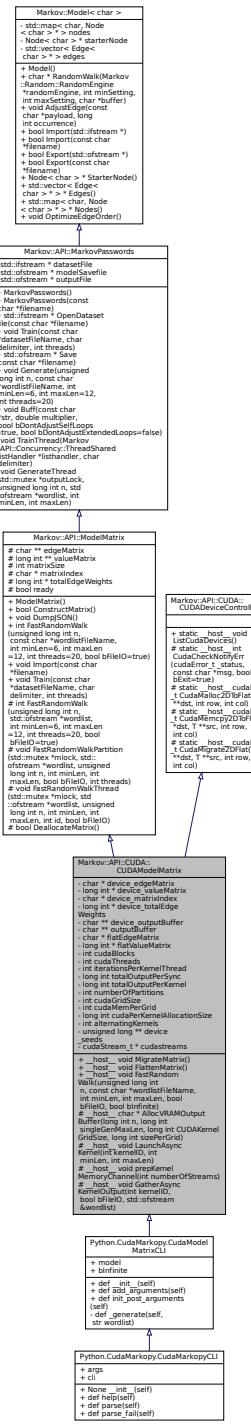
- [Markopy/CudaMarkopy/src/CLI/cudamarkopy.py](#)

9.10 Markov::API::CUDA::CUDAModelMatrix Class Reference

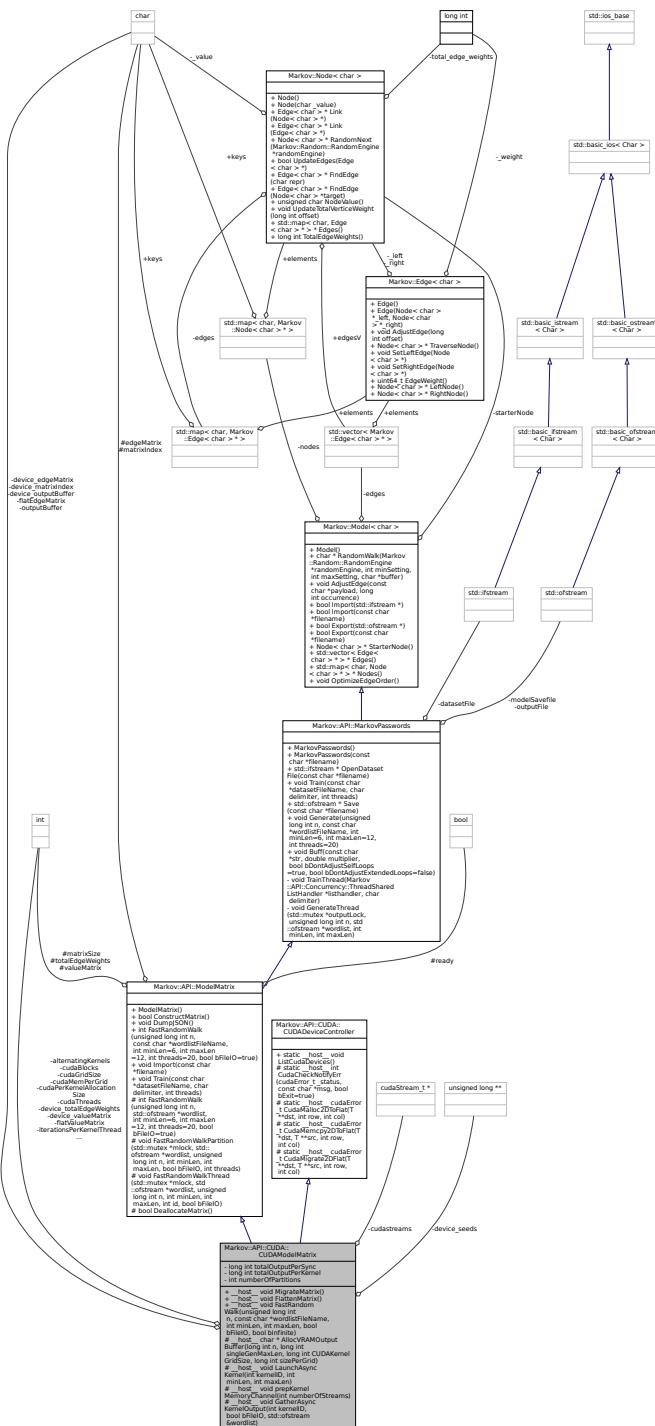
Extension of [Markov::API::ModelMatrix](#) which is modified to run on GPU devices.

```
#include <cudaModelMatrix.h>
```

Inheritance diagram for Markov::API::CUDA::CUDAModelMatrix::



Collaboration diagram for Markov::API::CUDA::CUDAModelMatrix::



Public Member Functions

- `__host__ void MigrateMatrix ()`
Migrate the class members to the VRAM.
 - `__host__ void FlattenMatrix ()`
Flatten migrated matrix from 2d to 1d.
 - `__host__ void FastRandomWalk (unsigned long int n, const char *wordlistFileName, int minLen, int maxLen, bool bFileIO, bool bInfinite)`

- **Random** walk on the Matrix-reduced [Markov::Model](#).
 - bool [ConstructMatrix](#) ()

Construct the related Matrix data for the model.
 - void [DumpJSON](#) ()

Debug function to dump the model to a JSON file.
 - int [FastRandomWalk](#) (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileO=true)

Random walk on the Matrix-reduced [Markov::Model](#).
- void [Import](#) (const char *filename)

Open a file to import with filename, and call bool [Model::Import](#) with std::ifstream.
- bool [Import](#) (std::ifstream *)

Import a file to construct the model.
- void [Train](#) (const char *datasetFileName, char delimiter, int threads)

Train the model with the dataset file.
- std::ifstream * [OpenDatasetFile](#) (const char *filename)

Open dataset file and return the ifstream pointer.
- std::ofstream * [Save](#) (const char *filename)

Export model to file.
- void [Generate](#) (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

Call [Markov::Model::RandomWalk](#) n times, and collect output.
- void [Buff](#) (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)

Buff expression of some characters in the model.
- char * [RandomWalk](#) ([Markov::Random::RandomEngine](#) *randomEngine, int minSetting, int maxSetting, char *buffer)

Do a random walk on this model.
- void [AdjustEdge](#) (const char *payload, long int occurrence)

Adjust the model with a single string.
- bool [Export](#) (std::ofstream *)

Export a file of the model.
- bool [Export](#) (const char *filename)

Open a file to export with filename, and call bool [Model::Export](#) with std::ofstream.
- **Node< char > *** [StarterNode](#) ()

Return starter Node.
- std::vector< [Edge< char >](#) * > * [Edges](#) ()

Return a vector of all the edges in the model.
- std::map< char, [Node< char >](#) * > * [Nodes](#) ()

Return starter Node.
- void [OptimizeEdgeOrder](#) ()

Sort edges of all nodes in the model ordered by edge weights.

Static Public Member Functions

- static __host__ void [ListCudaDevices](#) ()

List CUDA devices in the system.

Protected Member Functions

- `__host__ char * AllocVRAMOutputBuffer` (long int n, long int singleGenMaxLen, long int CUDAKernelGrid←Size, long int sizePerGrid)

Allocate the output buffer for kernel operation.
- `__host__ void LaunchAsyncKernel` (int kernelID, int minLen, int maxLen)
- `__host__ void prepKernelMemoryChannel` (int numberOfWorklists)
- `__host__ void GatherAsyncKernelOutput` (int kernelID, bool bFileIO, std::ofstream &wordlist)
- `int FastRandomWalk` (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced `Markov::Model`.
- `void FastRandomWalkPartition` (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

A single partition of `FastRandomWalk` event.
- `void FastRandomWalkThread` (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)

A single thread of a single partition of `FastRandomWalk`.
- `bool DeallocateMatrix ()`

Deallocate matrix and make it ready for re-construction.

Static Protected Member Functions

- `static __host__ int CudaCheckNotifyErr` (cudaError_t _status, const char *msg, bool bExit=true)

Check results of the last operation on GPU.
- template<typename T >
 `static __host__ cudaError_t CudaMalloc2DToFlat` (T **dst, int row, int col)

Malloc a 2D array in device space.
- template<typename T >
 `static __host__ cudaError_t CudaMemcpy2DToFlat` (T *dst, T **src, int row, int col)

Memcpy a 2D array in device space after flattening.
- template<typename T >
 `static __host__ cudaError_t CudaMigrate2DFlat` (T **dst, T **src, int row, int col)

Both malloc and memcpy a 2D array into device VRAM.

Protected Attributes

- `char ** edgeMatrix`

2-D Character array for the edge Matrix (The characters of Nodes)
- `long int ** valueMatrix`

2-d Integer array for the value Matrix (For the weights of Edges)
- `int matrixSize`

to hold Matrix size
- `char * matrixIndex`

to hold the Matrix index (To hold the orders of 2-D arrays')
- `long int * totalEdgeWeights`

Array of the Total Edge Weights.
- `bool ready`

True when matrix is constructed. False if not.

Private Member Functions

- `void TrainThread` (`Markov::API::Concurrency::ThreadSharedListHandler` *listHandler, char delimiter)

A single thread invoked by the Train function.
- `void GenerateThread` (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)

A single thread invoked by the Generate function.

Private Attributes

- `char * device_edgeMatrix`
VRAM Address pointer of edge matrix (from `modelMatrix.h`)
- `long int * device_valueMatrix`
VRAM Address pointer of value matrix (from `modelMatrix.h`)
- `char * device_matrixIndex`
VRAM Address pointer of matrixIndex (from `modelMatrix.h`)
- `long int * device_totalEdgeWeights`
VRAM Address pointer of total edge weights (from `modelMatrix.h`)
- `char ** device_outputBuffer`
RandomWalk results in device.
- `char ** outputBuffer`
RandomWalk results in host.
- `char * flatEdgeMatrix`
Adding Edge matrix end-to-end and resize to 1-D array for better performance on traversing.
- `long int * flatValueMatrix`
Adding Value matrix end-to-end and resize to 1-D array for better performance on traversing.
- `int cudaBlocks`
- `int cudaThreads`
- `int iterationsPerKernelThread`
- `long int totalOutputPerSync`
- `long int totalOutputPerKernel`
- `int numberOfPartitions`
- `int cudaGridSize`
- `int cudaMemPerGrid`
- `long int cudaPerKernelAllocationSize`
- `int alternatingKernels`
- `unsigned long ** device_seeds`
- `cudaStream_t * cudastreams`
- `std::ifstream * datasetFile`
- `std::ofstream * modelSavefile`
Dataset file input of our system
- `std::ofstream * outputFile`
File to save model of our system
- `std::map<char, Node<char *> > nodes`
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- `Node<char *> * starterNode`
Starter Node of this model.
- `std::vector<Edge<char *> > edges`
A list of all edges in this model.

9.10.1 Detailed Description

Extension of `Markov::API::ModelMatrix` which is modified to run on GPU devices.

This implementation only supports Nvidia devices.

Class to flatten and reduce `Markov::Model` to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)

To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are synchronized and files are flushed every 50M operations.

Definition at line 19 of file `cudaModelMatrix.h`.

9.10.2 Member Function Documentation

9.10.2.1 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.10.2.2 AllocVRAMOutputBuffer()

```
__host__ char* Markov::API::CUDA::CUDAModelMatrix::AllocVRAMOutputBuffer (
    long int n,
    long int singleGenMaxLen,
    long int CUDAKernelGridSize,
    long int sizePerGrid ) [protected]
```

Allocate the output buffer for kernel operation.

TODO

Parameters

n	- Number of passwords to generate.
singleGenMaxLen	- maximum string length for a single generation
CUDAKernelGridSize	- Total number of grid members in CUDA kernel
sizePerGrid	- Size to allocate per grid member

Returns

pointer to the allocation on VRAM

9.10.2.3 Buff()

```
void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]
```

Buff expression of some characters in the model.

Parameters

<i>str</i>	A string containing all the characters to be buffed
<i>multiplier</i>	A constant value to buff the nodes with.
<i>bDontAdjustSelfEdges</i>	Do not adjust weights if target node is same as source node
<i>bDontAdjustExtendedLoops</i>	Do not adjust if both source and target nodes are in first parameter

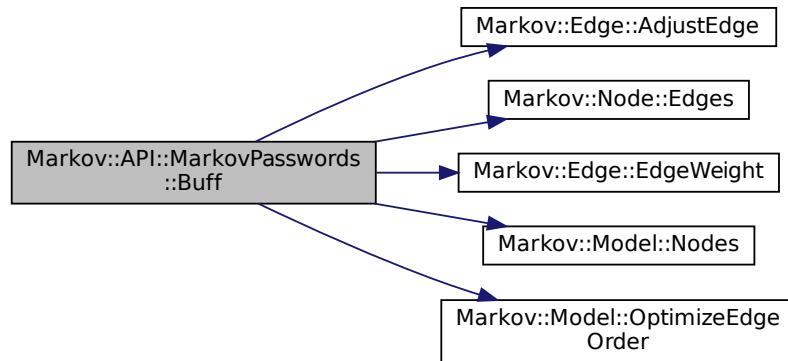
Definition at line 153 of file [markovPasswords.cpp](#).

```
00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos) {
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops) {
00165                     if(buffstr.find(repr)!= std::string::npos) {
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173         }
00174     }
00175     i++;
00176 }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.4 ConstructMatrix()

```
bool Markov::API::ModelMatrix::ConstructMatrix () [inherited]
```

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content. This will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

True if constructed. False if already constructed.

Definition at line 31 of file [modelMatrix.cpp](#).

```

00031
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
  
```

```

00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes) {
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights ();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight ();
00070                 this->edgeMatrix[i][j] = node->edgesV[j]->RightNode ()->NodeValue ();
00071             }
00072         }
00073         i++;
00074     }
00075     this->ready = true;
00076     return true;
00077 //this->DumpJSON();
00078 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::API::ModelMatrix::ModelMatrix](#)

[Markov::API::ModelMatrix::matrixSize](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), [Markov::Node< storageType >::NodeValue\(\)](#)

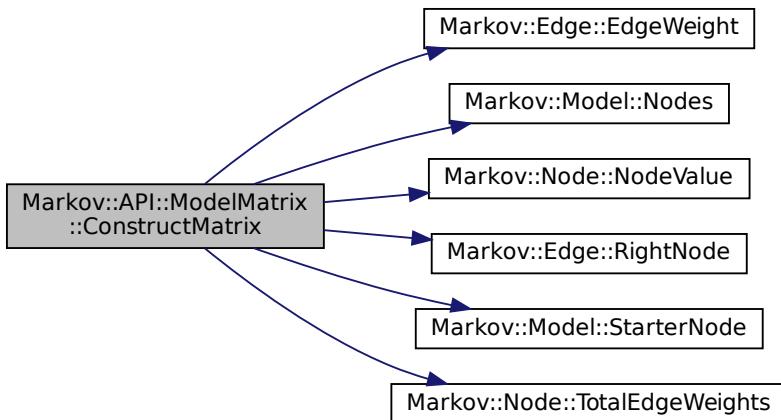
[Markov::API::ModelMatrix::ready](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::StarterNode\(\)](#)

[Markov::API::ModelMatrix::totalEdgeWeights](#), [Markov::Node< storageType >::TotalEdgeWeights\(\)](#), and [Markov::API::ModelMatrix::valueMatrix](#)

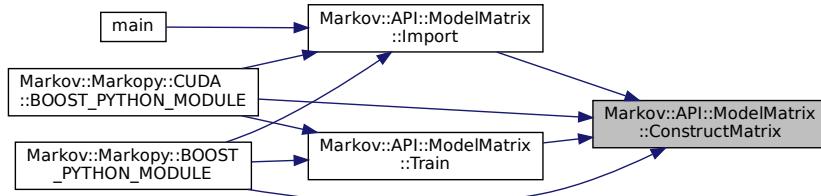
Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#),

[Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.5 CudaCheckNotifyErr()

```
__host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr (
    cudaError_t _status,
    const char * msg,
    bool bExit = true ) [static], [protected], [inherited]
```

Check results of the last operation on GPU.

Check the status returned from cudaMalloc/cudaMemcpy to find failures.

If a failure occurs, its assumed beyond redemption, and exited.

Parameters

_status	Cuda error status to check
msg	Message to print in case of a failure

Returns

0 if successful, 1 if failure. **Example output:**

```
char *da, a = "test";
cudaStatus = cudaMalloc((char **) &da, 5 * sizeof(char *));
CudaCheckNotifyErr(cudaStatus, "Failed to allocate VRAM for *da.\n");
```

Definition at line 32 of file [cudaDeviceController.cu](#).

```
00032
{
00033     if (_status != cudaSuccess) {
00034         std::cerr << "\033[1;31m" << msg << " -> " << cudaGetErrorString(_status) << "(" << _status <<
00035         ")" << "\033[0m" << "\n";
00036     if(bExit) {
00037         cudaDeviceReset();
00038         exit(1);
00039     }
00040 }
00041     return 0;
00042 }
```

9.10.2.6 CudaMalloc2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat (
    T ** dst,
    int row,
    int col ) [inline], [static], [protected], [inherited]
```

Malloc a 2D array in device space.

This function will allocate enough space on VRAM for flattened 2D array.

Parameters

<i>dst</i>	destination pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

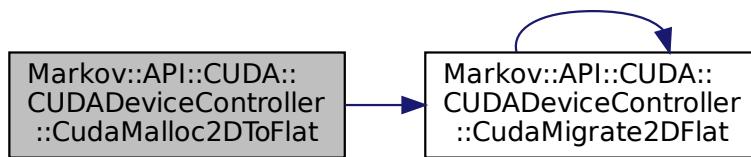
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess) {
    CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
}
```

Definition at line 75 of file [cudaDeviceController.h](#).

```
00075
00076     cudaError_t cudastatus = cudaMalloc((T **)dst, row*col*sizeof(T));
00077     CudaCheckNotifyErr(cudastatus, "cudaMalloc Failed.", false);
00078     return cudastatus;
00079 }
```

References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:

**9.10.2.7 CudaMemcpy2DToFlat()**

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat (
    T * dst,
    T ** src,
    int row,
    int col ) [inline], [static], [protected], [inherited]
```

Memcpy a 2D array in device space after flattening.

Resulting buffer will not be true 2D array.

Parameters

<i>dst</i>	destination pointer
<i>rc</i>	source pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

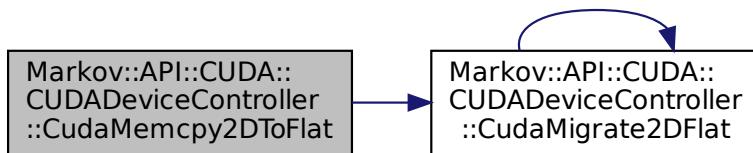
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
CudaCheckNotifyErr(cudastatus, " CudaMemcpy2DToFlat Failed.", false);
```

Definition at line 103 of file [cudaDeviceController.h](#).

```
00103
00104     T* tempbuf = new T[row*col];
00105     for(int i=0;i<row;i++){
00106         memcpy(&(tempbuf[row*i]), src[i], col);
00107     }
00108     return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109
00110 }
```

References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:

**9.10.2.8 CudaMigrate2DFlat()**

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat (
    T ** dst,
    T ** src,
    int row,
    int col ) [inline], [static], [protected], [inherited]
```

Both malloc and memcpy a 2D array into device VRAM.

Resulting buffer will not be true 2D array.

Parameters

<i>dst</i>	destination pointer
<i>rc</i>	source pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMigrate2DFlat<long int>(
    &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
CudaCheckNotifyErr(cudastatus, " Cuda failed to initialize value matrix row.");
```

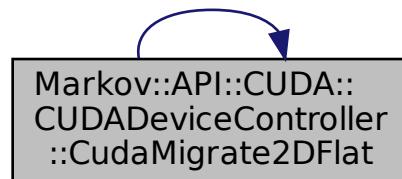
Definition at line 132 of file [cudaDeviceController.h](#).

```
00132
00133     cudaError_t cudastatus;
00134     cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135     if(cudastatus!=cudaSuccess) {
00136         CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
00137         return cudastatus;
00138     }
00139     cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140     CudaCheckNotifyErr(cudastatus, " CudaMemcpy2DToFlat Failed.", false);
00141     return cudastatus;
00142 }
```

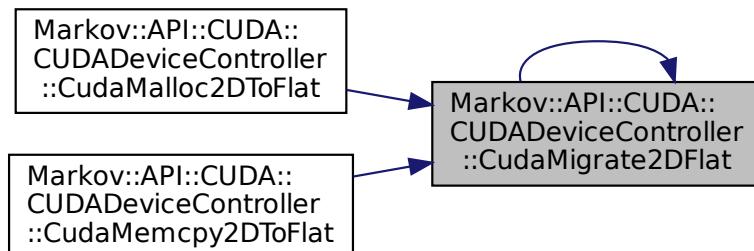
References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Referenced by [Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat\(\)](#), [Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat\(\)](#) and [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.9 DeallocateMatrix()

```
bool Markov::API::ModelMatrix::DeallocateMatrix ( ) [protected], [inherited]
Deallocate matrix and make it ready for re-construction.
```

Returns

True if deallocated. False if matrix was not initialized

Definition at line 81 of file [modelMatrix.cpp](#).

```
00081
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
```

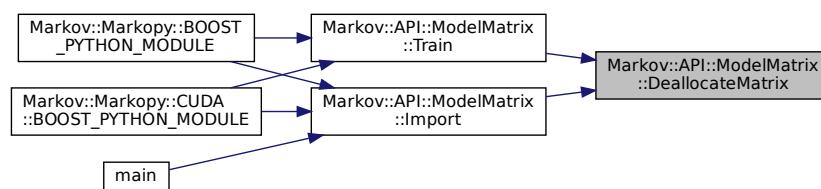
```

00085     for(int i=0;i<this->matrixSize;i++) {
00086         delete[] this->edgeMatrix[i];
00087     }
00088     delete[] this->edgeMatrix;
00089
00090     for(int i=0;i<this->matrixSize;i++) {
00091         delete[] this->valueMatrix[i];
00092     }
00093     delete[] this->valueMatrix;
00094
00095     this->matrixSize = -1;
00096     this->ready = false;
00097     return true;
00098 }
00099 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::API::ModelMatrix::ready](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the caller graph for this function:



9.10.2.10 DumpJSON()

`void Markov::API::ModelMatrix::DumpJSON() [inherited]`

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

Definition at line 101 of file [modelMatrix.cpp](#).

```

00101
00102
00103     std::cout << "{\n      \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++) {
00105         if(this->matrixIndex[i]=='"') std::cout << "\\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout << "\\\\\\"";
00107         else if(this->matrixIndex[i]==0) std::cout << "\\\\x00";
00108         else if(i==0) std::cout << "\\\\xff";
00109         else if(this->matrixIndex[i]=='\\n') std::cout << "\\n";
00110         else std::cout << this->matrixIndex[i];
00111     }
00112     std::cout <<
00113     "\",\n      \"edgemap\": {\n        \"\n
00114
00115         for(int i=0;i<this->matrixSize;i++) {
00116             if(this->matrixIndex[i]=='"') std::cout << "      \\\\\\"": [";
00117             else if(this->matrixIndex[i]=='\\') std::cout << "      \\\\\\\\"": [";
00118             else if(this->matrixIndex[i]==0) std::cout << "      \\\\\\\x00": [";
00119             else if(this->matrixIndex[i]<0) std::cout << "      \\\\\\\xf": [";
00120             else std::cout << "      \" " << this->matrixIndex[i] << "\": [";
00121             for(int j=0;j<this->matrixSize;j++) {
00122                 if(this->edgeMatrix[i][j]=='"') std::cout << "          \\\\\\"";
00123                 else if(this->edgeMatrix[i][j]=='\\') std::cout << "          \\\\\\\\"";
00124                 else if(this->edgeMatrix[i][j]==0) std::cout << "          \\\\\\\x00";
00125                 else if(this->edgeMatrix[i][j]<0) std::cout << "          \\\\\\\xf";
00126                 else if(this->matrixIndex[i]=='\n') std::cout << "          \\\\\\\n";
00127                 else std::cout << "          \" " << this->edgeMatrix[i][j] << "\": ";
00128                 if(j!=this->matrixSize-1) std::cout << ", ";
00129             }
00130         }
00131         std::cout << "      ],\n      \"weightmap\": {\n        \"\n
00132
00133         std::cout << "      \" weightmap\": {\n        \"\n
00134         for(int i=0;i<this->matrixSize;i++) {
```

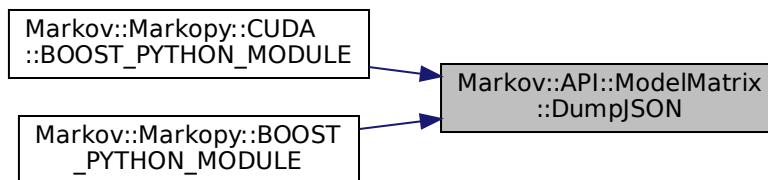
```

00137     if(this->matrixIndex[i]=='"') std::cout << "    \"\\\"\\\"": [";
00138     else if(this->matrixIndex[i]=='\\\\') std::cout << "    \"\\\\\\\\\\\\\\\\": [";
00139     else if(this->matrixIndex[i]==0) std::cout << "    \"\\\\\\\\x00\\\\": [";
00140     else if(this->matrixIndex[i]<0) std::cout << "    \"\\\\\\\\x{f}\\\\": [";
00141     else std::cout << "    \" " << this->matrixIndex[i] << "\": [";
00142
00143     for(int j=0;j<this->matrixSize;j++) {
00144         std::cout << this->valueMatrix[i][j];
00145         if(j!=this->matrixSize-1) std::cout << ", ";
00146     }
00147     std::cout << "], \n";
00148 }
00149 std::cout << " } \n} \n";
00150 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the caller graph for this function:



9.10.2.11 Edges()

```
std::vector<Edge<char *>>* Markov::Model< char >::Edges () [inline], [inherited]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.10.2.12 Export() [1/2]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
```

Open a file to export with filename, and call bool [Model::Export](#) with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```

00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.10.2.13 Export() [2/2]

```
bool Markov::Model< char >::Export (
    std::ofstream * f )  [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file [model.h](#).

```
00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <
00294         "\n";
00295
00296     return true;
00297 }
```

9.10.2.14 FastRandomWalk() [1/3]

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen,
    int maxLen,
    bool bFileIO,
    bool bInfinite )
```

Random walk on the Matrix-reduced [Markov::Model](#).

TODO

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

Definition at line 58 of file [cudaModelMatrix.cu](#).

```
00058
00059     cudaDeviceProp prop;
00060     int device=0;
00061     cudaGetDeviceProperties(&prop, device);
00062     cudaChooseDevice(&device, &prop);
00063     //std::cout << "Flattening matrix." << std::endl;
00064     this->FlattenMatrix();
00065     //std::cout << "Migrating matrix." << std::endl;
00066     this->MigrateMatrix();
00067     //std::cout << "Migrated matrix." << std::endl;
00068     std::ofstream wordlist;
```

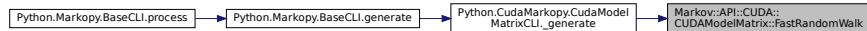
```

00069     if(bFileIO)
00070         wordlist.open(wordlistFileName);
00071
00072
00073     cudaBlocks = 1024;
00074     cudaThreads = 256;
00075     iterationsPerKernelThread = 100;
00076     alternatingKernels = 2;
00077     totalOutputPerKernel = (long int)cudaBlocks*(long int)cudaThreads*iterationsPerKernelThread;
00078     totalOutputPerSync= totalOutputPerKernel*alternatingKernels;
00079     numberofPartitions = n/totalOutputPerSync;
00080     cudaGridSize = cudaBlocks*cudaThreads;
00081     cudaMemPerGrid = (maxLength+2)*iterationsPerKernelThread;
00082     cudaPerKernelAllocationSize = cudaGridSize*cudaMemPerGrid;
00083     this->prepKernelMemoryChannel(alternatingKernels);
00084
00085     unsigned long int leftover = n - (totalOutputPerSync*numberofPartitions);
00086
00087     if(bInfinite && !numberofPartitions) numberofPartitions=5;
00088     std::cerr << cudaPerKernelAllocationSize << "\n";
00089
00090     if(n%totalOutputPerSync) std::cerr << "For optimization, request outputs multiples of " <<
00091     totalOutputPerSync << ".\n";
00092
00093     //start kernelID 1
00094     this->LaunchAsyncKernel(1, minLen, maxLength);
00095
00096     for(int i=1;i<numberofPartitions;i++){
00097         if(bInfinite) i=0;
00098
00099         //wait kernelID1 to finish, and start kernelID 0
00100         cudaStreamSynchronize(this->cudastreams[1]);
00101         this->LaunchAsyncKernel(0, minLen, maxLength);
00102
00103         //start memcpy from kernel 1 (block until done)
00104         this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00105
00106         //wait kernelID 0 to finish, then start kernelID1
00107         cudaStreamSynchronize(this->cudastreams[0]);
00108         this->LaunchAsyncKernel(1, minLen, maxLength);
00109
00110         //start memcpy from kernel 0 (block until done)
00111         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00112
00113     }
00114
00115     //wait kernelID1 to finish, and start kernelID 0
00116     cudaStreamSynchronize(this->cudastreams[1]);
00117     this->LaunchAsyncKernel(0, minLen, maxLength);
00118     this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00119     cudaStreamSynchronize(this->cudastreams[0]);
00120     this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00121
00122     if(!leftover) return;
00123     alternatingKernels=1;
00124     std::cerr << "Remaining line count (" << leftover << ") is lower than partition. Adjusting CUDA
00125     workload..\n";
00126     this->iterationsPerKernelThread = leftover/cudaGridSize;
00127     this->LaunchAsyncKernel(0, minLen, maxLength);
00128     cudaStreamSynchronize(this->cudastreams[0]);
00129     this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00130
00131     leftover -= this->iterationsPerKernelThread*cudaGridSize;
00132     if(!leftover) return;
00133
00134     std::cerr << "Remaining line count (" << leftover << ") is lower than minimum possible. Handing
00135     over to CPU generation.\n";
00136     this->iterationsPerKernelThread = leftover/cudaGridSize;
00137
00138     leftover -= this->iterationsPerKernelThread;
00139
00140     if(!leftover) return;
00141     std::cerr << "Remaining " << leftover << " lines are absolutely not worth printing.\n";
00142     Markov::API::ModelMatrix::ConstructMatrix();
00143     Markov::API::ModelMatrix::FastRandomWalk(leftover, &wordlist, minLen, maxLength, 1, bFileIO);
00144
00145 }
```

References [alternatingKernels](#), [cudaBlocks](#), [cudaGridSize](#), [cudaMemPerGrid](#), [cudaPerKernelAllocationSize](#), [cudaThreads](#), [iterationsPerKernelThread](#), [numberofPartitions](#), [totalOutputPerKernel](#), and [totalOutputPerSync](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI::_generate\(\)](#).

Here is the caller graph for this function:



9.10.2.15 FastRandomWalk() [2/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

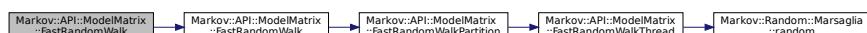
Definition at line 217 of file [modelMatrix.cpp](#).

```
00217
{
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

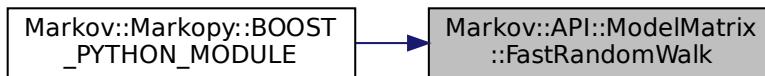
References [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.16 FastRandomWalk() [3/3]

```

int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [protected], [inherited]

```

[Random walk on the Matrix-reduced Markov::Model.](#)

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by 99.6.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```

Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);

```

[Definition at line 204 of file modelMatrix.cpp.](#)

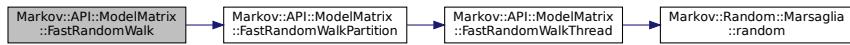
```

00204
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
00209     threads);
00210     else{
00211         int numberOfPartitions = n/50000000ull;
00212         for(int i=0;i<numberOfPartitions;i++)
00213             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
00214             threads);
00215     }
00216     return 0;
00217 }
```

[References](#) [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

[Referenced by](#) [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.17 FastRandomWalkPartition()

```
void Markov::API::ModelMatrix::FastRandomWalkPartition (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    bool bFileIO,
    int threads ) [protected], [inherited]
```

A single partition of FastRandomWalk event.

Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation
- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 225 of file [modelMatrix.cpp](#).

```
00225
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231 }
```

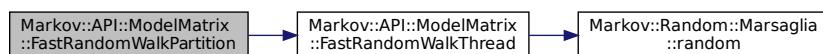
```

00232     int id = 0;
00233     for(int i=0;i<threads;i++) {
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00235                                         mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00236         id++;
00237     }
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00239                                         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00240     for(int i=0;i<threads;i++) {
00241         threadsV[i]~>join();
00242     }
00243 }
```

References [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.18 FastRandomWalkThread()

```

void Markov::API::ModelMatrix::FastRandomWalkThread (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    int id,
    bool bFileIO ) [protected], [inherited]
```

A single thread of a single partition of [FastRandomWalk](#).

A [FastRandomWalkPartition](#) will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>id</i>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 153 of file [modelMatrix.cpp](#).

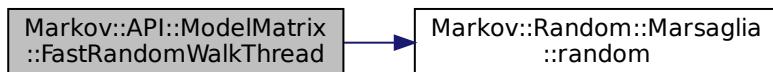
00153

```

00154     if(n==0)  return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179             if (len >= maxLen) break;
00180             else if ((next < 0) && (len < minLen)) continue;
00181             else if (next < 0) break;
00182             cur = next;
00183             res[bufferctr + len++] = cur;
00184         }
00185     }
00186     res[bufferctr + len++] = '\n';
00187     bufferctr+=len;
00188
00189 }
00190 if(bFileIO) {
00191     mlock->lock();
00192     *wordlist << res;
00193     mlock->unlock();
00194 }else{
00195     mlock->lock();
00196     std::cout << res;
00197     mlock->unlock();
00198 }
00199 delete res;
00200
00201 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::Random::Marsaglia::random\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#). Referenced by [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.19 FlattenMatrix()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix ( )
```

Flatten migrated matrix from 2d to 1d.

Definition at line 261 of file [cudaModelMatrix.cu](#).

```
00261     this->flatEdgeMatrix = new char[this->matrixSize*this->matrixSize];
00262     this->flatValueMatrix = new long int[this->matrixSize*this->matrixSize];
00263     for(int i=0;i<this->matrixSize;i++){
00264         memcpy(&this->flatEdgeMatrix[i*this->matrixSize], this->edgeMatrix[i], this->matrixSize );
00265         memcpy(&this->flatValueMatrix[i*this->matrixSize], this->valueMatrix[i],
00266             this->matrixSize*sizeof(long int) );
00267     }
00268 }
```

References [flatEdgeMatrix](#), [flatValueMatrix](#), and [Markov::API::ModelMatrix::matrixSize](#).

9.10.2.20 GatherAsyncKernelOutput()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput (
    int kernelID,
    bool bFileIO,
    std::ofstream & wordlist ) [protected]
```

Definition at line 180 of file [cudaModelMatrix.cu](#).

```
00180     {
00181         {
00182             cudaMemcpy(this->outputBuffer[kernelID],this->device_outputBuffer[kernelID],cudaPerKernelAllocationSize,
00183             cudaMemcpyDeviceToHost);
00184             //std::cerr << "Kernel" << kernelID << " output copied\n";
00185             if(bFileIO){
00186                 for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00187                     wordlist << &this->outputBuffer[kernelID][j];
00188                 }
00189             }else{
00190                 for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00191                     std::cout << &this->outputBuffer[kernelID][j];
00192                 }
00193             }
00194         }
00195     }
```

References [cudaMemPerGrid](#), [cudaPerKernelAllocationSize](#), and [outputBuffer](#).

9.10.2.21 Generate()

```
void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]
```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. [FastRandomWalk](#) will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

```
00118     {
00119     char* res;
```

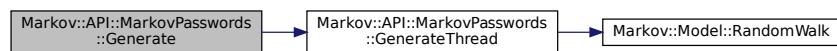
```

00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread> threadsV;
00127     for(int i=0;i<threads;i++) {
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129             &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130     }
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]~>join();
00133         delete threadsV[i];
00134     }
00135     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00136 }
00137
00138 }
```

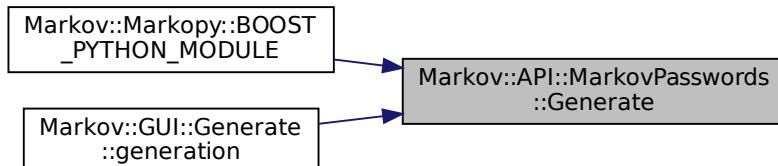
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.22 GenerateThread()

```

void Markov::API::MarkovPasswords::GenerateThread (
    std::mutex * outputLock,
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen,
    int maxLen ) [private], [inherited]
```

A single thread invoked by the Generate function.

DEPRECATED: See [Markov::API::MatrixModel::FastRandomWalkThread](#) for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Parameters

<i>outputLock</i>	- shared mutex lock to lock during output operation. Prevents race condition on write.
<i>n</i>	number of lines to be generated by this thread
<i>wordlist</i>	wordlistfile

Parameters

<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate

Definition at line 140 of file [markovPasswords.cpp](#).

```

00140
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist << res << "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

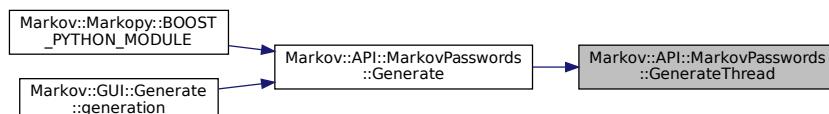
References [Markov::Model< NodeStorageType >::RandomWalk\(\)](#).

Referenced by [Markov::API::MarkovPasswords::Generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.23 Import() [1/2]

```
void Markov::API::ModelMatrix::Import (
    const char * filename ) [inherited]
```

Open a file to import with filename, and call bool [Model::Import](#) with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.

Definition at line 19 of file [modelMatrix.cpp](#).

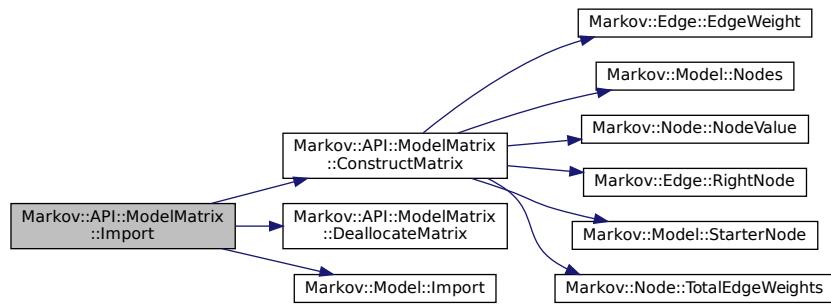
```

00019
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

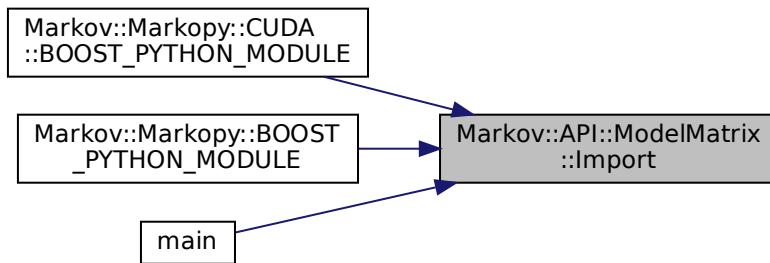
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.24 Import() [2/2]

```
bool Markov::Model< char >::Import (
    std::ifstream * f )  [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the [wiki](#) and [github](#) [readme](#) pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file [model.h](#).

```
00216
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
```

```

00224     //std::cout << "cell: " << cell << std::endl;
00225     src = cell[0];
00226     target = cell[cell.length() - 1];
00227     char* j;
00228     oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00229     //std::cout << oc << "\n";
00230     Markov::Node<NodeStorageType>* srcN;
00231     Markov::Node<NodeStorageType>* targetN;
00232     Markov::Edge<NodeStorageType>* e;
00233     if (this->nodes.find(src) == this->nodes.end()) {
00234         srcN = new Markov::Node<NodeStorageType>(src);
00235         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00236         //std::cout << "Creating new node at start.\n";
00237     }
00238     else {
00239         srcN = this->nodes.find(src)->second;
00240     }
00241
00242     if (this->nodes.find(target) == this->nodes.end()) {
00243         targetN = new Markov::Node<NodeStorageType>(target);
00244         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00245         //std::cout << "Creating new node at end.\n";
00246     }
00247     else {
00248         targetN = this->nodes.find(target)->second;
00249     }
00250     e = srcN->Link(targetN);
00251     e->AdjustEdge(oc);
00252     this->edges.push_back(e);
00253
00254     //std::cout << int(srcN->nodeValue()) << " --> " << e->EdgeWeight() << "--> " <<
00255     int(targetN->nodeValue()) << "\n";
00256
00257 }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.10.2.25 LaunchAsyncKernel()

```

__host__ void Markov::API::CUDA::CUDAModelMatrix::LaunchAsyncKernel (
    int kernelID,
    int minLen,
    int maxLen ) [protected]
```

Definition at line 171 of file [cudaModelMatrix.cu](#).

```

00171
00172
00173     //if(kernelID == 0); // cudaStreamSynchronize(this->cudastreams[2]);
00174     //else cudaStreamSynchronize(this->cudastreams[kernelID-1]);
00175     FastRandomWalkCUDAKernel<<cudaBlocks,cudaThreads,0,
00176     this->cudastreams[kernelID]>>(iterationsPerKernelThread, minLen, maxLen,
00177     this->device_outputBuffer[kernelID], this->device_matrixIndex,
00178     this->device_totalEdgeWeights, this->device_valueMatrix, this->device_edgeMatrix,
00179     this->matrixSize, cudaMemPerGrid, this->device_seeds[kernelID]);
00180     //std::cerr << "Started kernel" << kernelID << "\n";
00181 }
```

9.10.2.26 ListCudaDevices()

```

__host__ void Markov::API::CUDA::CUDADeviceController::ListCudaDevices () [static], [inherited]
List CUDA devices in the system.
```

This function will print details of every CUDA capable device in the system.

Example output:

```

Device Number: 0
Device name: GeForce RTX 2070
Memory Clock Rate (KHz): 7001000
Memory Bus Width (bits): 256
Peak Memory Bandwidth (GB/s): 448.064
Max Linear Threads: 1024
```

Definition at line 16 of file [cudaDeviceController.cu](#).

```

00016
00017     host.                                         { //list cuda Capable devices on
00018         int nDevices;
00019         cudaGetDeviceCount(&nDevices);
```

```

00019     for (int i = 0; i < nDevices; i++) {
00020         cudaDeviceProp prop;
00021         cudaGetDeviceProperties(&prop, i);
00022         std::cerr << "Device Number: " << i << "\n";
00023         std::cerr << "Device name: " << prop.name << "\n";
00024         std::cerr << "Memory Clock Rate (KHz): " << prop.memoryClockRate << "\n";
00025         std::cerr << "Memory Bus Width (bits): " << prop.memoryBusWidth << "\n";
00026         std::cerr << "Peak Memory Bandwidth (GB/s): " << 2.0 * prop.memoryClockRate *
00027             (prop.memoryBusWidth / 8) / 1.0e6 << "\n";
00028         std::cerr << "Max Linear Threads: " << prop.maxThreadsDim[0] << "\n";
00029     }
00030 }
```

9.10.2.27 MigrateMatrix()

`__host__ void Markov::API::CUDAModelMatrix::MigrateMatrix ()`

Migrate the class members to the VRAM.

Cannot be used without calling [Markov::API::ModelMatrix::ConstructMatrix](#) at least once. This function will manage the memory allocation and data transfer from CPU RAM to GPU VRAM.

Newly allocated VRAM pointers are set in the class member variables.

Definition at line 20 of file [cudaModelMatrix.cu](#).

```

00020     cudaError_t cudastatus;
00021
00022     {
00023         cudastatus = cudaMalloc((char**)&(this->device_matrixIndex),
00024             this->matrixSize*sizeof(char));
00025         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_matrixIndex.");
00026
00027         cudastatus = cudaMalloc((long int **)&(this->device_totalEdgeWeights),
00028             this->matrixSize*sizeof(long int));
00029         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_totalEdgeWeights.");
00030
00031         cudastatus = cudaMemcpy(this->device_matrixIndex, this->matrixIndex,
00032             this->matrixSize*sizeof(char), cudaMemcpyHostToDevice);
00033         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Index)");
00034
00035         cudastatus = cudaMemcpy(this->device_totalEdgeWeights, this->totalEdgeWeights,
00036             this->matrixSize*sizeof(long int), cudaMemcpyHostToDevice);
00037         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Total Edge Values)");
00038
00039         cudastatus = CudaMigrate2DFlat<char>(
00040             &(this->device_edgeMatrix), this->edgeMatrix, this->matrixSize, this->matrixSize);
00041         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize edge matrix.");
00042
00043         cudastatus = CudaMigrate2DFlat<long int>(
00044             &(this->device_valueMatrix), this->valueMatrix, this->matrixSize, this->matrixSize);
00045         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize value matrix row.");
00046     }
```

9.10.2.28 Nodes()

`std::map<char , Node<char >*>* Markov::Model< char >::Nodes () [inline], [inherited]`

Return starter [Node](#).

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.10.2.29 OpenDatasetFile()

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename ) [inherited]
```

Open dataset file and return the ifstream pointer.

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

`ifstream*` to the the dataset file

Definition at line 51 of file [markovPasswords.cpp](#).

```

00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References [Markov::Model< NodeStorageType >::Import\(\)](#).

Here is the call graph for this function:

**9.10.2.30 OptimizeEdgeOrder()**

`void Markov::Model< char >::OptimizeEdgeOrder [inherited]`

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file [model.h](#).

```

00265
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
00269             Edge<NodeStorageType> *rhs) ->bool {
00270             return lhs->EdgeWeight () > rhs->EdgeWeight ();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         //    std::cout << x.second->edgesV[i]->EdgeWeight () << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00278 }
```

9.10.2.31 prepKernelMemoryChannel()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel (
    int number_of_streams ) [protected]
```

Definition at line 145 of file [cudaModelMatrix.cu](#).

```

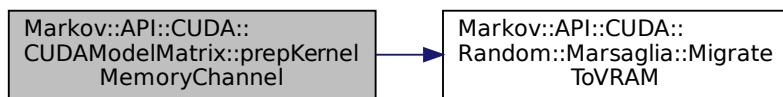
00145
00146
00147     this->cudastreams = new cudaStream_t[number_of_streams];
00148     for(int i=0;i<number_of_streams;i++)
00149         cudaStreamCreate(&this->cudastreams[i]);
00150
00151     this->outputBuffer = new char*[number_of_streams];
00152     for(int i=0;i<number_of_streams;i++)
00153         this->outputBuffer[i]= new char[cudaPerKernelAllocationSize];
00154 }
```

```

00155     cudaError_t cudastatus;
00156     this-> device_outputBuffer = new char*[numberOfStreams];
00157     for(int i=0;i<numberOfStreams;i++){
00158         cudastatus = cudaMalloc((char**)&(device_outputBuffer[i]),
00159             cudaPerKernelAllocationSize);
00160         CudaCheckNotifyErr(cudastatus, "Failed to establish memory channel. Possibly out of
VRAM?");
00161     }
00162     this-> device_seeds = new unsigned long*[numberOfStreams];
00163     for(int i=0;i<numberOfStreams;i++){
00164         Markov::API::CUDA::Random::Marsaglia *MEarr = new
Markov::API::CUDA::Random::Marsaglia[cudaGridSize];
00165         this->device_seeds[i] = Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(MEarr,
cudaGridSize);
00166         delete[] MEarr;
00167     }
00168 }
00169 }
```

References [cudaGridSize](#), [cudaPerKernelAllocationSize](#), [device_outputBuffer](#), [device_seeds](#), [Markov::API::CUDA::Random::Marsaglia](#) and [outputBuffer](#).

Here is the call graph for this function:



9.10.2.32 RandomWalk()

```

char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```

Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate

Parameters

<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```

00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319         else if (temp_node == NULL) {
00320             break;
00321         }
00322         n = temp_node;
00323         buffer[len++] = n->NodeValue();
00324     }
00325
00326     //null terminate the string
00327     buffer[len] = 0x00;
00328
00329     //do something with the generated string
00330     return buffer; //for now
00331
00332
00333
00334 }
```

9.10.2.33 Save()

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
```

Export model to file.

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

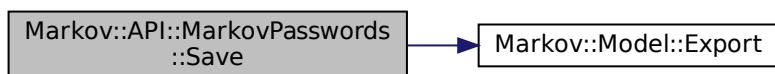
`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

```
00106     std::ofstream* exportFile;
00107
00108     std::ofstream newFile(filename);
00109     exportFile = &newFile;
00110
00111     this->Export(exportFile);
00112
00113     return exportFile;
00114
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:

**9.10.2.34 StarterNode()**

`Node< char >* Markov::Model< char >::StarterNode()` [inline], [inherited]
Return starter [Node](#).

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.10.2.35 Train()

```
void Markov::API::ModelMatrix::Train (
    const char * datasetFileName,
    char delimiter,
    int threads) [inherited]
```

Train the model with the dataset file.

Parameters

<code>datasetFileName</code>	- <code>Ifstream*</code> to the dataset. If null, use class member
<code>delimiter</code>	- a character, same as the delimiter in dataset content
<code>threads</code>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

Construct the matrix when done.

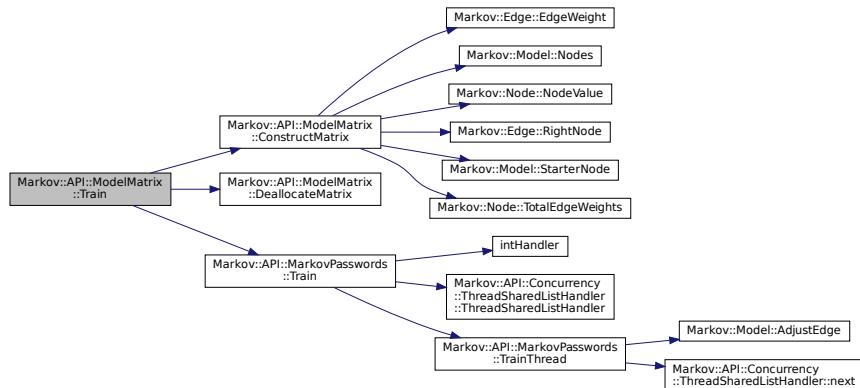
Definition at line 25 of file [modelMatrix.cpp](#).

```
00025
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName, delimiter, threads);
00028     this->ConstructMatrix();
00029 }
```

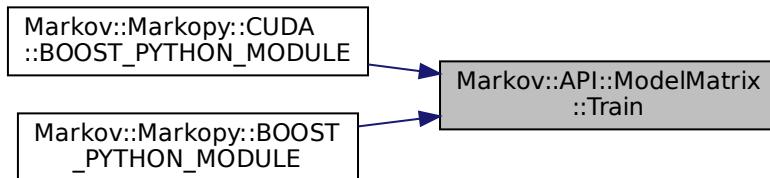
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and

[Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the call graph for this function:



Here is the caller graph for this function:



9.10.2.36 TrainThread()

```
void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private], [inherited]
```

A single thread invoked by the Train function.

Parameters

<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

Definition at line 85 of file [markovPasswords.cpp](#).

```
00085
00086     char format_str[] = "%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
```

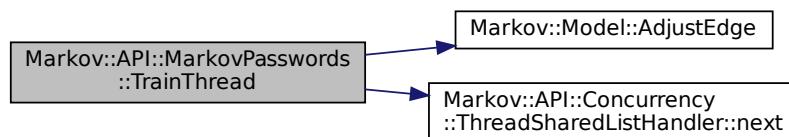
```

00096     sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length() + 5); //<== changed format_str to->
00097     "%ld,%s"
00098     sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100     this->AdjustEdge((const char*)linebuf, oc);
00101     delete linebuf;
00102 }
00103 }
```

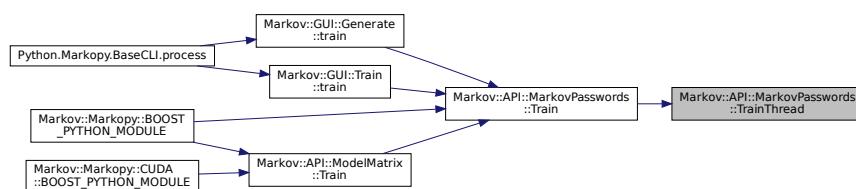
References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), [keepRunning](#), and [Markov::API::Concurrency::ThreadSharedListHandler::next](#).

Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.10.3 Member Data Documentation

9.10.3.1 alternatingKernels

`int Markov::API::CUDA::CUDAModelMatrix::alternatingKernels [private]`

Definition at line [135](#) of file [cudaModelMatrix.h](#).

Referenced by [FastRandomWalk\(\)](#).

9.10.3.2 cudaBlocks

`int Markov::API::CUDA::CUDAModelMatrix::cudaBlocks [private]`

Definition at line [125](#) of file [cudaModelMatrix.h](#).

Referenced by [FastRandomWalk\(\)](#).

9.10.3.3 cudaGridSize

`int Markov::API::CUDA::CUDAModelMatrix::cudaGridSize [private]`

Definition at line [131](#) of file [cudaModelMatrix.h](#).

Referenced by [FastRandomWalk\(\)](#), and [prepKernelMemoryChannel\(\)](#).

9.10.3.4 cudaMemPerGrid

```
int Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid [private]  
Definition at line 132 of file cudaModelMatrix.h.  
Referenced by FastRandomWalk\(\), and GatherAsyncKernelOutput\(\).
```

9.10.3.5 cudaPerKernelAllocationSize

```
long int Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize [private]  
Definition at line 133 of file cudaModelMatrix.h.  
Referenced by FastRandomWalk\(\), GatherAsyncKernelOutput\(\), and prepKernelMemoryChannel\(\).
```

9.10.3.6 cudastreams

```
cudaStream_t* Markov::API::CUDA::CUDAModelMatrix::cudastreams [private]  
Definition at line 139 of file cudaModelMatrix.h.
```

9.10.3.7 cudaThreads

```
int Markov::API::CUDA::CUDAModelMatrix::cudaThreads [private]  
Definition at line 126 of file cudaModelMatrix.h.  
Referenced by FastRandomWalk\(\).
```

9.10.3.8 datasetFile

```
std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]  
Definition at line 123 of file markovPasswords.h.
```

9.10.3.9 device_edgeMatrix

```
char* Markov::API::CUDA::CUDAModelMatrix::device_edgeMatrix [private]  
VRAM Address pointer of edge matrix (from modelMatrix.h)  
Definition at line 88 of file cudaModelMatrix.h.
```

9.10.3.10 device_matrixIndex

```
char* Markov::API::CUDA::CUDAModelMatrix::device_matrixIndex [private]  
VRAM Address pointer of matrixIndex (from modelMatrix.h)  
Definition at line 98 of file cudaModelMatrix.h.
```

9.10.3.11 device_outputBuffer

```
char** Markov::API::CUDA::CUDAModelMatrix::device_outputBuffer [private]  
RandomWalk results in device.  
Definition at line 108 of file cudaModelMatrix.h.  
Referenced by prepKernelMemoryChannel\(\).
```

9.10.3.12 device_seeds

```
unsigned long** Markov::API::CUDA::CUDAModelMatrix::device_seeds [private]  
Definition at line 137 of file cudaModelMatrix.h.  
Referenced by prepKernelMemoryChannel\(\).
```

9.10.3.13 device_totalEdgeWeights

long int* Markov::API::CUDA::CUDAModelMatrix::device_totalEdgeWeights [private]
 VRAM Address pointer of total edge weights (from [modelMatrix.h](#))
 Definition at line 103 of file [cudaModelMatrix.h](#).

9.10.3.14 device_valueMatrix

long int* Markov::API::CUDA::CUDAModelMatrix::device_valueMatrix [private]
 VRAM Address pointer of value matrix (from [modelMatrix.h](#))
 Definition at line 93 of file [cudaModelMatrix.h](#).

9.10.3.15 edgeMatrix

char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]
 2-D Character array for the edge Matrix (The characters of Nodes)
 Definition at line 175 of file [modelMatrix.h](#).
 Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#),
[Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.10.3.16 edges

std::vector<[Edge](#)<char >>* Markov::Model< char >::edges [private], [inherited]
 A list of all edges in this model.
 Definition at line 204 of file [model.h](#).

9.10.3.17 flatEdgeMatrix

char* Markov::API::CUDA::CUDAModelMatrix::flatEdgeMatrix [private]
 Adding [Edge](#) matrix end-to-end and resize to 1-D array for better perfomance on traversing.
 Definition at line 118 of file [cudaModelMatrix.h](#).
 Referenced by [FlattenMatrix\(\)](#).

9.10.3.18 flatValueMatrix

long int* Markov::API::CUDA::CUDAModelMatrix::flatValueMatrix [private]
 Adding Value matrix end-to-end and resize to 1-D array for better perfomance on traversing.
 Definition at line 123 of file [cudaModelMatrix.h](#).
 Referenced by [FlattenMatrix\(\)](#).

9.10.3.19 iterationsPerKernelThread

int Markov::API::CUDA::CUDAModelMatrix::iterationsPerKernelThread [private]
 Definition at line 127 of file [cudaModelMatrix.h](#).
 Referenced by [FastRandomWalk\(\)](#).

9.10.3.20 matrixIndex

char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]
 to hold the Matrix index (To hold the orders of 2-D arrays')

Definition at line 190 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.10.3.21 matrixSize

```
int Markov::API::ModelMatrix::matrixSize [protected], [inherited]
to hold Matrix size
```

Definition at line 185 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#), and [FlattenMatrix\(\)](#).

9.10.3.22 modelSavefile

```
std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]
Dataset file input of our system
```

Definition at line 124 of file [markovPasswords.h](#).

9.10.3.23 nodes

```
std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
```

Definition at line 193 of file [model.h](#).

9.10.3.24 numberOfPartitions

```
int Markov::API::CUDA::CUDAModelMatrix::numberOfPartitions [private]
Definition at line 130 of file cudaModelMatrix.h.
```

Referenced by [FastRandomWalk\(\)](#).

9.10.3.25 outputBuffer

```
char** Markov::API::CUDA::CUDAModelMatrix::outputBuffer [private]
RandomWalk results in host.
```

Definition at line 113 of file [cudaModelMatrix.h](#).

Referenced by [GatherAsyncKernelOutput\(\)](#), and [prepKernelMemoryChannel\(\)](#).

9.10.3.26 outputFile

```
std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]
File to save model of our system
```

Definition at line 125 of file [markovPasswords.h](#).

9.10.3.27 ready

```
bool Markov::API::ModelMatrix::ready [protected], [inherited]
```

True when matrix is constructed. False if not.

Definition at line 200 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::ModelMatrix\(\)](#).

9.10.3.28 starterNode

`Node<char *>* Markov::Model< char >::starterNode [private], [inherited]`

Starter Node of this model.

Definition at line 198 of file [model.h](#).

9.10.3.29 totalEdgeWeights

`long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]`

Array of the Total Edge Weights.

Definition at line 195 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.10.3.30 totalOutputPerKernel

`long int Markov::API::CUDA::CUDAModelMatrix::totalOutputPerKernel [private]`

Definition at line 129 of file [cudaModelMatrix.h](#).

Referenced by [FastRandomWalk\(\)](#).

9.10.3.31 totalOutputPerSync

`long int Markov::API::CUDA::CUDAModelMatrix::totalOutputPerSync [private]`

Definition at line 128 of file [cudaModelMatrix.h](#).

Referenced by [FastRandomWalk\(\)](#).

9.10.3.32 valueMatrix

`long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]`

2-d Integer array for the value Matrix (For the weights of Edges)

Definition at line 180 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

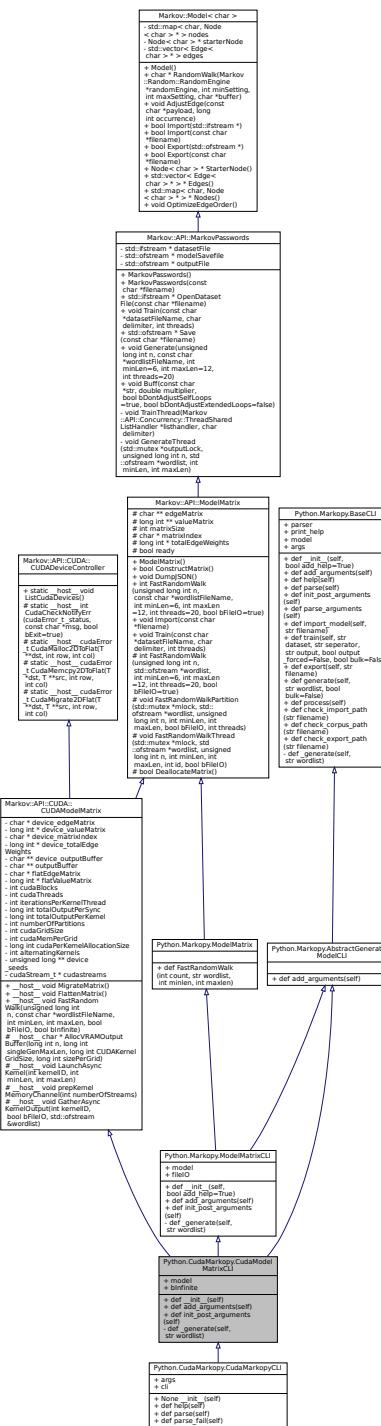
The documentation for this class was generated from the following files:

- [Markopy/CudaMarkovAPI/src/cudaModelMatrix.h](#)
- [Markopy/CudaMarkovAPI/src/cudaModelMatrix.cu](#)

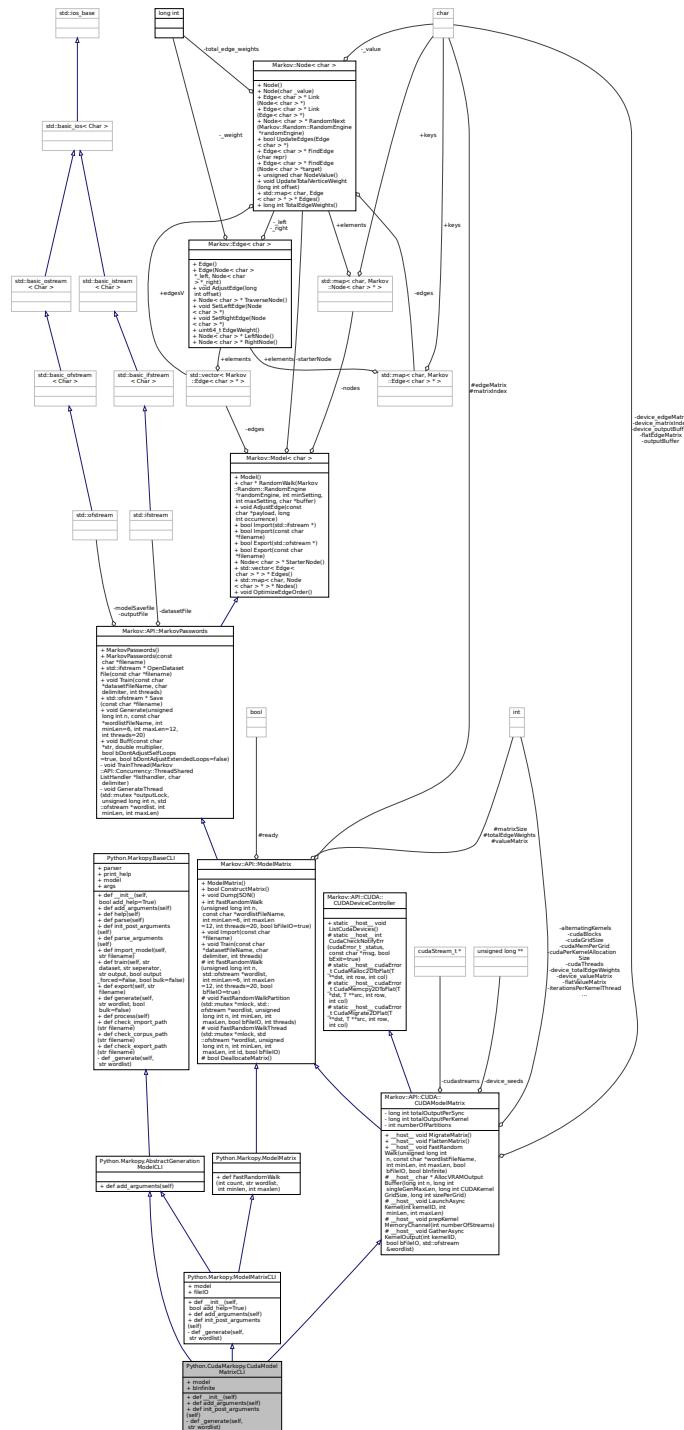
9.11 Python.CudaMarkopy.CudaModelMatrixCLI Class Reference

Python CLI wrapper for CudaModelMatrix.

Inheritance diagram for Python.CudaMarkopy.CudaModelMatrixCLI:



Collaboration diagram for Python.CudaMarkopy.CudaModelMatrixCLI:



Public Member Functions

- def `__init__` (self)
 - def `add_arguments` (self)
 - def `init_post_arguments` (self)
 - def `help` (self)
 - def `parse` (self)
 - def `parse_arguments` (self)

- def **import_model** (self, str filename)
Import a model file.
- def **train** (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)
Train a model via CLI parameters.
- def **export** (self, str filename)
Export model to a file.
- def **generate** (self, str wordlist, bool bulk=False)
Generate strings from the model.
- def **process** (self)
Process parameters for operation.
- def **FastRandomWalk** (int count, str wordlist, int minlen, int maxlen)
- int **FastRandomWalk** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)
Random walk on the Matrix-reduced [Markov::Model](#).
- bool **ConstructMatrix** ()
Construct the related Matrix data for the model.
- void **DumpJSON** ()
Debug function to dump the model to a JSON file.
- void **Import** (const char *filename)
Open a file to import with filename, and call bool [Model::Import](#) with std::ifstream.
- bool **Import** (std::ifstream *)
Import a file to construct the model.
- void **Train** (const char *datasetFileName, char delimiter, int threads)
Train the model with the dataset file.
- std::ifstream * **OpenDatasetFile** (const char *filename)
Open dataset file and return the ifstream pointer.
- std::ofstream * **Save** (const char *filename)
Export model to file.
- void **Generate** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)
Call [Markov::Model::RandomWalk](#) n times, and collect output.
- void **Buff** (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)
Buff expression of some characters in the model.
- char * **RandomWalk** ([Markov::Random::RandomEngine](#) *randomEngine, int minSetting, int maxSetting, char *buffer)
Do a random walk on this model.
- void **AdjustEdge** (const char *payload, long int occurrence)
Adjust the model with a single string.
- bool **Export** (std::ofstream *)
Export a file of the model.
- bool **Export** (const char *filename)
Open a file to export with filename, and call bool [Model::Export](#) with std::ofstream.
- Node< char > * **StarterNode** ()
Return starter Node.
- std::vector< Edge< char > * > * **Edges** ()
Return a vector of all the edges in the model.
- std::map< char, Node< char > * > * **Nodes** ()
Return starter Node.
- void **OptimizeEdgeOrder** ()
Sort edges of all nodes in the model ordered by edge weights.

- def `help` (self)
- def `parse` (self)
- def `parse_arguments` (self)
- def `import_model` (self, str filename)

Import a model file.
- def `train` (self, str dataset, str separator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def `export` (self, str filename)

Export model to a file.
- def `generate` (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def `process` (self)

Process parameters for operation.
- __host__ void `MigrateMatrix` ()

Migrate the class members to the VRAM.
- __host__ void `FlattenMatrix` ()

Flatten migrated matrix from 2d to 1d.
- __host__ void `FastRandomWalk` (unsigned long int n, const char *wordlistFileName, int minLen, int maxLen, bool bFileIO, bool bInfinite)

Random walk on the Matrix-reduced `Markov::Model`.
- int `FastRandomWalk` (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced `Markov::Model`.
- bool `ConstructMatrix` ()

Construct the related Matrix data for the model.
- void `DumpJSON` ()

Debug function to dump the model to a JSON file.
- void `Import` (const char *filename)

Open a file to import with filename, and call bool `Model::Import` with std::ifstream.
- bool `Import` (std::ifstream *)

Import a file to construct the model.
- void `Train` (const char *datasetFileName, char delimiter, int threads)

Train the model with the dataset file.
- std::ifstream * `OpenDatasetFile` (const char *filename)

Open dataset file and return the ifstream pointer.
- std::ofstream * `Save` (const char *filename)

Export model to file.
- void `Generate` (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

Call `Markov::Model::RandomWalk` n times, and collect output.
- void `Buff` (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)

Buff expression of some characters in the model.
- char * `RandomWalk` (`Markov::Random::RandomEngine` *randomEngine, int minSetting, int maxSetting, char *buffer)

Do a random walk on this model.
- void `AdjustEdge` (const char *payload, long int occurrence)

Adjust the model with a single string.
- bool `Export` (std::ofstream *)

Export a file of the model.
- bool `Export` (const char *filename)

- Open a file to export with filename, and call bool Model::Export with std::ofstream.
- Node< char > * **StarterNode** ()

Return starter Node.
- std::vector< Edge< char > * > * **Edges** ()

Return a vector of all the edges in the model.
- std::map< char, Node< char > * > * **Nodes** ()

Return starter Node.
- void **OptimizeEdgeOrder** ()

Sort edges of all nodes in the model ordered by edge weights.

Static Public Member Functions

- def **check_import_path** (str filename)

check import path for validity
- def **check_corpus_path** (str filename)

check import path for validity
- def **check_export_path** (str filename)

check import path for validity
- def **check_import_path** (str filename)

check import path for validity
- def **check_corpus_path** (str filename)

check import path for validity
- def **check_export_path** (str filename)

check import path for validity
- static __host__ void **ListCudaDevices** ()

List CUDA devices in the system.

Public Attributes

- **model**
- **bInfinite**
- **fileIO**
- **parser**
- **print_help**
- **args**
- **parser**
- **print_help**
- **args**

Protected Member Functions

- int **FastRandomWalk** (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced Markov::Model.
- void **FastRandomWalkPartition** (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

A single partition of FastRandomWalk event.
- void **FastRandomWalkThread** (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)

A single thread of a single partition of FastRandomWalk.
- bool **DeallocateMatrix** ()

Deallocate matrix and make it ready for re-construction.

- int [FastRandomWalk](#) (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced Markov::Model.
- [__host__ char * AllocVRAMOutputBuffer](#) (long int n, long int singleGenMaxLen, long int CUDAKernelGrid←Size, long int sizePerGrid)

Allocate the output buffer for kernel operation.
- [__host__ void LaunchAsyncKernel](#) (int kernelID, int minLen, int maxLen)
- [__host__ void prepKernelMemoryChannel](#) (int numberOfWorkStreams)
- [__host__ void GatherAsyncKernelOutput](#) (int kernelID, bool bFileIO, std::ofstream &wordlist)
- void [FastRandomWalkPartition](#) (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

A single partition of FastRandomWalk event.
- void [FastRandomWalkThread](#) (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)

A single thread of a single partition of FastRandomWalk.
- bool [DeallocateMatrix](#) ()

Deallocate matrix and make it ready for re-construction.

Static Protected Member Functions

- static [__host__ int CudaCheckNotifyErr](#) (cudaError_t _status, const char *msg, bool bExit=true)

Check results of the last operation on GPU.
- template<typename T >
 [static __host__ cudaError_t CudaMalloc2DToFlat](#) (T **dst, int row, int col)

Malloc a 2D array in device space.
- template<typename T >
 [static __host__ cudaError_t CudaMemcpy2DToFlat](#) (T *dst, T **src, int row, int col)

Memcpy a 2D array in device space after flattening.
- template<typename T >
 [static __host__ cudaError_t CudaMigrate2DFlat](#) (T **dst, T **src, int row, int col)

Both malloc and memcpy a 2D array into device VRAM.

Protected Attributes

- char ** [edgeMatrix](#)

2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** [valueMatrix](#)

2-d Integer array for the value Matrix (For the weights of Edges)
- int [matrixSize](#)

to hold Matrix size
- char * [matrixIndex](#)

to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * [totalEdgeWeights](#)

Array of the Total Edge Weights.
- bool [ready](#)

True when matrix is constructed. False if not.
- char ** [edgeMatrix](#)

2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** [valueMatrix](#)

2-d Integer array for the value Matrix (For the weights of Edges)
- int [matrixSize](#)

to hold Matrix size
- char * [matrixIndex](#)

- long int * **totalEdgeWeights**
Array of the Total Edge Weights.
- bool **ready**
True when matrix is constructed. False if not.

Private Member Functions

- def **_generate** (self, str wordlist)
wrapper for generate function.
- void **TrainThread** (Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char delimiter)
A single thread invoked by the Train function.
- void **GenerateThread** (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)
A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * **datasetFile**
- std::ofstream * **modelSavefile**
Dataset file input of our system
- std::ofstream * **outputFile**
File to save model of our system
- std::map< char, Node< char > * > **nodes**
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > * **starterNode**
Starter Node of this model.
- std::vector< Edge< char > * > **edges**
A list of all edges in this model.
- char * **device_edgeMatrix**
*VRAM Address pointer of edge matrix (from *modelMatrix.h*)*
- long int * **device_valueMatrix**
*VRAM Address pointer of value matrix (from *modelMatrix.h*)*
- char * **device_matrixIndex**
*VRAM Address pointer of matrixIndex (from *modelMatrix.h*)*
- long int * **device_totalEdgeWeights**
*VRAM Address pointer of total edge weights (from *modelMatrix.h*)*
- char ** **device_outputBuffer**
RandomWalk results in device.
- char ** **outputBuffer**
RandomWalk results in host.
- char * **flatEdgeMatrix**
Adding Edge matrix end-to-end and resize to 1-D array for better performance on traversing.
- long int * **flatValueMatrix**
Adding Value matrix end-to-end and resize to 1-D array for better performance on traversing.
- int **cudaBlocks**
- int **cudaThreads**
- int **iterationsPerKernelThread**
- long int **totalOutputPerSync**
- long int **totalOutputPerKernel**
- int **numberOfPartitions**

- int `cudaGridSize`
- int `cudaMemPerGrid`
- long int `cudaPerKernelAllocationSize`
- int `alternatingKernels`
- unsigned long ** `device_seeds`
- `cudaStream_t * cudastreams`

9.11.1 Detailed Description

Python CLI wrapper for CudaModelMatrix.

Definition at line 55 of file `cudammx.py`.

9.11.2 Constructor & Destructor Documentation

9.11.2.1 `__init__()`

```
def Python.CudaMarkopy.CudaModelMatrixCLI.__init__ (
    self )
```

Reimplemented in `Python.CudaMarkopy.CudaMarkopyCLI`.

Definition at line 63 of file `cudammx.py`.

```
00063     def __init__(self):
00064         super().__init__()
00065         self.model = cudamarkopy.CUDAModelMatrix()
00066
```

9.11.3 Member Function Documentation

9.11.3.1 `_generate()`

```
def Python.CudaMarkopy.CudaModelMatrixCLI._generate (
    self,
    str wordlist ) [private]
```

wrapper for generate function.

This can be overloaded by other models

Parameters

<code>wordlist</code>	filename to generate to
-----------------------	-------------------------

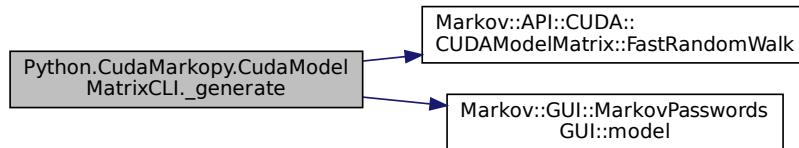
Reimplemented from `Python.Markopy.ModelMatrixCLI`.

Definition at line 75 of file `cudammx.py`.

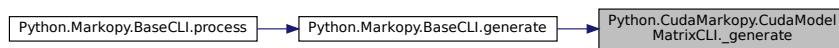
```
00075     def _generate(self, wordlist : str):
00076         self.model.FastRandomWalk(int(self.args.count), wordlist, int(self.args.min),
00077             int(self.args.max), self.fileIO, self.bInfinite)
```

References `Python.CudaMarkopy.CudaMarkopyCLI.args`, `Python.Markopy.BaseCLI.args`, `Python.Markopy.MarkopyCLI.args`, `Python.CudaMarkopy.CudaModelMatrixCLI.bInfinite`, `Markov::API::CUDA::CUDAModelMatrix.FastRandomWalk()`, `Python.Markopy.ModelMatrixCLI.fileIO`, `Python.CudaMarkopy.CudaModelMatrixCLI.model`, `Python.Markopy.BaseCLI.model`, `Python.Markopy.ModelMatrixCLI.model`, `Python.Markopy.MarkovPasswordsCLI.model`, and `Markov::GUI::MarkovPasswordsGUI.model`. Referenced by `Python.Markopy.BaseCLI.generate()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.2 add_arguments()

```
def Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments (
    self )
```

Reimplemented from [Python.Markopy.AbstractGenerationModelCLI](#).

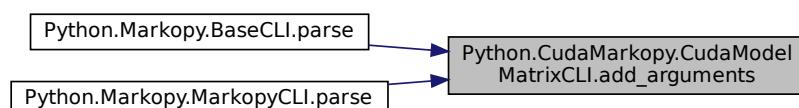
Definition at line [67](#) of file [cudammx.py](#).

```
00067     def add_arguments(self):
00068         super().add_arguments()
00069         self.parser.add_argument("-if", "--infinite", action="store_true", help="Infinite generation
mode")
00070 
```

References [Python.Markopy.BaseCLI.parser](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.11.3.3 AdjustEdge() [1/2]

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.11.3.4 AdjustEdge() [2/2]

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
```

```
00355 }
```

9.11.3.5 AllocVRAMOutputBuffer()

```
__host__ char* Markov::API::CUDA::CUDAModelMatrix::AllocVRAMOutputBuffer (
    long int n,
    long int singleGenMaxLen,
    long int CUDAKernelGridSize,
    long int sizePerGrid ) [protected], [inherited]
```

Allocate the output buffer for kernel operation.

TODO

Parameters

<i>n</i>	- Number of passwords to generate.
<i>singleGenMaxLen</i>	- maximum string length for a single generation
<i>CUDAKernelGridSize</i>	- Total number of grid members in CUDA kernel
<i>sizePerGrid</i>	- Size to allocate per grid member

Returns

pointer to the allocation on VRAM

9.11.3.6 Buff() [1/2]

```
void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]
```

Buff expression of some characters in the model.

Parameters

<i>str</i>	A string containing all the characters to be buffed
<i>multiplier</i>	A constant value to buff the nodes with.
<i>bDontAdjustSelfEdges</i>	Do not adjust weights if target node is same as source node
<i>bDontAdjustExtendedLoops</i>	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

```
00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges) {
00162             if(buffstr.find(targetrepr)!= std::string::npos) {
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops) {
00165                     if(buffstr.find(repr)!= std::string::npos) {
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight ();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173         }
00174     }
00175 }
```

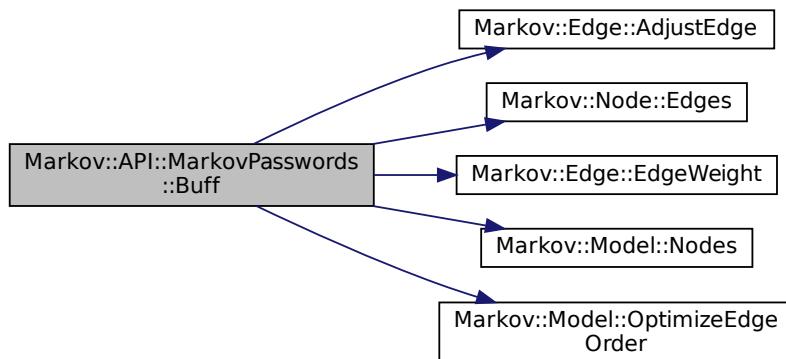
```

00173         }
00174     i++;
00175 }
00176 }
00177 this->OptimizeEdgeOrder();
00178 }
00179 }
```

References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.7 Buff() [2/2]

```

void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]
```

Buff expression of some characters in the model.

Parameters

<i>str</i>	A string containing all the characters to be buffed
<i>multiplier</i>	A constant value to buff the nodes with.
<i>bDontAdjustSelfEdges</i>	Do not adjust weights if target node is same as source node
<i>bDontAdjustExtendedLoops</i>	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

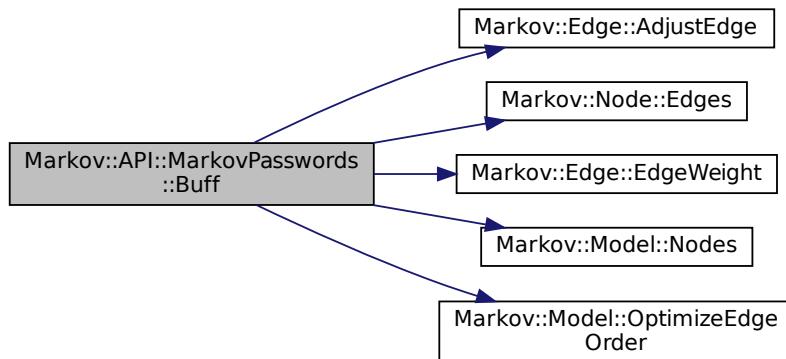
```

00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges) {
00162             if(buffstr.find(targetrepr)!= std::string::npos) {
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops) {
00165                     if(buffstr.find(repr)!= std::string::npos) {
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173         }
00174         i++;
00175     }
00176 }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

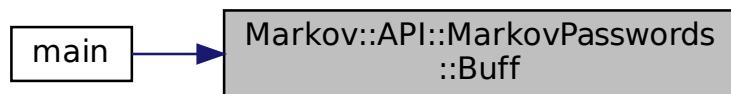
References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.8 check_corpus_path() [1/2]

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

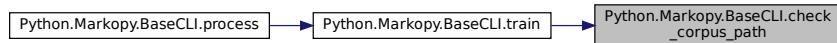
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.11.3.9 check_corpus_path() [2/2]

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.11.3.10 check_export_path() [1/2]

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.11.3.11 check_export_path() [2/2]

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.11.3.12 check_import_path() [1/2]

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

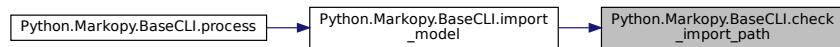
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173             """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.11.3.13 check_import_path() [2/2]

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

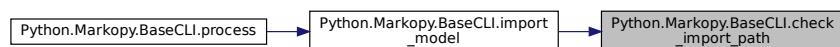
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173             """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.11.3.14 ConstructMatrix() [1/2]

```
bool Markov::API::ModelMatrix::ConstructMatrix () [inherited]
```

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content.

this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

True if constructed. False if already constructed.

Definition at line 31 of file [modelMatrix.cpp](#).

```
00031     {  
00032         if(this->ready) return false;  
00033         this->matrixSize = this->StarterNode()->edgesV.size() + 2;  
00034  
00035         this->matrixIndex = new char[this->matrixSize];  
00036         this->totalEdgeWeights = new long int[this->matrixSize];  
00037  
00038         this->edgeMatrix = new char*[this->matrixSize];  
00039         for(int i=0;i<this->matrixSize;i++){  
00040             this->edgeMatrix[i] = new char[this->matrixSize];  
00041         }  
00042         this->valueMatrix = new long int*[this->matrixSize];  
00043         for(int i=0;i<this->matrixSize;i++){  
00044             this->valueMatrix[i] = new long int[this->matrixSize];  
00045         }  
00046         std::map< char, Node< char > * > *nodes;  
00047         nodes = this->Nodes();  
00048         int i=0;  
00049         for (auto const& [repr, node] : *nodes){  
00050             if(repr!=0) this->matrixIndex[i] = repr;  
00051             else this->matrixIndex[i] = 199;  
00052             this->totalEdgeWeights[i] = node->TotalEdgeWeights();  
00053             for(int j=0;j<this->matrixSize;j++){  
00054                 char val = node->NodeValue();  
00055                 if(val < 0){  
00056                     for(int k=0;k<this->matrixSize;k++){  
00057                         this->valueMatrix[i][k] = 0;  
00058                         this->edgeMatrix[i][k] = 255;  
00059                     }  
00060                     break;  
00061                 }  
00062                 else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){  
00063                     this->valueMatrix[i][j] = 0;  
00064                     this->edgeMatrix[i][j] = 255;  
00065                 }else if(j==(this->matrixSize-1)) {  
00066                     this->valueMatrix[i][j] = 0;  
00067                     this->edgeMatrix[i][j] = 255;  
00068                 }else{  
00069                     this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();  
00070                     this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();  
00071                 }  
00072             }  
00073         }  
00074         i++;  
00075     }  
00076     this->ready = true;  
00077     return true;  
00078 //this->DumpJSON();  
00079 }
```

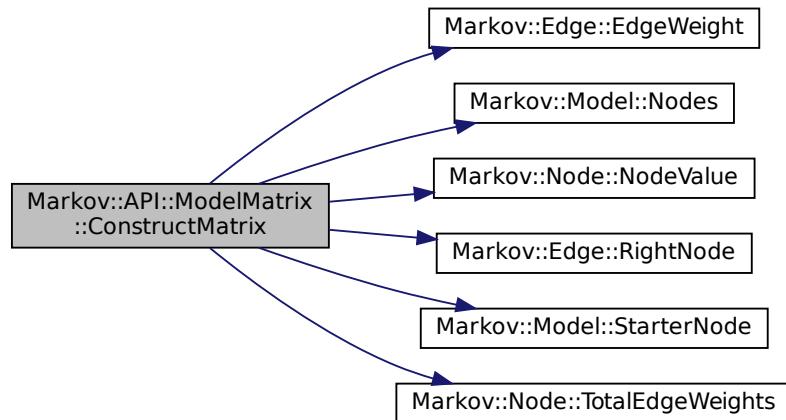
References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::API::ModelMatrix::Markov::API::ModelMatrix::matrixSize](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), [Markov::Node< storageType >::NodeValue\(\)](#)

[Markov::API::ModelMatrix::ready](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::Starter\(\)](#)

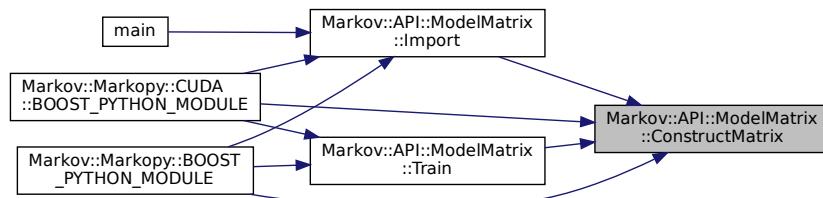
[Markov::API::ModelMatrix::totalEdgeWeights](#), [Markov::Node< storageType >::TotalEdgeWeights\(\)](#), and [Markov::API::ModelMatrix::valueMatrix](#)

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.15 `ConstructMatrix()` [2/2]

```
bool Markov::API::ModelMatrix::ConstructMatrix () [inherited]
Construct the related Matrix data for the model.
```

This operation can be used after importing/training to allocate and populate the matrix content. This will initialize: `char** edgeMatrix` -> a 2D array of mapping left and right connections of each edge. `long int **valueMatrix` -> a 2D array representing the edge weights. `int matrixSize` -> Size of the matrix, aka total number of nodes. `char* matrixIndex` -> order of nodes in the model `long int *totalEdgeWeights` -> total edge weights of each Node.

Returns

True if constructed. False if already constructed.

Definition at line 31 of file [modelMatrix.cpp](#).

```

00031
00032     if(this->ready)  return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
  
```

```

00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes) {
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072         }
00073     }
00074     i++;
00075 }
00076 this->ready = true;
00077 return true;
00078 //this->DumpJSON();
00079 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::API::ModelMatrix::ModelMatrix](#)

[Markov::API::ModelMatrix::matrixSize](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), [Markov::Node< storageType >::NodeValue\(\)](#)

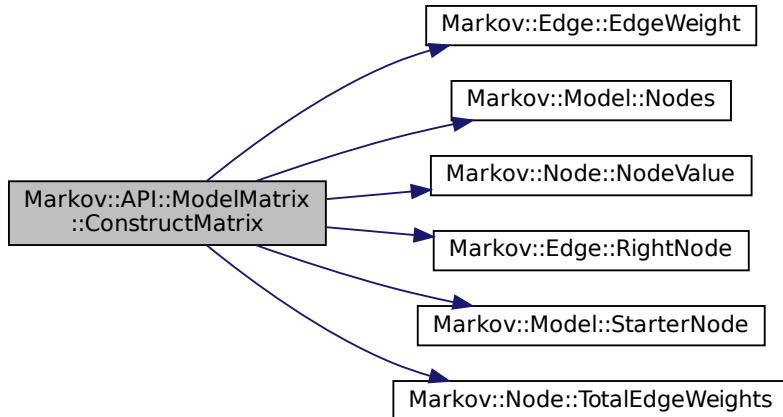
[Markov::API::ModelMatrix::ready](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::StarterNode](#)

[Markov::API::ModelMatrix::totalEdgeWeights](#), [Markov::Node< storageType >::TotalEdgeWeights\(\)](#), and [Markov::API::ModelMatrix::value](#)

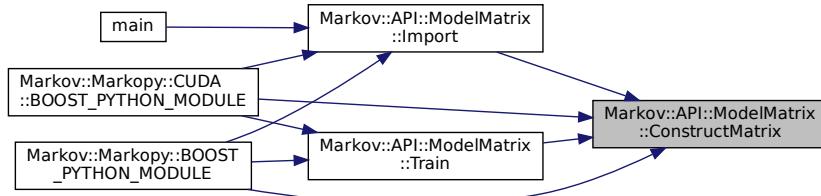
Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#),

[Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.16 CudaCheckNotifyErr()

```
__host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr (
    cudaError_t _status,
    const char * msg,
    bool bExit = true ) [static], [protected], [inherited]
```

Check results of the last operation on GPU.

Check the status returned from cudaMalloc/cudaMemcpy to find failures.

If a failure occurs, its assumed beyond redemption, and exited.

Parameters

<code>_status</code>	Cuda error status to check
<code>msg</code>	Message to print in case of a failure

Returns

0 if successful, 1 if failure. **Example output:**

```
char *da, a = "test";
cudaStatus = cudaMalloc((char **) &da, 5 * sizeof(char *));
CudaCheckNotifyErr(cudaStatus, "Failed to allocate VRAM for *da.\n");
```

Definition at line 32 of file [cudaDeviceController.cu](#).

```
00032
{
00033     if (_status != cudaSuccess) {
00034         std::cerr << "\033[1;31m" << msg << " -> " << cudaGetErrorString(_status) << "(" << _status <<
00035         ")" << "\033[0m" << "\n";
00036     if(bExit) {
00037         cudaDeviceReset();
00038         exit(1);
00039     }
00040 }
00041     return 0;
00042 }
```

9.11.3.17 CudaMalloc2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat (
    T ** dst,
    int row,
    int col ) [inline], [static], [protected], [inherited]
```

Malloc a 2D array in device space.

This function will allocate enough space on VRAM for flattened 2D array.

Parameters

<i>dst</i>	destination pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

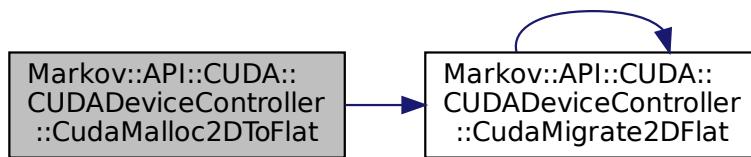
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
}
```

Definition at line 75 of file [cudaDeviceController.h](#).

```
00075
00076     cudaError_t cudastatus = cudaMalloc((T **)dst, row*col*sizeof(T));
00077     CudaCheckNotifyErr(cudastatus, "cudaMalloc Failed.", false);
00078     return cudastatus;
00079 }
```

References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:



9.11.3.18 CudaMemcpy2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat (
    T * dst,
    T ** src,
    int row,
    int col ) [inline], [static], [protected], [inherited]
```

Memcpy a 2D array in device space after flattening.

Resulting buffer will not be true 2D array.

Parameters

<i>dst</i>	destination pointer
<i>rc</i>	source pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

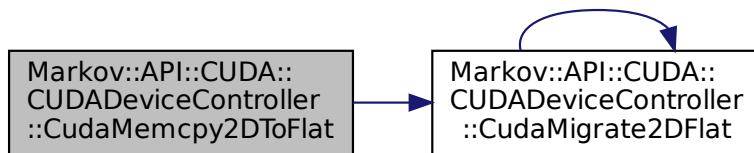
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
CudaCheckNotifyErr(cudastatus, " CudaMemcpy2DToFlat Failed.", false);
```

Definition at line 103 of file [cudaDeviceController.h](#).

```
00103
00104     T* tempbuf = new T[row*col];
00105     for(int i=0;i<row;i++){
00106         memcpy(&(tempbuf[row*i]), src[i], col);
00107     }
00108     return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109
00110 }
```

References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:

**9.11.3.19 CudaMigrate2DFlat()**

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat (
    T ** dst,
    T ** src,
    int row,
    int col ) [inline], [static], [protected], [inherited]
```

Both malloc and memcpy a 2D array into device VRAM.

Resulting buffer will not be true 2D array.

Parameters

<i>dst</i>	destination pointer
<i>rc</i>	source pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMigrate2DFlat<long int>(
    &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
CudaCheckNotifyErr(cudastatus, " Cuda failed to initialize value matrix row.");
```

Definition at line 132 of file [cudaDeviceController.h](#).

```
00132
00133     cudaError_t cudastatus;
00134     cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135     if(cudastatus!=cudaSuccess) {
00136         CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
00137         return cudastatus;
00138     }
00139     cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140     CudaCheckNotifyErr(cudastatus, " CudaMemcpy2DToFlat Failed.", false);
00141     return cudastatus;
00142 }
```

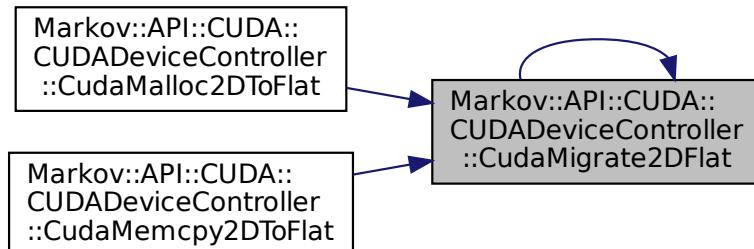
References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Referenced by [Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat\(\)](#), [Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat\(\)](#) and [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.20 DeallocateMatrix() [1/2]

```
bool Markov::API::ModelMatrix::DeallocateMatrix ( ) [protected], [inherited]
Deallocate matrix and make it ready for re-construction.
```

Returns

True if deallocated. False if matrix was not initialized

Definition at line 81 of file [modelMatrix.cpp](#).

```
00081
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
```

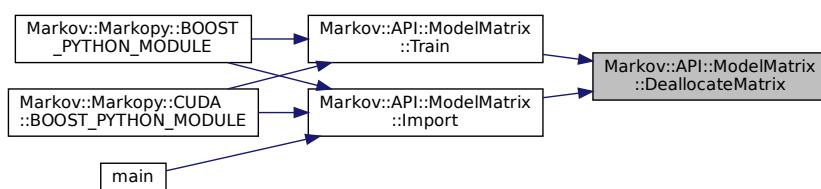
```

00085     for(int i=0;i<this->matrixSize;i++) {
00086         delete[] this->edgeMatrix[i];
00087     }
00088     delete[] this->edgeMatrix;
00089
00090     for(int i=0;i<this->matrixSize;i++) {
00091         delete[] this->valueMatrix[i];
00092     }
00093     delete[] this->valueMatrix;
00094
00095     this->matrixSize = -1;
00096     this->ready = false;
00097     return true;
00098 }
00099 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::API::ModelMatrix::ready](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the caller graph for this function:



9.11.3.21 DeallocateMatrix() [2/2]

```
bool Markov::API::ModelMatrix::DeallocateMatrix ( ) [protected], [inherited]
```

Deallocate matrix and make it ready for re-construction.

Returns

True if deallocated. False if matrix was not initialized

Definition at line 81 of file [modelMatrix.cpp](#).

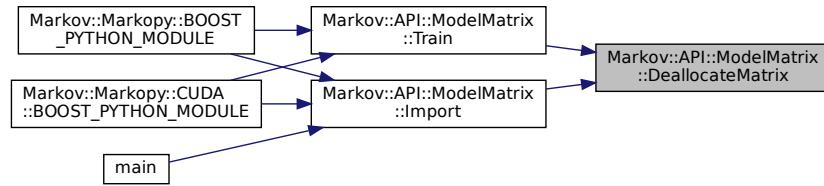
```

00081     if(!this->ready) return false;
00082     delete[] this->matrixIndex;
00083     delete[] this->totalEdgeWeights;
00084
00085     for(int i=0;i<this->matrixSize;i++) {
00086         delete[] this->edgeMatrix[i];
00087     }
00088     delete[] this->edgeMatrix;
00089
00090     for(int i=0;i<this->matrixSize;i++) {
00091         delete[] this->valueMatrix[i];
00092     }
00093     delete[] this->valueMatrix;
00094
00095     this->matrixSize = -1;
00096     this->ready = false;
00097     return true;
00098 }
00099 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::API::ModelMatrix::ready](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the caller graph for this function:



9.11.3.22 DumpJSON() [1/2]

void Markov::API::ModelMatrix::DumpJSON () [inherited]

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

Definition at line 101 of file [modelMatrix.cpp](#).

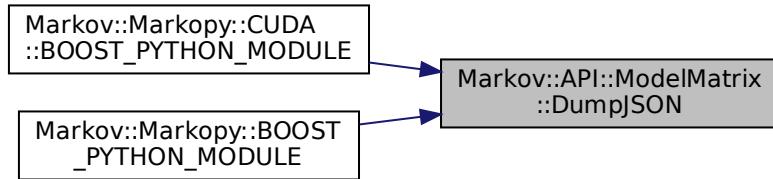
```

00101
00102
00103     std::cout << "{\n      \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++) {
00105         if(this->matrixIndex[i]=='"') std::cout << "\\\\"\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout << "\\\\\\"";
00107         else if(this->matrixIndex[i]==0) std::cout << "\\\\x00";
00108         else if(i==0) std::cout << "\\\\xff";
00109         else if(this->matrixIndex[i]=='\\n') std::cout << "\\n";
00110         else std::cout << this->matrixIndex[i];
00111     }
00112     std::cout <
00113     " \\n"
00114     "   \"edgemap\": {\n\\n
00115
00116     for(int i=0;i<this->matrixSize;i++) {
00117         if(this->matrixIndex[i]=='"') std::cout << "      \"\\\\\"\\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\\\\\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\\\\\x00\\": [";
00120         else if(this->matrixIndex[i]<0) std::cout << "      \"\\\\\\\\xf\\": [";
00121         else std::cout << "      \" \" << this->matrixIndex[i] << "\": [";
00122         for(int j=0;j<this->matrixSize;j++) {
00123             if(this->edgeMatrix[i][j]=='"') std::cout << "        \"\\\\\"\\": ";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout << "        \"\\\\\\\\\\": ";
00125             else if(this->edgeMatrix[i][j]==0) std::cout << "        \"\\\\\\\\x00\\": ";
00126             else if(this->edgeMatrix[i][j]<0) std::cout << "        \"\\\\\\\\xf\\": ";
00127             else if(this->matrixIndex[i]=='\\n') std::cout << "        \"\\\\n\\": ";
00128             else std::cout << "        \" \" << this->edgeMatrix[i][j] << "\": ";
00129             if(j!=this->matrixSize-1) std::cout << ", ";
00130         }
00131     std::cout << "],\\n";
00132   }
00133   std::cout << "},\\n";
00134
00135   std::cout << "  \" weightmap\": {\n\\n
00136     for(int i=0;i<this->matrixSize;i++) {
00137         if(this->matrixIndex[i]=='"') std::cout << "    \"\\\\\"\\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout << "    \"\\\\\\\\\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout << "    \"\\\\\\\\x00\\": [";
00140         else if(this->matrixIndex[i]<0) std::cout << "    \"\\\\\\\\xf\\": [";
00141         else std::cout << "    \" \" << this->matrixIndex[i] << "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++) {
00144             std::cout << this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout << ", ";
00146         }
00147     std::cout << "],\\n";
00148   }
00149   std::cout << "  }\\n}\\n";
00150 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the caller graph for this function:



9.11.3.23 DumpJSON() [2/2]

void Markov::API::ModelMatrix::DumpJSON () [inherited]

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

Definition at line 101 of file [modelMatrix.cpp](#).

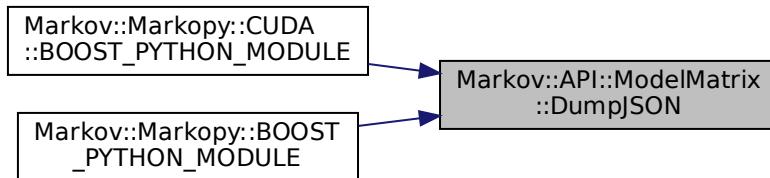
```

00101
00102
00103     std::cout << "{\n    \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++) {
00105         if(this->matrixIndex[i]=='"') std::cout << "\\\\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout << "\\\\\\"";
00107         else if(this->matrixIndex[i]==0) std::cout << "\\\\x00";
00108         else if(i==0) std::cout << "\\\\xff";
00109         else if(this->matrixIndex[i]=='\\n') std::cout << "\\n";
00110         else std::cout << this->matrixIndex[i];
00111     }
00112     std::cout <
00113     "\\",\\n"
00114     "    \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++) {
00117         if(this->matrixIndex[i]=='"') std::cout << "        \"\\\\\"\\": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout << "        "\\\\\\\\"\\": [";
00119         else if(this->matrixIndex[i]==0) std::cout << "        "\\\\\\\\"x00\\": [";
00120         else if(this->matrixIndex[i]<0) std::cout << "        "\\\\\\\\"xf\\": [";
00121         else std::cout << "        \"\" << this->matrixIndex[i] << "\": [";
00122         for(int j=0;j<this->matrixSize;j++) {
00123             if(this->edgeMatrix[i][j]=='"') std::cout << "            \"\\\\\"\\": ";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout << "            "\\\\\\\\"\\": ";
00125             else if(this->edgeMatrix[i][j]==0) std::cout << "            "\\\\\\\\"x00\\": ";
00126             else if(this->edgeMatrix[i][j]<0) std::cout << "            "\\\\\\\\"xf\\": ";
00127             else if(this->matrixIndex[i]=='\\n') std::cout << "            \"\\\\\\n\\": ";
00128             else std::cout << "            \"\" << this->edgeMatrix[i][j] << "\": ";
00129             if(j!=this->matrixSize-1) std::cout << ", ";
00130         }
00131         std::cout << "],\\n";
00132     }
00133     std::cout << "},\\n";
00134
00135     std::cout << "    \"weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++) {
00137         if(this->matrixIndex[i]=='"') std::cout << "        \"\\\\\"\\": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout << "        "\\\\\\\\"\\": [";
00139         else if(this->matrixIndex[i]==0) std::cout << "        "\\\\\\\\"x00\\": [";
00140         else if(this->matrixIndex[i]<0) std::cout << "        "\\\\\\\\"xf\\": [";
00141         else std::cout << "        \"\" << this->matrixIndex[i] << "\": [";
00142
00143         for(int j=0;j<this->matrixSize;j++) {
00144             std::cout << this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout << ", ";
00146         }
00147         std::cout << "],\\n";
00148     }
00149     std::cout << "  }\\n}\\n";
  
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the caller graph for this function:



9.11.3.24 Edges() [1/2]

`std::vector<Edge<char *>>* Markov::Model< char >::Edges () [inline], [inherited]`
Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file `model.h`.

```
00176 { return &edges; }
```

9.11.3.25 Edges() [2/2]

`std::vector<Edge<char *>>* Markov::Model< char >::Edges () [inline], [inherited]`
Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file `model.h`.

```
00176 { return &edges; }
```

9.11.3.26 Export() [1/4]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
Open a file to export with filename, and call bool Model::Export with std::ofstream.
```

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file `model.h`.

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.11.3.27 Export() [2/4]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file model.h.

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.11.3.28 export() [1/2]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

Definition at line 138 of file base.py.

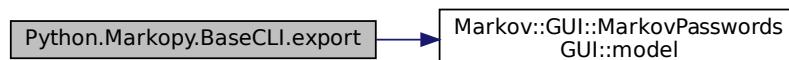
```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.mo](#)

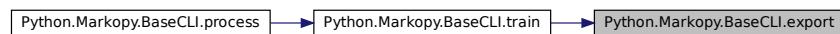
[Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.29 `export()` [2/2]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<code>filename</code>	filename to export to
-----------------------	-----------------------

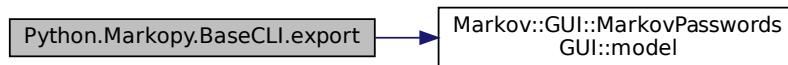
Definition at line 138 of file [base.py](#).

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.30 `Export()` [3/4]

```
bool Markov::Model< char >::Export (
    std::ofstream * f ) [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file [model.h](#).

```
00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
```

```

00292     //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <
00293     *f << e->LeftNode()->NodeValue() << "\n";
00294   }
00295 
00296   return true;
00297 }
```

9.11.3.31 Export() [4/4]

```
bool Markov::Model< char >::Export (
    std::ofstream * f )  [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file model.h.

```

00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291       e = this->edges[i];
00292       //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <
00293       *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <
00294       "\n";
00295     }
00296   return true;
00297 }
```

9.11.3.32 FastRandomWalk() [1/6]

```
def Python.Markopy.ModelMatrix.FastRandomWalk (
    int count,
    str wordlist,
    int minlen,
    int maxlen )  [inherited]
```

Definition at line 48 of file mm.py.

```
00048 def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049     pass
```

Referenced by [Python.Markopy.ModelMatrixCLI._generate\(\)](#).

Here is the caller graph for this function:



9.11.3.33 FastRandomWalk() [2/6]

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen,
```

```

    int maxLen,
    bool bFileIO,
    bool bInfinite ) [inherited]

```

Random walk on the Matrix-reduced [Markov::Model](#).

TODO

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```

Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);

```

Definition at line 58 of file [cudaModelMatrix.cu](#).

```

00058
00059     cudaDeviceProp prop;
00060     int device=0;
00061     cudaGetDeviceProperties(&prop, device);
00062     cudaChooseDevice(&device, &prop);
00063     //std::cout << "Flattening matrix." << std::endl;
00064     this->FlattenMatrix();
00065     //std::cout << "Migrating matrix." << std::endl;
00066     this->MigrateMatrix();
00067     //std::cout << "Migrated matrix." << std::endl;
00068     std::ofstream wordlist;
00069     if(bFileIO)
00070         wordlist.open(wordlistFileName);
00071
00072
00073     cudaBlocks = 1024;
00074     cudaThreads = 256;
00075     iterationsPerKernelThread = 100;
00076     alternatingKernels = 2;
00077     totalOutputPerKernel = (long int)cudaBlocks*(long int)cudaThreads*iterationsPerKernelThread;
00078     totalOutputPerSync= totalOutputPerKernel*alternatingKernels;
00079     numberofPartitions = n/totalOutputPerSync;
00080     cudaGridSize = cudaBlocks*cudatThreads;
00081     cudaMemPerGrid = (maxLen+2)*iterationsPerKernelThread;
00082     cudaPerKernelAllocationSize = cudaGridSize*cudatMemPerGrid;
00083     this->prepKernelMemoryChannel(alternatingKernels);
00084
00085     unsigned long int leftover = n - (totalOutputPerSync*numberofPartitions);
00086
00087     if(bInfinite && !numberofPartitions) numberofPartitions=5;
00088     std::cerr << cudaPerKernelAllocationSize << "\n";
00089
00090     if(n%totalOutputPerSync) std::cerr << "For optimization, request outputs multiples of "<<
00091     totalOutputPerSync << ".\n";
00092
00093     //start kernelID 1
00094     this->LaunchAsyncKernel(1, minLen, maxLen);
00095
00096     for(int i=1;i<numberofPartitions;i++) {
00097         if(bInfinite) i=0;
00098
00099         //wait kernelID1 to finish, and start kernelID 0
00100         cudaStreamSynchronize(this->cudastreams[1]);
00101         this->LaunchAsyncKernel(0, minLen, maxLen);
00102
00103         //start memcpy from kernel 1 (block until done)
00104         this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00105
00106         //wait kernelID 0 to finish, then start kernelID1
00107         cudaStreamSynchronize(this->cudastreams[0]);
00108         this->LaunchAsyncKernel(1, minLen, maxLen);
00109
00110         //start memcpy from kernel 0 (block until done)
00111         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00112     }
00113
00114     //wait kernelID1 to finish, and start kernelID 0

```

```

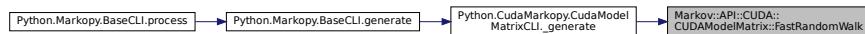
00115     cudaStreamSynchronize(this->cudastreams[1]);
00116     this->LaunchAsyncKernel(0, minLen, maxLen);
00117     this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00118     cudaStreamSynchronize(this->cudastreams[0]);
00119     this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00120
00121
00122     if(!leftover) return;
00123     alternatingKernels=1;
00124     std::cerr << "Remaining line count (" << leftover << ") is lower than partition. Adjusting CUDA
workload..\n";
00125     this->iterationsPerKernelThread = leftover/cudaGridSize;
00126     this->LaunchAsyncKernel(0, minLen, maxLen);
00127     cudaStreamSynchronize(this->cudastreams[0]);
00128     this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00129
00130     leftover -= this->iterationsPerKernelThread*cudaGridSize;
00131     if(!leftover) return;
00132
00133     std::cerr << "Remaining line count (" << leftover << ") is lower than minimum possible. Handing
over to CPU generation.\n";
00134     this->iterationsPerKernelThread = leftover/cudaGridSize;
00135
00136     leftover -= this->iterationsPerKernelThread;
00137
00138     if(!leftover) return;
00139     std::cerr << "Remaining " << leftover << " lines are absolutely not worth printing.\n";
00140     Markov::API::ModelMatrix::ConstructMatrix();
00141     Markov::API::ModelMatrix::FastRandomWalk(leftover, &wordlist, minLen, maxLen, 1, bFileIO);
00142
00143 }

```

References [Markov::API::CUDA::CUDAModelMatrix::alternatingKernels](#), [Markov::API::CUDA::CUDAModelMatrix::cudaBlocks](#), [Markov::API::CUDA::CUDAModelMatrix::cudaGridSize](#), [Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid](#), [Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize](#), [Markov::API::CUDA::CUDAModelMatrix::cudaThreads](#), [Markov::API::CUDA::CUDAModelMatrix::iterationsPerKernelThread](#), [Markov::API::CUDA::CUDAModelMatrix::numberOfPartitions](#), [Markov::API::CUDA::CUDAModelMatrix::totalOutputPerKernel](#), and [Markov::API::CUDA::CUDAModelMatrix::totalOutputPerSync](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI::_generate\(\)](#).

Here is the caller graph for this function:



9.11.3.34 FastRandomWalk() [3/6]

```

int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [inherited]

```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to

Parameters

<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

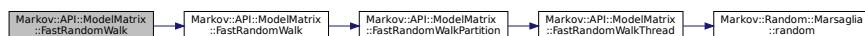
Definition at line 217 of file [modelMatrix.cpp](#).

```
00217
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

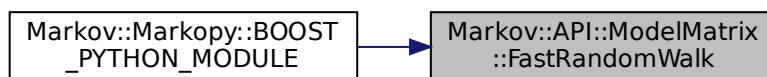
References [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.35 FastRandomWalk() [4/6]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

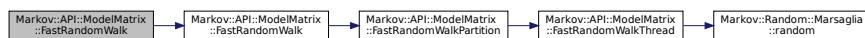
Definition at line 217 of file [modelMatrix.cpp](#).

```
00217
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

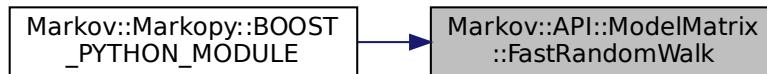
References [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.36 FastRandomWalk() [5/6]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [protected], [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

Definition at line 204 of file [modelMatrix.cpp](#).

```
00204     {
00205
00206
00207     std::mutex mlock;
00208     if(n<50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
00209     threads);
00210     else{
00211         int numberOfPartitions = n/50000000ull;
00212         for(int i=0;i<numberOfPartitions;i++)
00213             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
00214             threads);
00215     }
00214     return 0;
00215 }
```

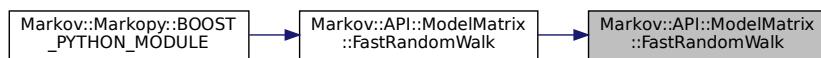
References [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.37 FastRandomWalk() [6/6]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [protected], [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 204 of file [modelMatrix.cpp](#).

```
00204
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
00209     threads);
00210     else{
00211         int numberOfPartitions = n/50000000ull;
00212         for(int i=0;i<numberOfPartitions;i++)
00213             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
00214             threads);
00215 }
```

References [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.38 FastRandomWalkPartition() [1/2]

```
void Markov::API::ModelMatrix::FastRandomWalkPartition (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    bool bFileIO,
    int threads ) [protected], [inherited]
```

A single partition of FastRandomWalk event.

Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation
- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 225 of file [modelMatrix.cpp](#).

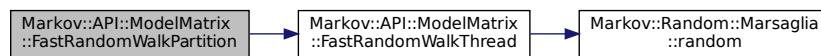
```

00225
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++) {
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00235                                         mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00236         id++;
00237     }
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00239                                         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00240     for(int i=0;i<threads;i++) {
00241         threadsV[i]->join();
00242     }
00243 }
```

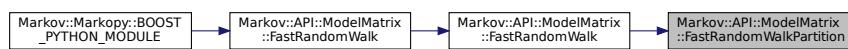
References [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.39 FastRandomWalkPartition() [2/2]

```
void Markov::API::ModelMatrix::FastRandomWalkPartition (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    bool bFileIO,
    int threads ) [protected], [inherited]
```

A single partition of FastRandomWalk event.

Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation
- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

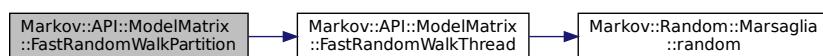
Definition at line 225 of file [modelMatrix.cpp](#).

```
00225
00226     int iterationsPerThread = n/threads;
00227     int iterationsPerThreadCarryOver = n%threads;
00228
00229     std::vector<std::thread*> threadsV;
00230
00231     int id = 0;
00232     for(int i=0;i<threads;i++) {
00233         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00234                                         mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00239                                         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00240
00241     for(int i=0;i<threads;i++) {
00242         threadsV[i]->join();
00243     }
```

References [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.40 FastRandomWalkThread() [1/2]

```
void Markov::API::ModelMatrix::FastRandomWalkThread (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    int id,
    bool bFileIO ) [protected], [inherited]
```

A single thread of a single partition of FastRandomWalk.

A FastRandomWalkPartition will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>id</i>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 153 of file [modelMatrix.cpp](#).

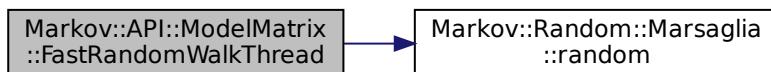
```
00153
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179             if (len >= maxLen) break;
00180             else if ((next < 0) && (len < minLen)) continue;
00181             else if (next < 0) break;
00182             cur = next;
00183         }
00184     }
00185 }
```

```

00184         res[bufferctr + len++] = cur;
00185     }
00186     res[bufferctr + len++] = '\n';
00187     bufferctr+=len;
00188 }
00189 if(bFileIO){
00190     mlock->lock();
00191     *wordlist << res;
00192     mlock->unlock();
00193 }else{
00194     mlock->lock();
00195     std::cout << res;
00196     mlock->unlock();
00197 }
00198 delete res;
00199
00200 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::Random::Marsaglia::random\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#). Referenced by [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.41 FastRandomWalkThread() [2/2]

```

void Markov::API::ModelMatrix::FastRandomWalkThread (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    int id,
    bool bFileIO ) [protected], [inherited]
```

A single thread of a single partition of FastRandomWalk.

A FastRandomWalkPartition will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate

Parameters

<i>id</i>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

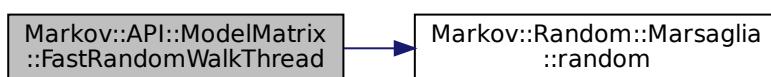
Definition at line 153 of file [modelMatrix.cpp](#).

```

00153
00154     if(n==0)  return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179             if (len >= maxLen)  break;
00180             else if ((next < 0) && (len < minLen))  continue;
00181             else if (next < 0)  break;
00182             cur = next;
00183             res[bufferctr + len++] = cur;
00184         }
00185     }
00186     res[bufferctr + len++] = '\n';
00187     bufferctr+=len;
00188
00189 }
00190 if(bFileIO){
00191     mlock->lock();
00192     *wordlist << res;
00193     mlock->unlock();
00194 }else{
00195     mlock->lock();
00196     std::cout << res;
00197     mlock->unlock();
00198 }
00199 delete res;
00200
00201 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::Random::Marsaglia::random\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#). Referenced by [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.42 FlattenMatrix()

`__host__ void Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix () [inherited]`
Flatten migrated matrix from 2d to 1d.

Definition at line 261 of file [cudaModelMatrix.cu](#).

```

00261     {
00262         this->flatEdgeMatrix = new char[this->matrixSize*this->matrixSize];
00263
00264         this->flatValueMatrix = new long int[this->matrixSize*this->matrixSize];
00265         for(int i=0;i<this->matrixSize;i++){
00266             memcpy(&this->flatEdgeMatrix[i*this->matrixSize], this->edgeMatrix[i], this->matrixSize );
00267             memcpy(&this->flatValueMatrix[i*this->matrixSize], this->valueMatrix[i],
00268                 this->matrixSize*sizeof(long int) );
00269         }
00269     }
  
```

References [Markov::API::CUDA::CUDAModelMatrix::flatEdgeMatrix](#), [Markov::API::CUDA::CUDAModelMatrix::flatValueMatrix](#), and [Markov::API::ModelMatrix::matrixSize](#).

9.11.3.43 GatherAsyncKernelOutput()

`__host__ void Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput (`
 `int kernelID,`
 `bool bFileIO,`
 `std::ofstream & wordlist) [protected], [inherited]`

Definition at line 180 of file [cudaModelMatrix.cu](#).

```

00180     {
00181         {
00182             cudaMemcpy(this->outputBuffer[kernelID],this->device_outputBuffer[kernelID],cudaPerKernelAllocationSize,
00183             cudaMemcpyDeviceToHost);
00182             //std::cerr << "Kernel" << kernelID << " output copied\n";
00183             if(bFileIO){
00184                 for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00185                     wordlist << &this->outputBuffer[kernelID][j];
00186                 }
00187             }else{
00188                 for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00189                     std::cout << &this->outputBuffer[kernelID][j];
00190                 }
00191             }
00192         }
  
```

References [Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid](#), [Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize](#), and [Markov::API::CUDA::CUDAModelMatrix::outputBuffer](#).

9.11.3.44 generate() [1/2]

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159
```

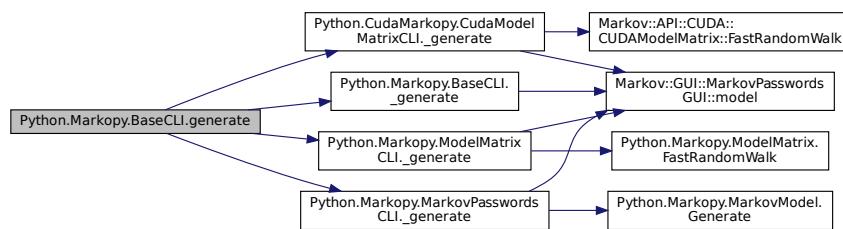
References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#),

[Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopy](#)

[Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.45 generate() [2/2]

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
```

```

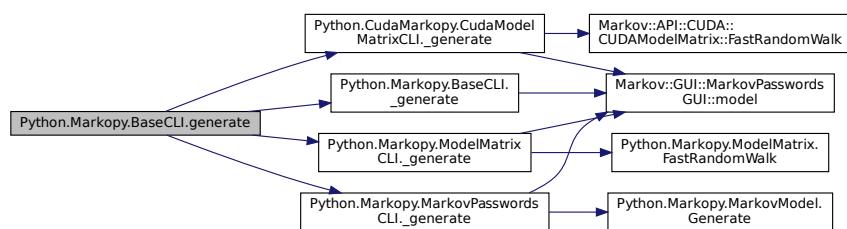
00149         @param wordlist wordlist filename
00150         @param bulk marks bulk operation with directories
00151     """
00152     if not (wordlist or self.args.count):
00153         logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154         Exiting.")
00155     return False
00156
00157     if(bulk and os.path.isfile(wordlist)):
00158         logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00159     self._generate(wordlist)

```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopy](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.46 Generate() [1/2]

```

void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]

```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. [FastRandomWalk](#) will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

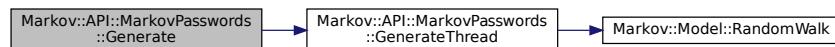
```

00118     {
00119     char* res;
00120     char print[100];
00121     std::ifstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++) {
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129             &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130     }
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137 }
00138 }
```

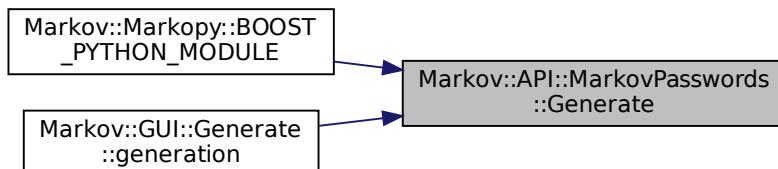
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.47 Generate() [2/2]

```

void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]

```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

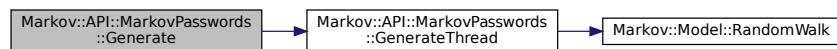
```

00118
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++) {
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129                                         &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130     }
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]~>join();
00133         delete threadsV[i];
00134     }
00135     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00136
00137
00138 }
```

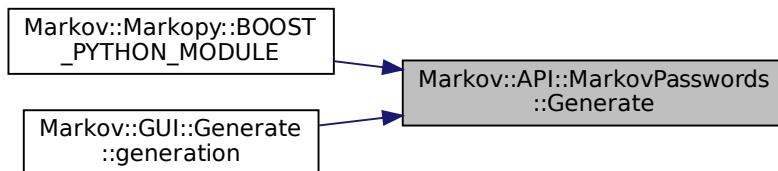
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.48 GenerateThread()

```

void Markov::API::MarkovPasswords::GenerateThread (
    std::mutex * outputLock,
    unsigned long int n,
    std::ofstream * wordlist,
```

```
    int minLen,
    int maxLen ) [private], [inherited]
```

A single thread invoked by the Generate function.

DEPRECATED: See `Markov::API::MatrixModel::FastRandomWalkThread` for more information. This has been replaced with a much more performance-optimized method. `FastRandomWalk` will reduce the runtime by %96.5 on average.

Parameters

<code>outputLock</code>	- shared mutex lock to lock during output operation. Prevents race condition on write.
<code>n</code>	number of lines to be generated by this thread
<code>wordlist</code>	wordlistfile
<code>minLen</code>	- Minimum password length to generate
<code>maxLen</code>	- Maximum password length to generate

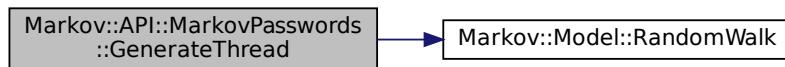
Definition at line 140 of file `markovPasswords.cpp`.

```
00140
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist << res << "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

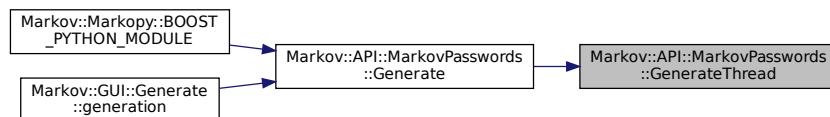
References `Markov::Model< NodeStorageType >::RandomWalk()`.

Referenced by `Markov::API::MarkovPasswords::Generate()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.49 help() [1/2]

```
def Python.Markopy.BaseCLI.help (
    self ) [inherited]
```

Reimplemented in `Python.Markopy.MarkopyCLI`, and `Python.CudaMarkopy.CudaMarkopyCLI`.

Definition at line 51 of file `base.py`.

```
00051     def help(self):
00052         """! @brief Handle help strings. Defaults to argparse's help"
```

```
00053     self.print_help()
00054
References Python.Markopy.BaseCLI.print\_help.
Referenced by Python.Markopy.MarkopyCLI.add\_arguments\(\).
Here is the caller graph for this function:
```



9.11.3.50 help() [2/2]

```
def Python.Markopy.BaseCLI.help (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 51 of file [base.py](#).

```
00051     def help(self):
00052         """! @brief Handle help strings. Defaults to argparse's help"""
00053         self.print_help()
00054
```

References [Python.Markopy.BaseCLI.print_help](#).

Referenced by [Python.Markopy.MarkopyCLI.add_arguments\(\)](#).

Here is the caller graph for this function:



9.11.3.51 Import() [1/4]

```
void Markov::API::ModelMatrix::Import (
    const char * filename ) [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.

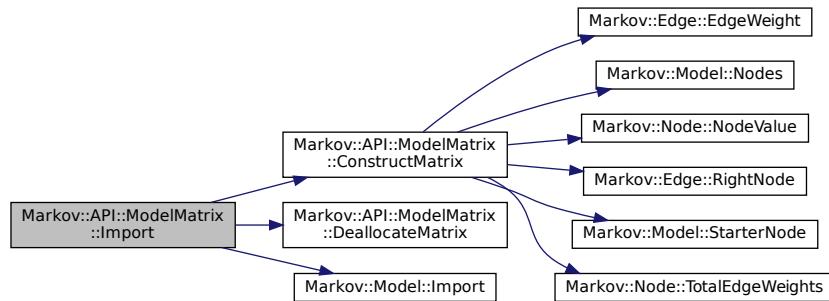
Definition at line 19 of file [modelMatrix.cpp](#).

```
00019
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

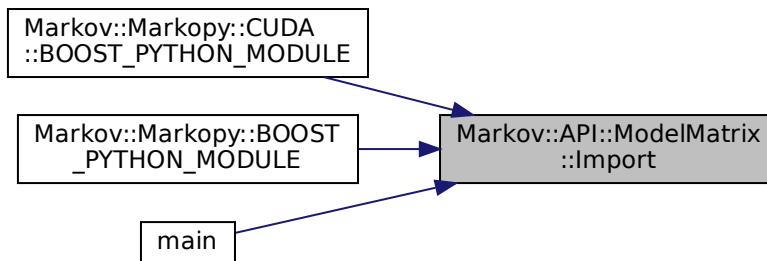
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.52 Import() [2/4]

```
void Markov::API::ModelMatrix::Import (
    const char * filename ) [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.

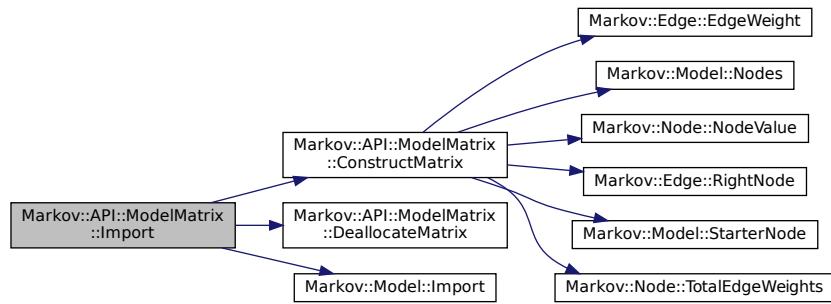
Definition at line 19 of file [modelMatrix.cpp](#).

```
00019
00020     this->DeallocateMatrix(); {
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

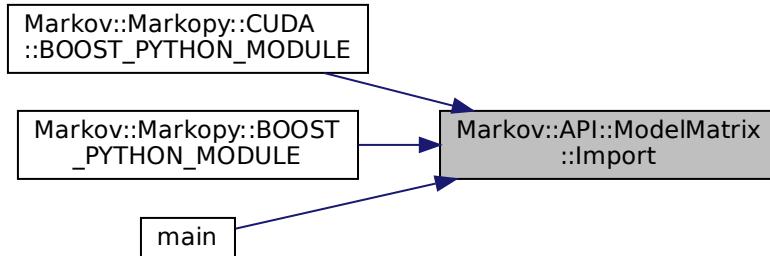
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.53 Import() [3/4]

```
bool Markov::Model< char >::Import (
    std::ifstream * f) [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file [model.h](#).

```
00216
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
```

```

00227     char* j;
00228     oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00229     //std::cout << oc << "\n";
00230     Markov::Node<NodeStorageType>* srcN;
00231     Markov::Node<NodeStorageType>* targetN;
00232     Markov::Edge<NodeStorageType>* e;
00233     if (this->nodes.find(src) == this->nodes.end()) {
00234         srcN = new Markov::Node<NodeStorageType>(src);
00235         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00236         //std::cout << "Creating new node at start.\n";
00237     }
00238     else {
00239         srcN = this->nodes.find(src)->second;
00240     }
00241     if (this->nodes.find(target) == this->nodes.end()) {
00242         targetN = new Markov::Node<NodeStorageType>(target);
00243         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00244         //std::cout << "Creating new node at end.\n";
00245     }
00246     else {
00247         targetN = this->nodes.find(target)->second;
00248     }
00249     e = srcN->Link(targetN);
00250     e->AdjustEdge(oc);
00251     this->edges.push_back(e);
00252
00253     //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00254     int(targetN->NodeValue()) << "\n";
00255
00256
00257 }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.11.3.54 Import() [4/4]

```
bool Markov::Model< char >::Import (
    std::ifstream * f )  [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file model.h.

```

00216
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
```

```

00240         }
00241
00242     if (this->nodes.find(target) == this->nodes.end()) {
00243         targetN = new Markov::Node<NodeStorageType>(target);
00244         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00245         //std::cout << "Creating new node at end.\n";
00246     }
00247     else {
00248         targetN = this->nodes.find(target)->second;
00249     }
00250     e = srcN->Link(targetN);
00251     e->AdjustEdge(oc);
00252     this->edges.push_back(e);
00253
00254     //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "-->" <<
00255     int(targetN->NodeValue()) << "\n";
00256
00257 }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.11.3.55 import_model() [1/2]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<code>filename</code>	filename to import
-----------------------	--------------------

Definition at line 77 of file [base.py](#).

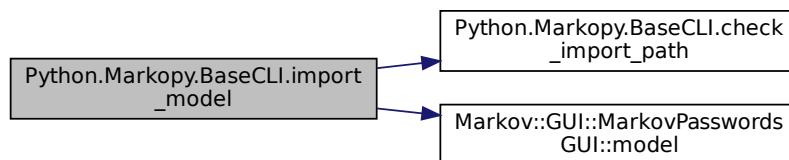
```

00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
00086             directory")
00087             return False
00088
00089         self.model.Import(filename)
00090         logging pprint("Model imported successfully.", 2)
00091
00092
00093
```

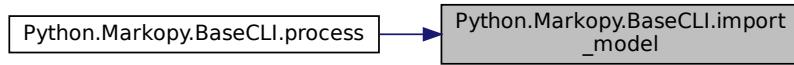
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.56 import_model() [2/2]

```
def Python.Markopy.BaseCLI.import_model ( self, str filename ) [inherited]
```

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

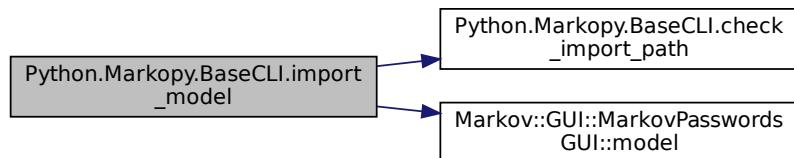
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

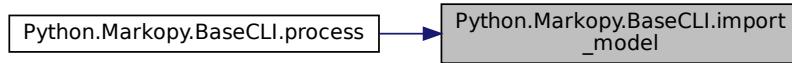
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#)

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.57 init_post_arguments()

```
def Python.CudaMarkopy.CudaModelMatrixCLI.init_post_arguments (
    self )
```

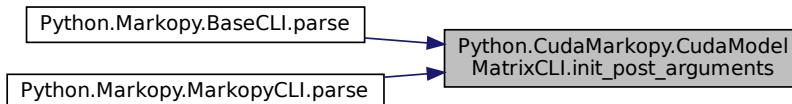
Reimplemented from [Python.Markopy.ModelMatrixCLI](#).

Definition at line 71 of file [cudamxx.py](#).

```
00071     def init_post_arguments(self):
00072         super().init_post_arguments()
00073         self.bInfinite = self.args.infinite
00074
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.11.3.58 LaunchAsyncKernel()

```
__host__ void Markov::API::CUDA::CUDAModelMatrix::LaunchAsyncKernel (
    int kernelID,
    int minLen,
    int maxLen ) [protected], [inherited]
```

Definition at line 171 of file [cudaModelMatrix.cu](#).

```
00171
00172
00173     //if(kernelID == 0); // cudaStreamSynchronize(this->cudastreams[2]);
00174     //else cudaStreamSynchronize(this->cudastreams[kernelID-1]);
00175     FastRandomWalkCUDAKernel<<cudaBlocks,cudaThreads,0,
00176     this->cudastreams[kernelID]>>(iterationsPerKernelThread, minLen, maxLen,
00177     this->device_outputBuffer[kernelID], this->device_matrixIndex,
00178     this->device_totalEdgeWeights, this->device_valueMatrix, this->device_edgeMatrix,
00179     this->matrixSize, cudaMemPerGrid, this->device_seeds[kernelID]);
00177     //std::cerr << "Started kernel" << kernelID << "\n";
00178 }
```

9.11.3.59 ListCudaDevices()

```
__host__ void Markov::API::CUDA::CUDADeviceController::ListCudaDevices ( ) [static], [inherited]
```

List CUDA devices in the system.

This function will print details of every CUDA capable device in the system.

Example output:

```
Device Number: 0
Device name: GeForce RTX 2070
```

```
Memory Clock Rate (KHz): 7001000
Memory Bus Width (bits): 256
Peak Memory Bandwidth (GB/s): 448.064
Max Linear Threads: 1024
```

Definition at line 16 of file [cudaDeviceController.cu](#).

```
00016                                     { //list cuda Capable devices on
00017     host.
00018     int nDevices;
00019     cudaGetDeviceCount(&nDevices);
00020     for (int i = 0; i < nDevices; i++) {
00021         cudaDeviceProp prop;
00022         cudaGetDeviceProperties(&prop, i);
00023         std::cerr << "Device Number: " << i << "\n";
00024         std::cerr << "Device name: " << prop.name << "\n";
00025         std::cerr << "Memory Clock Rate (KHz): " << prop.memoryClockRate << "\n";
00026         std::cerr << "Memory Bus Width (bits): " << prop.memoryBusWidth << "\n";
00027         std::cerr << "Peak Memory Bandwidth (GB/s): " << 2.0 * prop.memoryClockRate *
00028             (prop.memoryBusWidth / 8) / 1.0e6 << "\n";
00029         std::cerr << "Max Linear Threads: " << prop.maxThreadsDim[0] << "\n";
00030     }
00031 }
```

9.11.3.60 MigrateMatrix()

```
host void Markov::API::CUDA::CUDAModelMatrix::MigrateMatrix () [inherited]
```

Migrate the class members to the VRAM.

Cannot be used without calling [Markov::API::ModelMatrix::ConstructMatrix](#) at least once. This function will manage the memory allocation and data transfer from CPU RAM to GPU VRAM.

Newly allocated VRAM pointers are set in the class member variables.

Definition at line 20 of file [cudaModelMatrix.cu](#).

```
00020                                     {
00021     cudastatus;
00022
00023     cudastatus = cudaMalloc((char**)&(this->device_matrixIndex),
00024         this->matrixSize*sizeof(char));
00025     CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_matrixIndex.");
00026
00027     cudastatus = cudaMalloc((long int **)&(this->device_totalEdgeWeights),
00028         this->matrixSize*sizeof(long int));
00029     CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_totalEdgeWeights.");
00030
00031     cudastatus = cudaMemcpy(this->device_matrixIndex, this->matrixIndex,
00032         this->matrixSize*sizeof(char), cudaMemcpyHostToDevice);
00033     CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Index)");
00034
00035     cudastatus = cudaMemcpy(this->device_totalEdgeWeights, this->totalEdgeWeights,
00036         this->matrixSize*sizeof(long int), cudaMemcpyHostToDevice);
00037     CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Total Edge Values)");
00038
00039     cudastatus = CudaMigrate2DFlat<char>(
00040         &(this->device_edgeMatrix), this->edgeMatrix, this->matrixSize, this->matrixSize);
00041     CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize edge matrix.");
00042
00043     cudastatus = CudaMigrate2DFlat<long int>(
00044         &(this->device_valueMatrix), this->valueMatrix, this->matrixSize, this->matrixSize);
00045     CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
00046 }
```

9.11.3.61 Nodes() [1/2]

```
std::map<char , Node<char *>>* Markov::Model< char >::Nodes () [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.11.3.62 Nodes() [2/2]

```
std::map<char , Node<char >*>* Markov::Model< char >::Nodes ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.11.3.63 OpenDatasetFile() [1/2]

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename ) [inherited]
```

Open dataset file and return the ifstream pointer.

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

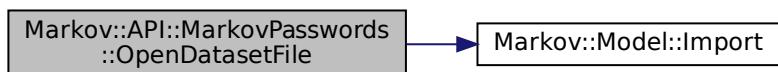
`ifstream*` to the the dataset file

Definition at line 51 of file [markovPasswords.cpp](#).

```
00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060
00061 } {
```

References [Markov::Model< NodeStorageType >::Import\(\)](#).

Here is the call graph for this function:



9.11.3.64 OpenDatasetFile() [2/2]

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename ) [inherited]
```

Open dataset file and return the ifstream pointer.

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

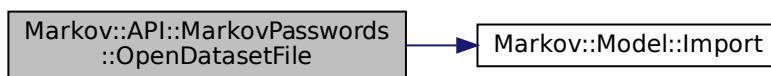
`ifstream*` to the dataset file

Definition at line 51 of file `markovPasswords.cpp`.

```
00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References `Markov::Model< NodeStorageType >::Import()`.

Here is the call graph for this function:

**9.11.3.65 OptimizeEdgeOrder() [1/2]**

`void Markov::Model< char >::OptimizeEdgeOrder [inherited]`

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file `model.h`.

```
00265
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType*>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
00269                     Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight() > rhs->EdgeWeight();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         //    std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00278 }
```

9.11.3.66 OptimizeEdgeOrder() [2/2]

`void Markov::Model< char >::OptimizeEdgeOrder [inherited]`

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file `model.h`.

```
00265
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType*>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
00269                     Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight() > rhs->EdgeWeight();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         //    std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00278 }
```

9.11.3.67 parse() [1/2]

```
def Python.Markopy.BaseCLI.parse (
    self ) [inherited]
```

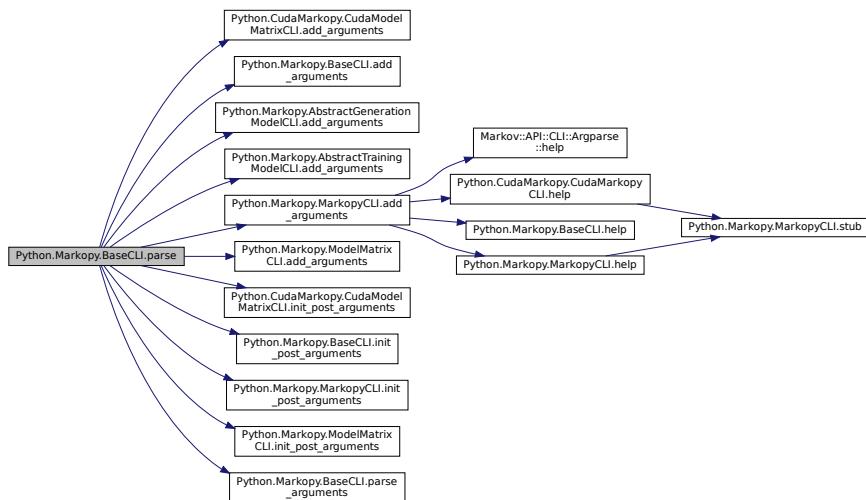
Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 55 of file [base.py](#).

```
00055     def parse(self):
00056         """! @brief add, parse and hook arguments"""
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.init_post_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.init_post_arguments\(\)](#) and [Python.Markopy.BaseCLI.parse_arguments\(\)](#).

Here is the call graph for this function:



9.11.3.68 parse() [2/2]

```
def Python.Markopy.BaseCLI.parse (
    self ) [inherited]
```

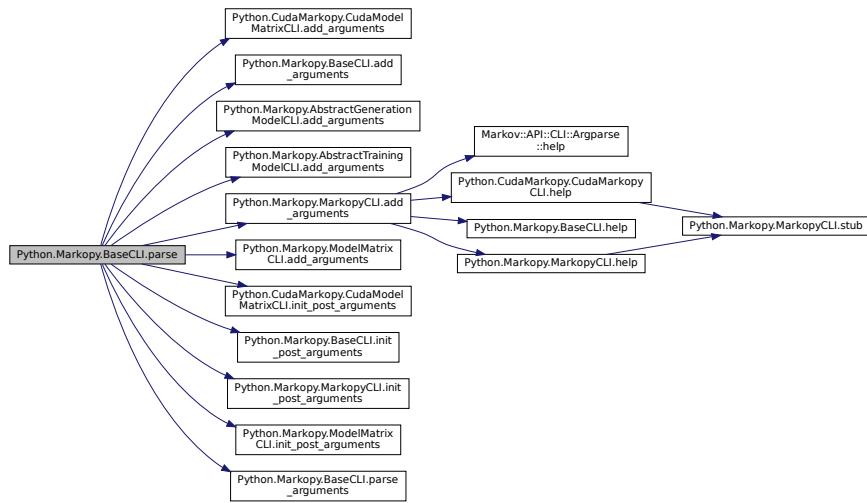
Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 55 of file [base.py](#).

```
00055     def parse(self):
00056         """! @brief add, parse and hook arguments"""
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.init_post_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.init_post_arguments\(\)](#) and [Python.Markopy.BaseCLI.parse_arguments\(\)](#).

Here is the call graph for this function:



9.11.3.69 parse_arguments() [1/2]

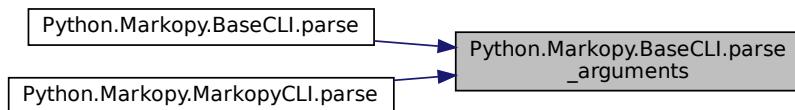
```
def Python.Markopy.BaseCLI.parse_arguments (
    self ) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.11.3.70 parse_arguments() [2/2]

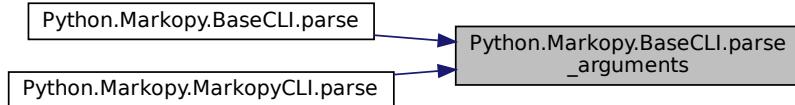
```
def Python.Markopy.BaseCLI.parse_arguments (
    self ) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.11.3.71 prepKernelMemoryChannel()

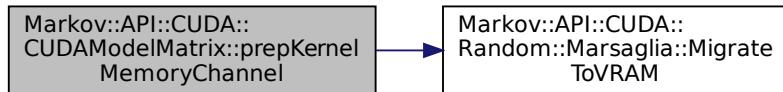
```
__host__ void Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel (
    int number_of_streams) [protected], [inherited]
```

Definition at line 145 of file [cudaModelMatrix.cu](#).

```
00145
00146
00147     this->cuadstreams = new cudaStream_t[number_of_streams];
00148     for(int i=0;i<number_of_streams;i++)
00149         cudaStreamCreate(&this->cuadstreams[i]);
00150
00151     this-> outputBuffer = new char*[number_of_streams];
00152     for(int i=0;i<number_of_streams;i++)
00153         this->outputBuffer[i]= new char[cudaPerKernelAllocationSize];
00154
00155     cudaError_t cudastatus;
00156     this-> device_outputBuffer = new char*[number_of_streams];
00157     for(int i=0;i<number_of_streams;i++){
00158         cudastatus = cudaMalloc((char**)&(device_outputBuffer[i]),
00159             cudaPerKernelAllocationSize);
00160         CudaCheckNotifyErr(cudastatus, "Failed to establish memory channel. Possibly out of
VRAM?");
00161     }
00162
00163     this-> device_seeds = new unsigned long*[number_of_streams];
00164     for(int i=0;i<number_of_streams;i++){
00165         Markov::API::CUDA::Random::Marsaglia *MEarr = new
Markov::API::CUDA::Random::Marsaglia[cudaGridSize];
00166         this->device_seeds[i] = Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(MEarr,
cudaGridSize);
00167         delete[] MEarr;
00168     }
00169 }
```

References [Markov::API::CUDA::CUDAModelMatrix::cudaGridSize](#), [Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocation](#), [Markov::API::CUDA::CUDAModelMatrix::device_outputBuffer](#), [Markov::API::CUDA::CUDAModelMatrix::device_seeds](#), [Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM\(\)](#), and [Markov::API::CUDA::CUDAModelMatrix::outputBuffer](#).

Here is the call graph for this function:



9.11.3.72 process() [1/2]

```
def Python.Markopy.BaseCLI.process (
    self) [inherited]
```

Process parameters for operation.

Reimplemented in [Python.Markopy.MarkopyCLI](#).

Definition at line 202 of file [base.py](#).

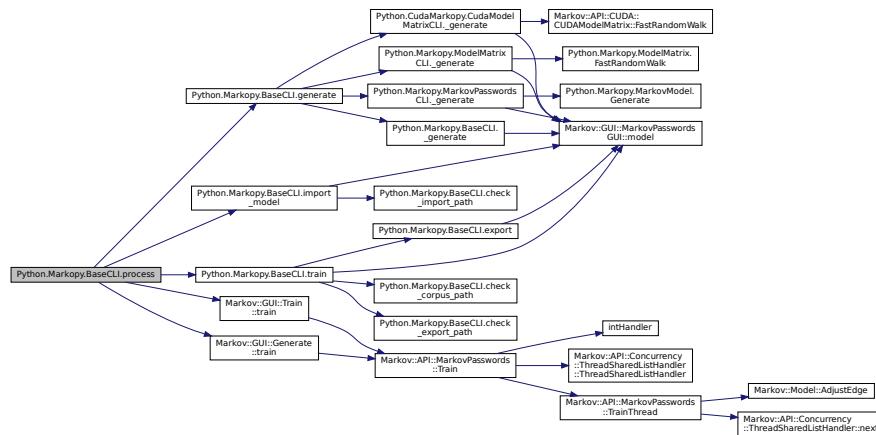
```

00202     def process(self):
00203         """
00204             @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging pprint(f"\"Bulk mode operation chosen.\", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
00210                     (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00211                     corpus_list = os.listdir(self.args.dataset)
00212                     for corpus in corpus_list:
00213                         self.import_model(self.args.input)
00214                         logging pprint(f"\"Training {self.args.input} with {corpus}\", 2)
00215                         output_file_name = corpus
00216                         model_extension = ""
00217                         if "." in self.args.input:
00218                             model_extension = self.args.input.split(".")[-1]
00219                             self.train(f"\"{self.args.dataset}/{corpus}\", self.args.seperator,
00220                                         f\"{self.args.output}/{corpus}.{model_extension}\", output_forced=True, bulk=True)
00221                         else:
00222                             logging pprint("In bulk training, output and dataset should be a directory.")
00223                             exit(1)
00224
00225             elif (self.args.mode.lower() == "generate"):
00226                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
00227                     (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00228                     model_list = os.listdir(self.args.input)
00229                     print(model_list)
00230                     for input in model_list:
00231                         logging pprint(f"\"Generating from {self.args.input}/{input} to
00232                                         {self.args.wordlist}/{input}.txt\", 2)
00233                         self.import_model(f"\"{self.args.input}/{input}\")
00234                         model_base = input
00235                         if "." in self.args.input:
00236                             model_base = input.split(".")[1]
00237                             self.generate(f"\"{self.args.wordlist}/{model_base}.txt\", bulk=True)
00238                         else:
00239                             logging pprint("In bulk generation, input and wordlist should be directory.")
00240
00241
00242             else:
00243                 self.import_model(self.args.input)
00244                 if (self.args.mode.lower() == "generate"):
00245                     self.generate(self.args.wordlist)
00246
00247             elif (self.args.mode.lower() == "train"):
00248                 self.train(self.args.dataset, self.args.seperator, self.args.output,
00249                           output_forced=True)
00250
00251             elif (self.args.mode.lower() == "combine"):
00252                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00253                 self.generate(self.args.wordlist)
00254
00255             else:
00256                 logging pprint("Invalid mode arguement given.")
00257                 logging pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00258                 exit(5)

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), [Markov::GUI::Generate.train\(\)](#), [Markov::GUI::Train.train\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



9.11.3.73 process() [2/2]

```
def Python.Markopy.BaseCLI.process ( self ) [inherited]
```

Process parameters for operation.

Reimplemented in [Python.Markopy.MarkopyCLI](#).

Definition at line 202 of file `base.py`.

```
00202     def process(self):
00203         """
00204             @brief Process parameters for operation
00205         """
00206
00207         if(self.args.bulk):
00208             logging pprint(f"\"Bulk mode operation chosen.\", 4)
00209             if (self.args.mode.lower() == "train"):
00210                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
00211 (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00212                 corpus_list = os.listdir(self.args.dataset)
00213                 for corpus in corpus_list:
00214                     self.import_model(self.args.input)
00215                     logging pprint(f"\"Training {self.args.input} with {corpus}\", 2)
00216                     output_file_name = corpus
00217                     model_extension = ""
00218                     if "." in self.args.input:
00219                         model_extension = self.args.input.split(".")[-1]
00220                         self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
00221 f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00222                     else:
00223                         logging pprint("In bulk training, output and dataset should be a directory.")
00224                         exit(1)
00225
00226             elif (self.args.mode.lower() == "generate"):
00227                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
00228 (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00229                     model_list = os.listdir(self.args.input)
00230                     print(model_list)
00231                     for input in model_list:
00232                         logging pprint(f"\"Generating from {self.args.input}/{input} to
00233 {self.args.wordlist}/{input}.txt\", 2)
00234                     self.import_model(f"{self.args.input}/{input}")
00235                     model_base = input
00236                     if "." in self.args.input:
00237                         model_base = input.split(".")[1]
00238                     self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00239                     else:
00240                         logging pprint("In bulk generation, input and wordlist should be directory.")
00241
00242             else:
00243                 self.import_model(self.args.input)
00244                 if (self.args.mode.lower() == "generate"):
00245                     self.generate(self.args.wordlist)
00246
00247                 elif (self.args.mode.lower() == "train"):
```

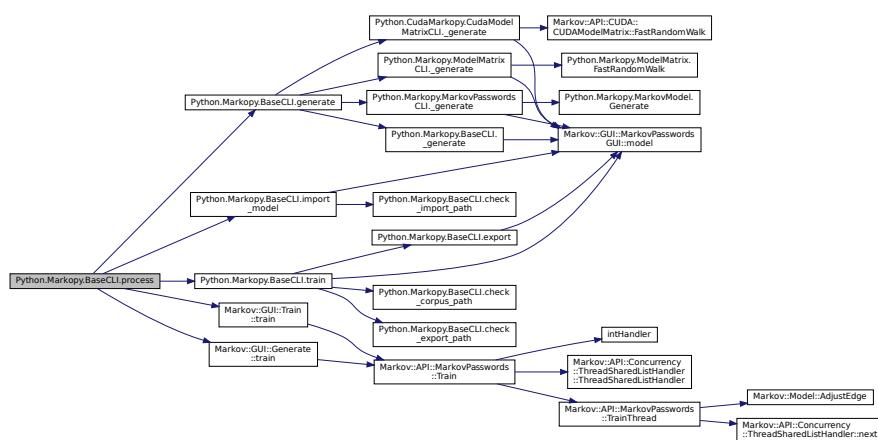
```

00244         self.train(self.args.dataset, self.args.seperator, self.args.output,
00245         output_forced=True)
00246
00247     elif(self.args.mode.lower() == "combine"):
00248         self.train(self.args.dataset, self.args.seperator, self.args.output)
00249         self.generate(self.args.wordlist)
00250
00251     else:
00252         logging pprint("Invalid mode arguement given.")
00253         logging pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00254         exit(5)
00255
00256

```

References [Python.CudaMarkopy.CudaModelMatrixCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), [Markov::GUI::Generate.train\(\)](#), [Markov::GUI::Train.train\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



9.11.3.74 RandomWalk() [1/2]

```

char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]

```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from [This library](#) is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```

Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}

```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```

00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319         else if (temp_node == NULL) {
00320             break;
00321         }
00322         n = temp_node;
00323
00324         buffer[len++] = n->NodeValue();
00325
00326     //null terminate the string
00327     buffer[len] = 0x00;
00328
00329     //do something with the generated string
00330     return buffer; //for now
00331
00332
00333
00334 }
```

9.11.3.75 RandomWalk() [2/2]

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
```

```
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```

00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext (randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL) {
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue ();
00327     }
00328
00329 //null terminate the string
00330 buffer[len] = 0x00;
00331
00332 //do something with the generated string
00333 return buffer; //for now
00334 }
```

9.11.3.76 Save() [1/2]

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
```

Export model to file.

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

```

00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export (exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.11.3.77 Save() [2/2]

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
Export model to file.
```

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

```
00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.11.3.78 StarterNode() [1/2]

```
Node< char *>* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.11.3.79 StarterNode() [2/2]

```
Node< char *>* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.11.3.80 Train() [1/2]

```
void Markov::API::ModelMatrix::Train (
    const char * datasetFileName,
    char delimiter,
    int threads ) [inherited]
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

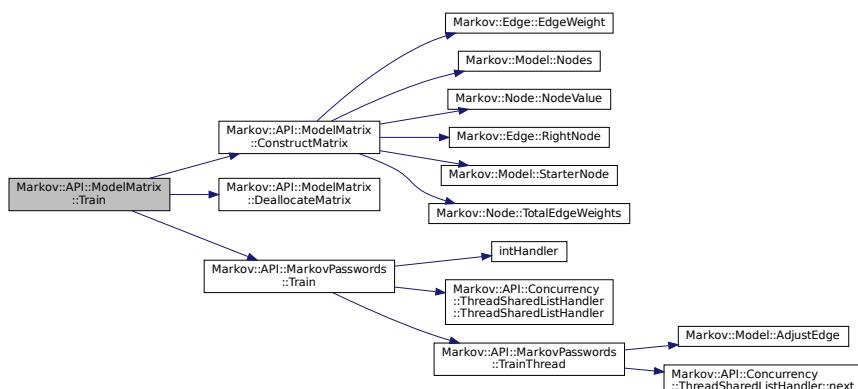
Construct the matrix when done.

Definition at line 25 of file [modelMatrix.cpp](#).

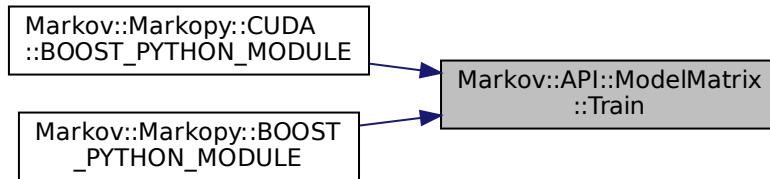
```
00025
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.81 Train() [2/2]

```
void Markov::API::ModelMatrix::Train (
    const char * datasetFileName,
    char delimiter,
    int threads ) [inherited]
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

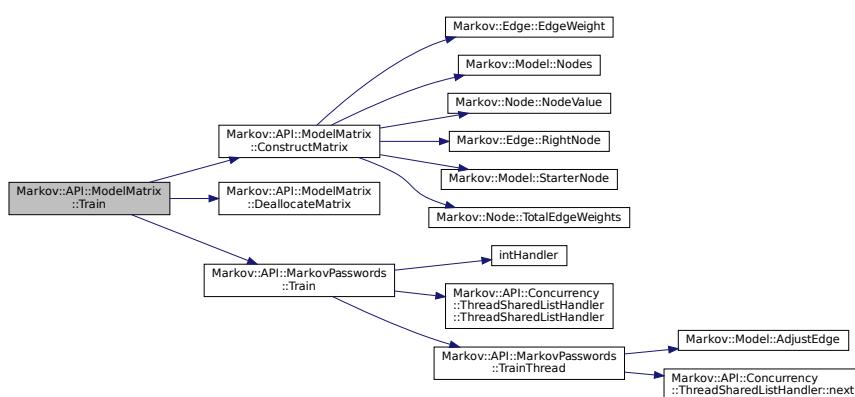
Construct the matrix when done.

Definition at line 25 of file [modelMatrix.cpp](#).

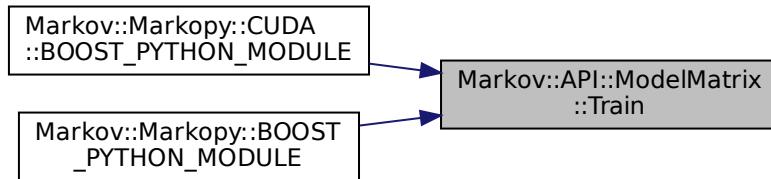
```
00025
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#). Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.82 train() [1/2]

```
def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False )  [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104         """
00105         logging pprint("Training.")
00106
00107         if not (dataset and seperator and (output or not output_forced)):
00108             logging pprint(f"Training mode requires -d/--dataset(', -o/--output' if output_forced
00109             else') and -s/--seperator parameters. Exiting.")
00110             return False
00111
00112         if not bulk and not self.check_corpus_path(dataset):
00113             logging pprint(f"\{dataset\} doesn't exists. Check the file path, or working directory")
00114             return False
00115
00116         if not self.check_export_path(output):
00117             logging pprint(f"Cannot create output at {output}")
00118             return False
00119
00120         if(seperator == '\\t'):
00121             logging pprint("Escaping seperator.", 3)
00122             seperator = '\t'
```

```

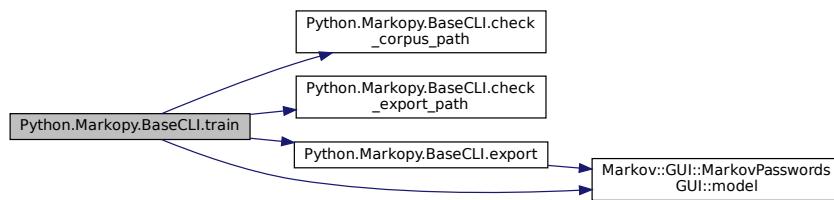
00122         if(len(seperator)!=1):
00123             logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00124             accepted.')
00125             exit(4)
00126
00127         logging pprint(f'Starting training.', 3)
00128         self.model.Train(dataset,seperator, int(self.args.threads))
00129         logging pprint(f'Training completed.', 2)
00130
00131         if(output):
00132             logging pprint(f'Exporting model to {output}', 2)
00133             self.export(output)
00134         else:
00135             logging pprint(f'Model will not be exported.', 1)
00136
00137     return True

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.83 train() [2/2]

```

def Python.Markopy.BaseCLI.train (
        self,
        str dataset,
        str seperator,
        str output,
        bool output_forced = False,
        bool bulk = False )  [inherited]

```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename

Parameters

<code>output_forced</code>	force overwrite
<code>bulk</code>	marks bulk operation with directories

Definition at line 94 of file `base.py`.

```

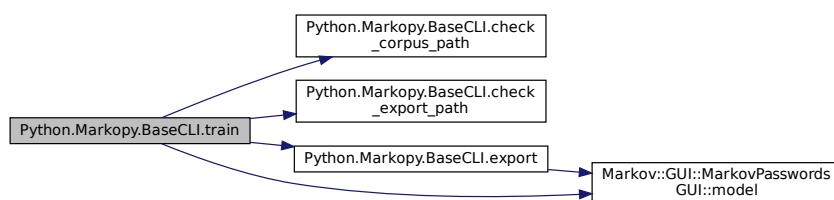
00094     def train(self, dataset : str, separator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096             """
00097                 @brief Train a model via CLI parameters
00098                 @param model Model instance
00099                 @param dataset filename for the dataset
00100                 @param separator separator used with the dataset
00101                 @param output output filename
00102                 @param output_forced force overwrite
00103                 @param bulk marks bulk operation with directories
00104             """
00105             logging pprint("Training.")
00106             if not (dataset and separator and (output or not output_forced)):
00107                 logging pprint(f"Training mode requires -d/--dataset', -o/--output' if output_forced
00108                 else") and -s/-separators parameters. Exiting.")
00109             return False
00110             if not bulk and not self.check_corpus_path(dataset):
00111                 logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112                 return False
00113
00114             if not self.check_export_path(output):
00115                 logging pprint(f"Cannot create output at {output}")
00116                 return False
00117
00118             if(separator == '\\t'):
00119                 logging pprint("Escaping separator.", 3)
00120                 separator = '\t'
00121
00122             if(len(separator)!=1):
00123                 logging pprint(f'Delimiter must be a single character, and "{separator}" is not
00124                 accepted.')
00125                 exit(4)
00126
00127             logging pprint(f'Starting training.', 3)
00128             self.model.Train(dataset,separator, int(self.args.threads))
00129             logging pprint(f'Training completed.', 2)
00130
00131             if(output):
00132                 logging pprint(f'Exporting model to {output}', 2)
00133                 self.export(output)
00134             else:
00135                 logging pprint(f'Model will not be exported.', 1)
00136
00137             return True
00138

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.3.84 TrainThread()

```
void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private], [inherited]
```

A single thread invoked by the Train function.

Parameters

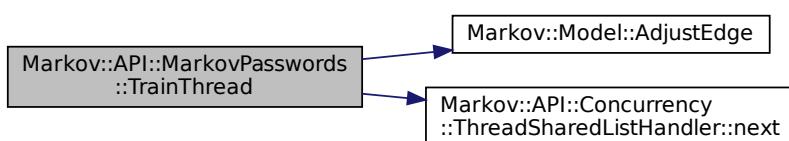
<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

Definition at line 85 of file [markovPasswords.cpp](#).

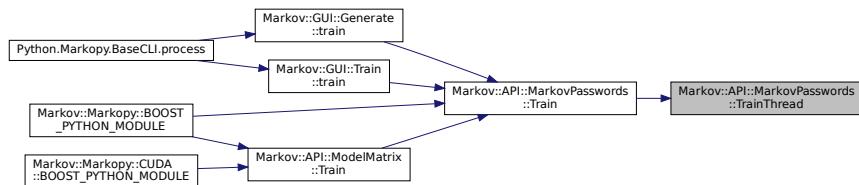
```
00085
00086     {
00087         char format_str[] = "%ld,%s";
00088         format_str[3]=delimiter;
00089         std::string line;
00090         while (listhandler->next(&line) && keepRunning) {
00091             long int oc;
00092             if (line.size() > 100) {
00093                 line = line.substr(0, 100);
00094             }
00095             char* linebuf = new char[line.length()+5];
00096             #ifdef _WIN32
00097             sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
00098             "%ld,%s"
00099             #else
00100             sscanf(line.c_str(), format_str, &oc, linebuf);
00101             #endif
00102             this->AdjustEdge((const char*)linebuf, oc);
00103             delete linebuf;
00104         }
00105     }
```

References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), [keepRunning](#), and [Markov::API::Concurrency::ThreadSharedListHandler::next](#). Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.11.4 Member Data Documentation

9.11.4.1 alternatingKernels

`int Markov::API::CUDA::CUDAModelMatrix::alternatingKernels [private], [inherited]`
Definition at line 135 of file [cudaModelMatrix.h](#).

Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#).

9.11.4.2 args [1/2]

`Python.Markopy.BaseCLI.args [inherited]`

Definition at line 75 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.MarkopyCLI.help\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.parse\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.parse_fail\(\)](#), [Python.Markopy.BaseCLI.process\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.11.4.3 args [2/2]

`Python.Markopy.BaseCLI.args [inherited]`

Definition at line 75 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.MarkopyCLI.help\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.parse\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.parse_fail\(\)](#), [Python.Markopy.BaseCLI.process\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.11.4.4 bInfinite

`Python.CudaMarkopy.CudaModelMatrixCLI.bInfinite`

Definition at line 73 of file [cudammx.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#).

9.11.4.5 cudaBlocks

`int Markov::API::CUDA::CUDAModelMatrix::cudaBlocks [private], [inherited]`

Definition at line 125 of file [cudaModelMatrix.h](#).

Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#).

9.11.4.6 cudaGridSize

```
int Markov::API::CUDA::CUDAModelMatrix::cudaGridSize [private], [inherited]  
Definition at line 131 of file cudaModelMatrix.h.
```

Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#), and [Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel\(\)](#).

9.11.4.7 cudaMemPerGrid

```
int Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid [private], [inherited]  
Definition at line 132 of file cudaModelMatrix.h.
```

Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#), and [Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernel\(\)](#).

9.11.4.8 cudaPerKernelAllocationSize

```
long int Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize [private], [inherited]  
Definition at line 133 of file cudaModelMatrix.h.
```

Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#), [Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernel\(\)](#), and [Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel\(\)](#).

9.11.4.9 cudastreams

```
cudaStream_t* Markov::API::CUDA::CUDAModelMatrix::cudastreams [private], [inherited]  
Definition at line 139 of file cudaModelMatrix.h.
```

9.11.4.10 cudaThreads

```
int Markov::API::CUDA::CUDAModelMatrix::cudaThreads [private], [inherited]  
Definition at line 126 of file cudaModelMatrix.h.
```

Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#).

9.11.4.11 datasetFile

```
std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]  
Definition at line 123 of file markovPasswords.h.
```

9.11.4.12 device_edgeMatrix

```
char* Markov::API::CUDA::CUDAModelMatrix::device_edgeMatrix [private], [inherited]  
VRAM Address pointer of edge matrix (from modelMatrix.h)  
Definition at line 88 of file cudaModelMatrix.h.
```

9.11.4.13 device_matrixIndex

```
char* Markov::API::CUDA::CUDAModelMatrix::device_matrixIndex [private], [inherited]  
VRAM Address pointer of matrixIndex (from modelMatrix.h)  
Definition at line 98 of file cudaModelMatrix.h.
```

9.11.4.14 device_outputBuffer

```
char** Markov::API::CUDA::CUDAModelMatrix::device_outputBuffer [private], [inherited]  
RandomWalk results in device.  
Definition at line 108 of file cudaModelMatrix.h.
```

Referenced by [Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel\(\)](#).

9.11.4.15 device_seeds

`unsigned long** Markov::API::CUDA::CUDAModelMatrix::device_seeds [private], [inherited]`
Definition at line [137](#) of file [cudaModelMatrix.h](#).

Referenced by [Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel\(\)](#).

9.11.4.16 device_totalEdgeWeights

`long int* Markov::API::CUDA::CUDAModelMatrix::device_totalEdgeWeights [private], [inherited]`
VRAM Address pointer of total edge weights (from [modelMatrix.h](#))
Definition at line [103](#) of file [cudaModelMatrix.h](#).

9.11.4.17 device_valueMatrix

`long int* Markov::API::CUDA::CUDAModelMatrix::device_valueMatrix [private], [inherited]`
VRAM Address pointer of value matrix (from [modelMatrix.h](#))
Definition at line [93](#) of file [cudaModelMatrix.h](#).

9.11.4.18 edgeMatrix [1/2]

`char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]`
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line [175](#) of file [modelMatrix.h](#).
Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.11.4.19 edgeMatrix [2/2]

`char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]`
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line [175](#) of file [modelMatrix.h](#).
Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.11.4.20 edges

`std::vector<Edge<char >> Markov::Model< char >::edges [private], [inherited]`
A list of all edges in this model.
Definition at line [204](#) of file [model.h](#).

9.11.4.21 fileIO

`Python.Markopy.ModelMatrixCLI.fileIO [inherited]`
Definition at line [38](#) of file [mmx.py](#).
Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), and [Python.Markopy.ModelMatrixCLI._generate\(\)](#).

9.11.4.22 flatEdgeMatrix

```
char* Markov::API::CUDA::CUDAModelMatrix::flatEdgeMatrix [private], [inherited]
Adding Edge matrix end-to-end and resize to 1-D array for better perfomance on traversing.
Definition at line 118 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix\(\).
```

9.11.4.23 flatValueMatrix

```
long int* Markov::API::CUDA::CUDAModelMatrix::flatValueMatrix [private], [inherited]
Adding Value matrix end-to-end and resize to 1-D array for better perfomance on traversing.
Definition at line 123 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix\(\).
```

9.11.4.24 iterationsPerKernelThread

```
int Markov::API::CUDA::CUDAModelMatrix::iterationsPerKernelThread [private], [inherited]
Definition at line 127 of file cudaModelMatrix.h.
Referenced by Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\).
```

9.11.4.25 matrixIndex [1/2]

```
char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]
to hold the Matrix index (To hold the orders of 2-D arrays')
Definition at line 190 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix\(\), Markov::API::ModelMatrix::DeallocateMatrix\(\),
Markov::API::ModelMatrix::DumpJSON\(\), and Markov::API::ModelMatrix::FastRandomWalkThread\(\).
```

9.11.4.26 matrixIndex [2/2]

```
char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]
to hold the Matrix index (To hold the orders of 2-D arrays')
Definition at line 190 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix\(\), Markov::API::ModelMatrix::DeallocateMatrix\(\),
Markov::API::ModelMatrix::DumpJSON\(\), and Markov::API::ModelMatrix::FastRandomWalkThread\(\).
```

9.11.4.27 matrixSize [1/2]

```
int Markov::API::ModelMatrix::matrixSize [protected], [inherited]
to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix\(\), Markov::API::ModelMatrix::DeallocateMatrix\(\),
Markov::API::ModelMatrix::DumpJSON\(\), Markov::API::ModelMatrix::FastRandomWalkThread\(\), and Markov::API::CUDA::CUDAMoo
```

9.11.4.28 matrixSize [2/2]

```
int Markov::API::ModelMatrix::matrixSize [protected], [inherited]
to hold Matrix size
Definition at line 185 of file modelMatrix.h.
Referenced by Markov::API::ModelMatrix::ConstructMatrix\(\), Markov::API::ModelMatrix::DeallocateMatrix\(\),
Markov::API::ModelMatrix::DumpJSON\(\), Markov::API::ModelMatrix::FastRandomWalkThread\(\), and Markov::API::CUDA::CUDAMoo
```

9.11.4.29 model

`Python.CudaMarkopy.CudaModelMatrixCLI.model`

Definition at line 65 of file `cudammx.py`.

Referenced by `Python.CudaMarkopy.CudaModelMatrixCLI._generate()`, `Python.Markopy.BaseCLI._generate()`, `Python.Markopy.ModelMatrixCLI._generate()`, `Python.Markopy.MarkovPasswordsCLI._generate()`, `Python.Markopy.BaseCLI.export()`, `Python.Markopy.BaseCLI.import_model()`, and `Python.Markopy.BaseCLI.train()`.

9.11.4.30 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`
Dataset file input of our system

Definition at line 124 of file `markovPasswords.h`.

9.11.4.31 nodes

`std::map<char , Node<char *>> Markov::Model< char >::nodes [private], [inherited]`

Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.

Definition at line 193 of file `model.h`.

9.11.4.32 numberOfPartitions

`int Markov::API::CUDA::CUDAModelMatrix::numberOfPartitions [private], [inherited]`

Definition at line 130 of file `cudaModelMatrix.h`.

Referenced by `Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk()`.

9.11.4.33 outputBuffer

`char** Markov::API::CUDA::CUDAModelMatrix::outputBuffer [private], [inherited]`

RandomWalk results in host.

Definition at line 113 of file `cudaModelMatrix.h`.

Referenced by `Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput()`, and `Markov::API::CUDA::CUDAModelMatrix::pr`

9.11.4.34 outputFile

`std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]`

File to save model of our system

Definition at line 125 of file `markovPasswords.h`.

9.11.4.35 parser [1/2]

`Python.Markopy.BaseCLI.parser [inherited]`

Definition at line 25 of file `base.py`.

Referenced by `Python.CudaMarkopy.CudaMarkopyCLI.__init__()`, `Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments()`, `Python.Markopy.BaseCLI.add_arguments()`, `Python.Markopy.AbstractGenerationModelCLI.add_arguments()`, `Python.Markopy.AbstractTrainingModelCLI.add_arguments()`, `Python.Markopy.MarkopyCLI.add_arguments()`, `Python.Markopy.ModelMatrixCLI.add_arguments()`, `Python.CudaMarkopy.CudaMarkopyCLI.help()`, and `Python.Markopy.MarkopyCLI`.

9.11.4.36 parser [2/2]

Python.Markopy.BaseCLI.parser [inherited]

Definition at line 25 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.11.4.37 print_help [1/2]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.11.4.38 print_help [2/2]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.11.4.39 ready [1/2]

bool Markov::API::ModelMatrix::ready [protected], [inherited]

True when matrix is constructed. False if not.

Definition at line 200 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::ModelMatrix\(\)](#).

9.11.4.40 ready [2/2]

bool Markov::API::ModelMatrix::ready [protected], [inherited]

True when matrix is constructed. False if not.

Definition at line 200 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::ModelMatrix\(\)](#).

9.11.4.41 starterNode

Node<char *>* Markov::Model< char >::starterNode [private], [inherited]

Starter Node of this model.

Definition at line 198 of file [model.h](#).

9.11.4.42 totalEdgeWeights [1/2]

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]

Array of the Total Edge Weights.

Definition at line 195 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.11.4.43 totalEdgeWeights [2/2]

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]
 Array of the Total Edge Weights.
 Definition at line 195 of file [modelMatrix.h](#).
 Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.11.4.44 totalOutputPerKernel

long int Markov::API::CUDA::CUDAModelMatrix::totalOutputPerKernel [private], [inherited]
 Definition at line 129 of file [cudaModelMatrix.h](#).
 Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#).

9.11.4.45 totalOutputPerSync

long int Markov::API::CUDA::CUDAModelMatrix::totalOutputPerSync [private], [inherited]
 Definition at line 128 of file [cudaModelMatrix.h](#).
 Referenced by [Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk\(\)](#).

9.11.4.46 valueMatrix [1/2]

long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]
 2-d Integer array for the value Matrix (For the weights of Edges)
 Definition at line 180 of file [modelMatrix.h](#).
 Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.11.4.47 valueMatrix [2/2]

long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]
 2-d Integer array for the value Matrix (For the weights of Edges)
 Definition at line 180 of file [modelMatrix.h](#).
 Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).
 The documentation for this class was generated from the following file:

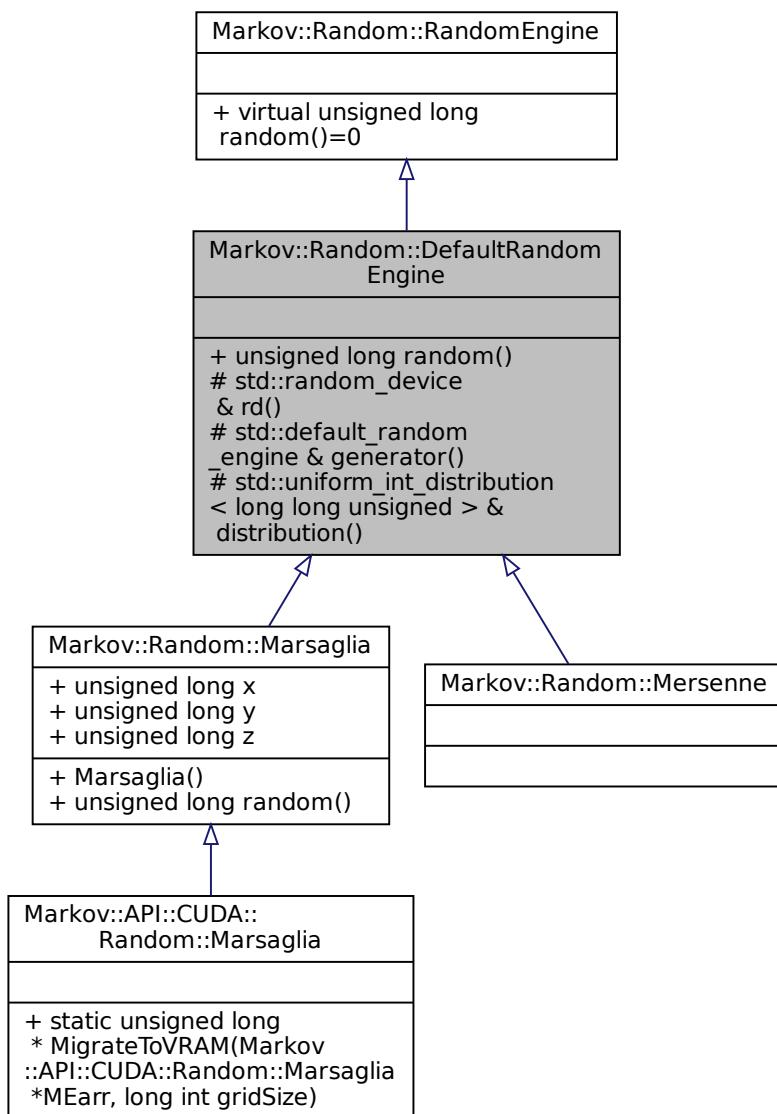
- [Markopy/CudaMarkopy/src/CLI/cudammx.py](#)

9.12 Markov::Random::DefaultRandomEngine Class Reference

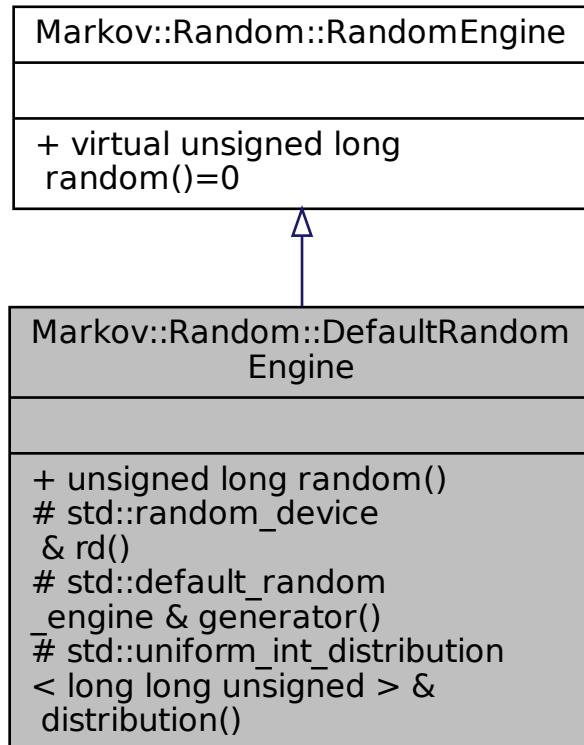
Implementation using [Random.h](#) default random engine.

```
#include <random.h>
```

Inheritance diagram for Markov::Random::DefaultRandomEngine:



Collaboration diagram for Markov::Random::DefaultRandomEngine:



Public Member Functions

- `unsigned long random ()`
Generate Random Number.

Protected Member Functions

- `std::random_device & rd ()`
Default random device for seeding.
- `std::default_random_engine & generator ()`
Default random engine for seeding.
- `std::uniform_int_distribution< long long unsigned > & distribution ()`
Distribution schema for seeding.

9.12.1 Detailed Description

Implementation using `Random.h` default random engine.
This engine is also used by other engines for seeding.

Example Use: Using Default Engine with RandomWalk

```

Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::DefaultRandomEngine randomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&randomEngine, 5, 10, res);
  
```

```

    std::cout << res << "\n";
}

Example Use: Generating a random number with Marsaglia Engine
Markov::Random::DefaultRandomEngine de;
std::cout << de.random();
Definition at line 61 of file random.h.

```

9.12.2 Member Function Documentation

9.12.2.1 distribution()

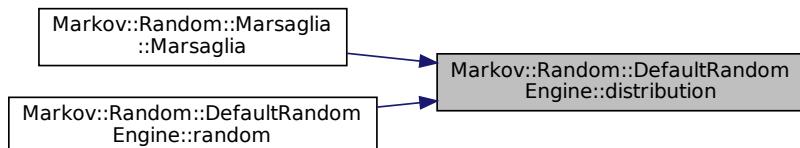
```

std::uniform_int_distribution<long long unsigned>& Markov::Random::DefaultRandomEngine::distribution
( ) [inline], [protected]
Distribution schema for seeding.
Definition at line 90 of file random.h.
00090
00091     static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffffff);
00092     return _distribution;
00093 }

```

Referenced by [Markov::Random::Marsaglia::Marsaglia\(\)](#), and [random\(\)](#).

Here is the caller graph for this function:



9.12.2.2 generator()

```

std::default_random_engine& Markov::Random::DefaultRandomEngine::generator ( ) [inline],
[protected]
Default random engine for seeding.
Definition at line 82 of file random.h.

```

```

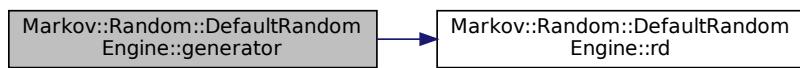
00082
00083     static std::default_random_engine _generator(rd());
00084     return _generator;
00085 }

```

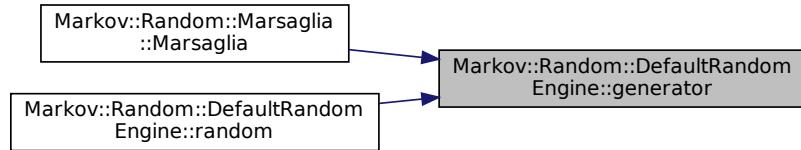
References [rd\(\)](#).

Referenced by [Markov::Random::Marsaglia::Marsaglia\(\)](#), and [random\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.2.3 random()

`unsigned long Markov::Random::DefaultRandomEngine::random () [inline], [virtual]`
Generate Random Number.

Returns

random number in long range.

Implements [Markov::Random::RandomEngine](#).

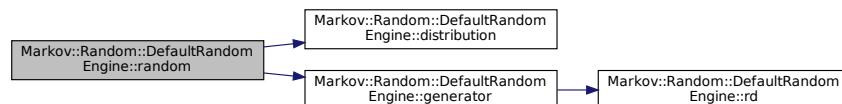
Reimplemented in [Markov::Random::Marsaglia](#).

Definition at line 66 of file [random.h](#).

```
00066     {
00067         return this->distribution() (this->generator());
00068     }
```

References [distribution\(\)](#), and [generator\(\)](#).

Here is the call graph for this function:



9.12.2.4 rd()

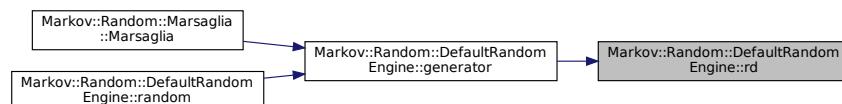
`std::random_device& Markov::Random::DefaultRandomEngine::rd () [inline], [protected]`
Default random device for seeding.

Definition at line 74 of file [random.h](#).

```
00074     static std::random_device _rd;
00075     return _rd;
00076 }
00077 }
```

Referenced by [generator\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

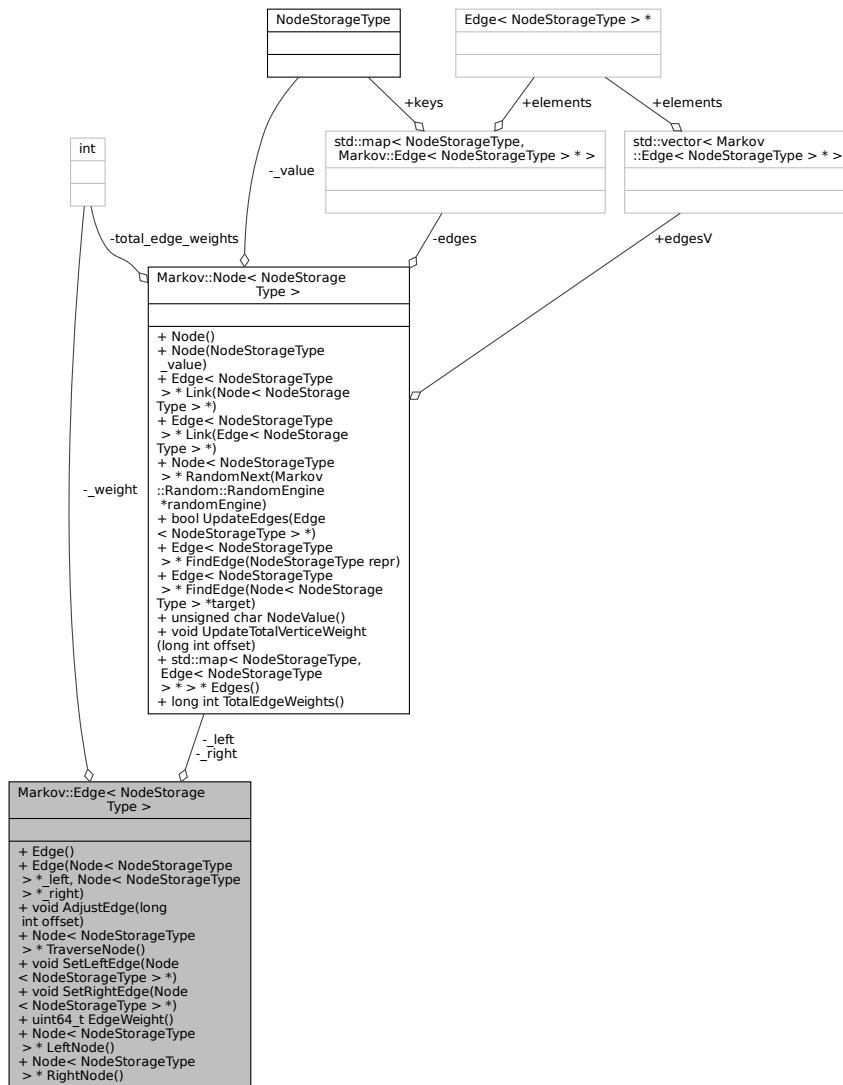
- Markopy/MarkovModel/src/random.h

9.13 Markov::Edge< NodeStorageType > Class Template Reference

[Edge](#) class used to link nodes in the model together.

```
#include <edge.h>
```

Collaboration diagram for Markov::Edge< NodeStorageType >:



Public Member Functions

- [Edge \(\)](#)
Default constructor.
- [Edge \(Node< NodeStorageType > * _left, Node< NodeStorageType > * _right\)](#)
Constructor. Initialize edge with given RightNode and LeftNode.
- [void AdjustEdge \(long int offset\)](#)
Adjust the edge EdgeWeight with offset. Adds the offset parameter to the edge EdgeWeight.
- [Node< NodeStorageType > * TraverseNode \(\)](#)
Traverse this edge to RightNode.

- void `SetLeftEdge (Node< NodeStorageType > *)`
`Set LeftNode of this edge.`
- void `SetRightEdge (Node< NodeStorageType > *)`
`Set RightNode of this edge.`
- uint64_t `EdgeWeight ()`
`return edge's EdgeWeight.`
- `Node< NodeStorageType > * LeftNode ()`
`return edge's LeftNode`
- `Node< NodeStorageType > * RightNode ()`
`return edge's RightNode`

Private Attributes

- `Node< NodeStorageType > * _left`
`source node`
- `Node< NodeStorageType > * _right`
`target node`
- long int `_weight`
`Edge Edge Weight.`

9.13.1 Detailed Description

```
template<typename NodeStorageType>
class Markov::Edge< NodeStorageType >
```

`Edge` class used to link nodes in the model together.

Has `LeftNode`, `RightNode`, and `EdgeWeight` of the edge. Edges are *UNIDIRECTIONAL* in this model. They can only be traversed `LeftNode` to `RightNode`.

Definition at line 23 of file `edge.h`.

9.13.2 Constructor & Destructor Documentation

9.13.2.1 `Edge()` [1/2]

```
template<typename NodeStorageType >
Markov::Edge< NodeStorageType >::Edge
```

Default constructor.

Definition at line 123 of file `edge.h`.

```
00123
00124     this->_left = NULL;
00125     this->_right = NULL;
00126     this->_weight = 0;
00127 }
```

9.13.2.2 `Edge()` [2/2]

```
template<typename NodeStorageType >
Markov::Edge< NodeStorageType >::Edge (
    Markov::Node< NodeStorageType > * _left,
    Markov::Node< NodeStorageType > * _right )
```

Constructor. Initialize edge with given `RightNode` and `LeftNode`.

Parameters

<code>_left</code>	- Left node of this edge.
<code>_right</code>	- Right node of this edge.

Example Use: Construct edge

```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Edge<unsigned char>* el = new Markov::Edge<unsigned char>(src, target1);
```

Definition at line 130 of file [edge.h](#).

```
00130 {
00131     this->_left = _left;
00132     this->_right = _right;
00133     this->_weight = 0;
00134 }
```

9.13.3 Member Function Documentation

9.13.3.1 AdjustEdge()

```
template<typename NodeStorageType >
void Markov::Edge< NodeStorageType >::AdjustEdge (
    long int offset )
```

Adjust the edge EdgeWeight with offset. Adds the offset parameter to the edge EdgeWeight.

Parameters

<i>offset</i>	- NodeValue to be added to the EdgeWeight
---------------	---

Example Use: Construct edge

```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Edge<unsigned char>* el = new Markov::Edge<unsigned char>(src, target1);
el->AdjustEdge(25);
```

Definition at line 137 of file [edge.h](#).

```
00137 {
00138     this->_weight += offset;
00139     this->LeftNode()->UpdateTotalVerticeWeight(offset);
00140 }
```

Referenced by [Markov::API::MarkovPasswords::Buff\(\)](#).

Here is the caller graph for this function:



9.13.3.2 EdgeWeight()

```
template<typename NodeStorageType >
uint64_t Markov::Edge< NodeStorageType >::EdgeWeight [inline]
return edge's EdgeWeight.
```

Returns

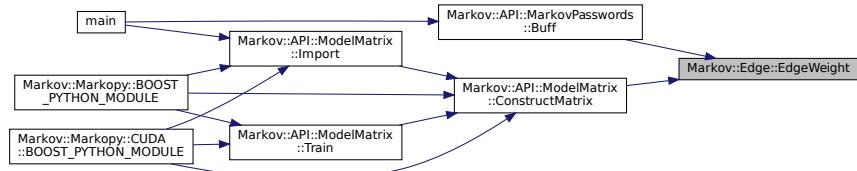
edge's EdgeWeight.

Definition at line 160 of file [edge.h](#).

```
00160 {
00161     return this->_weight;
00162 }
```

Referenced by [Markov::API::MarkovPasswords::Buff\(\)](#), and [Markov::API::ModelMatrix::ConstructMatrix\(\)](#).

Here is the caller graph for this function:



9.13.3.3 LeftNode()

```
template<typename NodeStorageType >
Markov::Node< NodeStorageType > * Markov::Edge< NodeStorageType >::LeftNode
return edge's LeftNode
```

Returns

edge's LeftNode.

Definition at line 165 of file [edge.h](#).

```
00165
00166     return this->_left;
00167 }
```

9.13.3.4 RightNode()

```
template<typename NodeStorageType >
Markov::Node< NodeStorageType > * Markov::Edge< NodeStorageType >::RightNode [inline]
return edge's RightNode
```

Returns

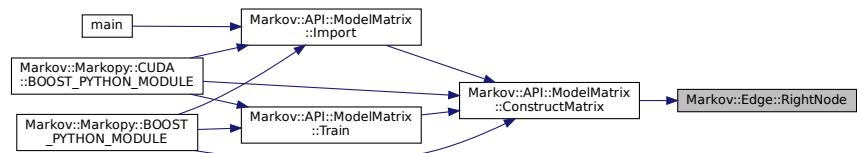
edge's RightNode.

Definition at line 170 of file [edge.h](#).

```
00170
00171     return this->_right;
00172 }
```

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#).

Here is the caller graph for this function:



9.13.3.5 SetLeftEdge()

```
template<typename NodeStorageType >
void Markov::Edge< NodeStorageType >::SetLeftEdge (
    Markov::Node< NodeStorageType > * n )
```

Set LeftNode of this edge.

Parameters

<code>node</code>	- Node to be linked with.
-------------------	---

Definition at line 150 of file [edge.h](#).

```
00150
00151     this->_left = n;
00152 }
```

9.13.3.6 SetRightEdge()

```
template<typename NodeStorageType >
void Markov::Edge< NodeStorageType >::SetRightEdge (
    Markov::Node< NodeStorageType > * n )
```

Set RightNode of this edge.

Parameters

<code>node</code>	- Node to be linked with.
-------------------	---

Definition at line 155 of file [edge.h](#).

```
00155
00156     this->_right = n;
00157 }
```

9.13.3.7 TraverseNode()

```
template<typename NodeStorageType >
Markov::Node< NodeStorageType > * Markov::Edge< NodeStorageType >::TraverseNode [inline]
Traverse this edge to RightNode.
```

Returns

Right node. If this is a terminator node, return NULL

Example Use: Traverse a node

```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
e1->AdjustEdge(25);
Markov::Edge<unsigned char>* e2 = e1->traverseNode();
```

Definition at line 143 of file [edge.h](#).

```
00143
00144     if (this->RightNode() ->NodeValue() == 0xff) //terminator node
00145         return NULL;
00146     return _right;
00147 }
```

References [Markov::Edge< NodeStorageType >::_right](#).

9.13.4 Member Data Documentation**9.13.4.1 _left**

```
template<typename NodeStorageType >
Node<NodeStorageType>* Markov::Edge< NodeStorageType >::_left [private]
source node
Definition at line 105 of file edge.h.
```

9.13.4.2 _right

```
template<typename NodeStorageType >
Node<NodeStorageType>* Markov::Edge< NodeStorageType >::_right [private]
target node
Definition at line 110 of file edge.h.
Referenced by Markov::Edge< NodeStorageType >::TraverseNode\(\).
```

9.13.4.3 _weight

```
template<typename NodeStorageType >
long int Markov::Edge< NodeStorageType >::_weight [private]
Edge Edge Weight.
Definition at line 115 of file edge.h.
The documentation for this class was generated from the following files:
```

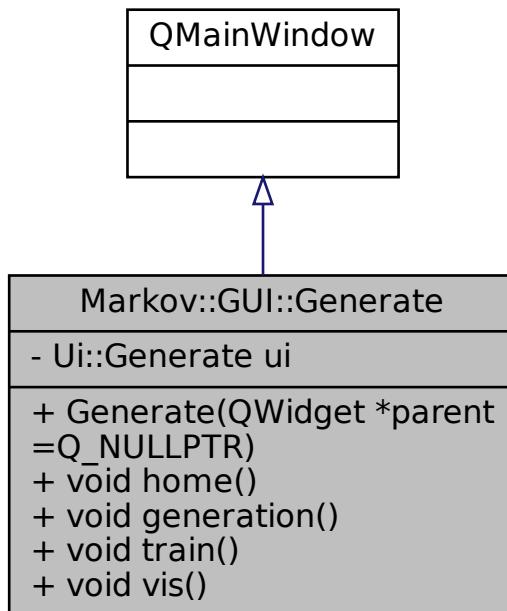
- [Markopy/MarkovModel/src/model.h](#)
- [Markopy/MarkovModel/src/edge.h](#)

9.14 Markov::GUI::Generate Class Reference

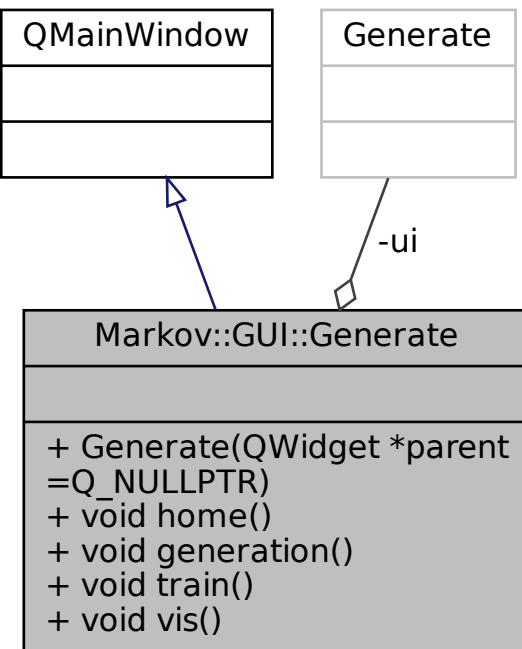
QT Generation page class.

```
#include <Generate.h>
```

Inheritance diagram for Markov::GUI::Generate:



Collaboration diagram for Markov::GUI::Generate:



Public Slots

- void `home ()`
- void `generation ()`
- void `train ()`
- void `vis ()`

Public Member Functions

- `Generate (QWidget *parent=Q_NULLPTR)`

Private Attributes

- `Ui::Generate ui`

9.14.1 Detailed Description

QT Generation page class.
Definition at line 15 of file [Generate.h](#).

9.14.2 Constructor & Destructor Documentation

9.14.2.1 Generate()

```
Generate::Generate (
    QWidget * parent = Q_NULLPTR )
```

Definition at line 20 of file [Generate.cpp](#).

```
00021     : QMainWindow(parent)
00022 {
00023     ui.setupUi(this);
00024
00025     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {generation(); });
00026     QObject::connect(ui.pushButton_4, &QPushButton::clicked, this, [this] {home(); });
00027     QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {train(); });
00028     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {vis(); });
00029
00030     ui.pushButton->setVisible(false);
00031     ui.lineEdit->setVisible(false);
00032     ui.lineEdit_2->setVisible(false);
00033     ui.lineEdit_3->setVisible(false);
00034     ui.label_3->setVisible(false);
00035     ui.label_4->setVisible(false);
00036     ui.label_5->setVisible(false);
00037
00038
00039
00040 }
```

References [ui](#).

9.14.3 Member Function Documentation

9.14.3.1 generation

```
void Generate::generation ( ) [slot]
```

Definition at line 42 of file [Generate.cpp](#).

```
00042     {
00043
00044
00045
00046
00047     QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00048     QFile file(file_name);
00049
00050
00051
00052     int numberPass = ui.lineEdit->text().toInt();
00053     int minLen = ui.lineEdit_2->text().toInt();
00054     int maxLen = ui.lineEdit_3->text().toInt();
00055     char* cstr;
00056     std::string fname = file_name.toStdString();
00057     cstr = new char[fname.size() + 1];
00058     strcpy(cstr, fname.c_str());
00059
00060     ui.label_6->setText("GENERATING!");
00061
00062     Markov::API::MarkovPasswords mp;
00063     mp.Import("src\CLI\sample_models\2gram-trained.mdl");
00064
00065     mp.Generate(numberPass, cstr, minLen, maxLen);
00066
00067     if (!file.open(QFile::ReadOnly | QFile::Text)) {
00068         QMessageBox::warning(this, "Error", "File Not Open!");
00069     }
00070     QTextStream in(&file);
00071     QString text = in.readAll();
00072     ui.plainTextEdit->setPlainText(text);
00073
00074     ui.label_6->setText("DONE!");
00075
00076
00077
00078     file.close();
00079 }
```

References [Markov::API::MarkovPasswords::Generate\(\)](#), and [ui](#).

Here is the call graph for this function:



9.14.3.2 home

```
void Generate::home ( ) [slot]
Definition at line 121 of file Generate.cpp.
00121
00122     {
00123         CLI* w = new CLI;
00124         w->show();
00125         this->close();
00125 }
```

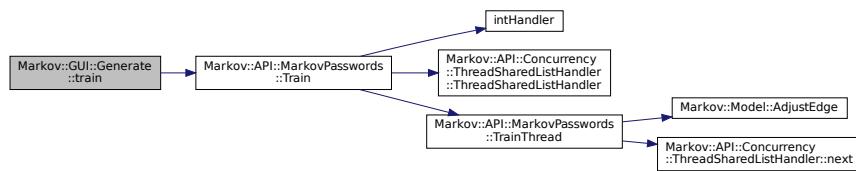
9.14.3.3 train

```
void Generate::train ( ) [slot]
Definition at line 82 of file Generate.cpp.
00082
00083     {
00084         QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00085         QFile file(file_name);
00086
00087         if (!file.open(QFile::ReadOnly | QFile::Text)) {
00088             QMessageBox::warning(this, "Error", "File Not Open!");
00089         }
00090         QTextStream in(&file);
00091         QString text = in.readAll();
00092
00093         char* cstr;
00094         std::string fname = file_name.toStdString();
00095         cstr = new char[fname.size() + 1];
00096         strcpy(cstr, fname.c_str());
00097
00098
00099
00100         char a = ',';
00101         Markov::API::MarkovPasswords mp;
00102         mp.Import("models\\2gram.mdl");
00103         mp.Train(cstr, a, 10);
00104         mp.Export("models\\finished.mdl");
00105
00106
00107
00108         ui.pushButton->setVisible(true);
00109         ui.lineEdit->setVisible(true);
00110         ui.lineEdit_2->setVisible(true);
00111         ui.lineEdit_3->setVisible(true);
00112         ui.label_3->setVisible(true);
00113         ui.label_4->setVisible(true);
00114         ui.label_5->setVisible(true);
00115
00116         file.close();
00117
00118
00119 }
```

References [Markov::API::MarkovPasswords::Train\(\)](#), and [ui](#).

Referenced by [Python.Markopy.BaseCLI::process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.14.3.4 vis

```

void Generate::vis ( ) [slot]
Definition at line 126 of file Generate.cpp.
00126 {
00127     MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
00128     w->show();
00129     this->close();
00130 }
  
```

9.14.4 Member Data Documentation

9.14.4.1 ui

```

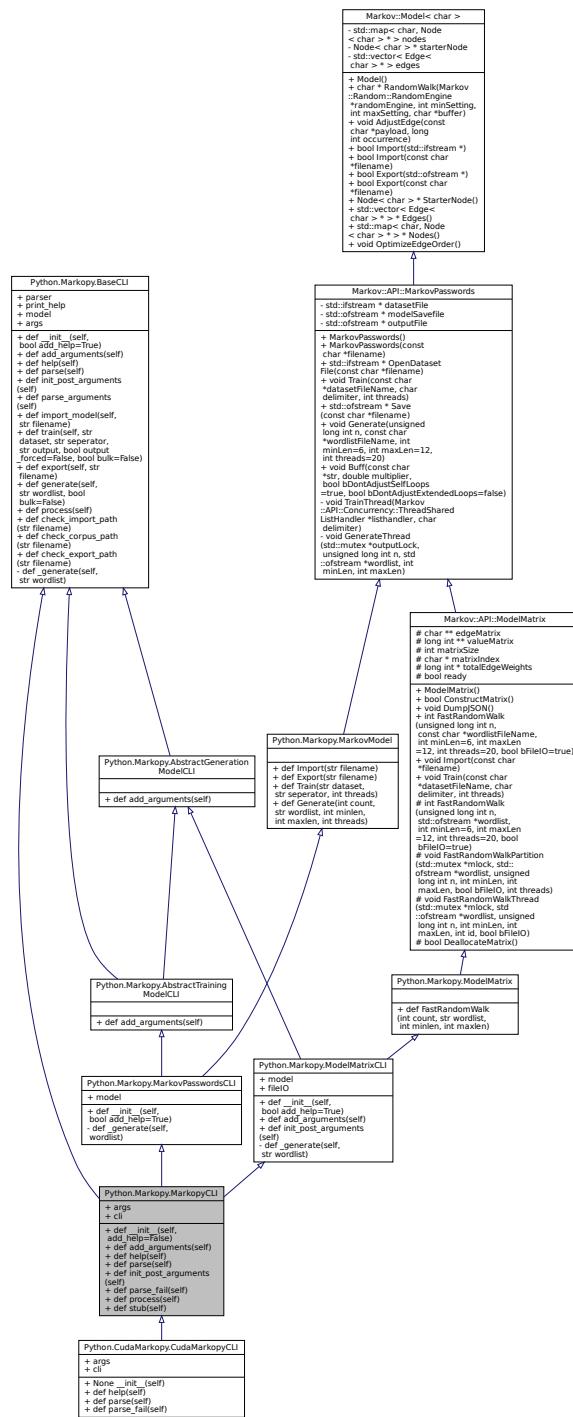
Ui::Generate Markov::GUI::Generate::ui [private]
Definition at line 21 of file Generate.h.
Referenced by Generate\(\), generation\(\), and train\(\).
The documentation for this class was generated from the following files:
  
```

- [Markopy/MarkovPasswordsGUI/src/Generate.h](#)
- [Markopy/MarkovPasswordsGUI/src/Generate.cpp](#)

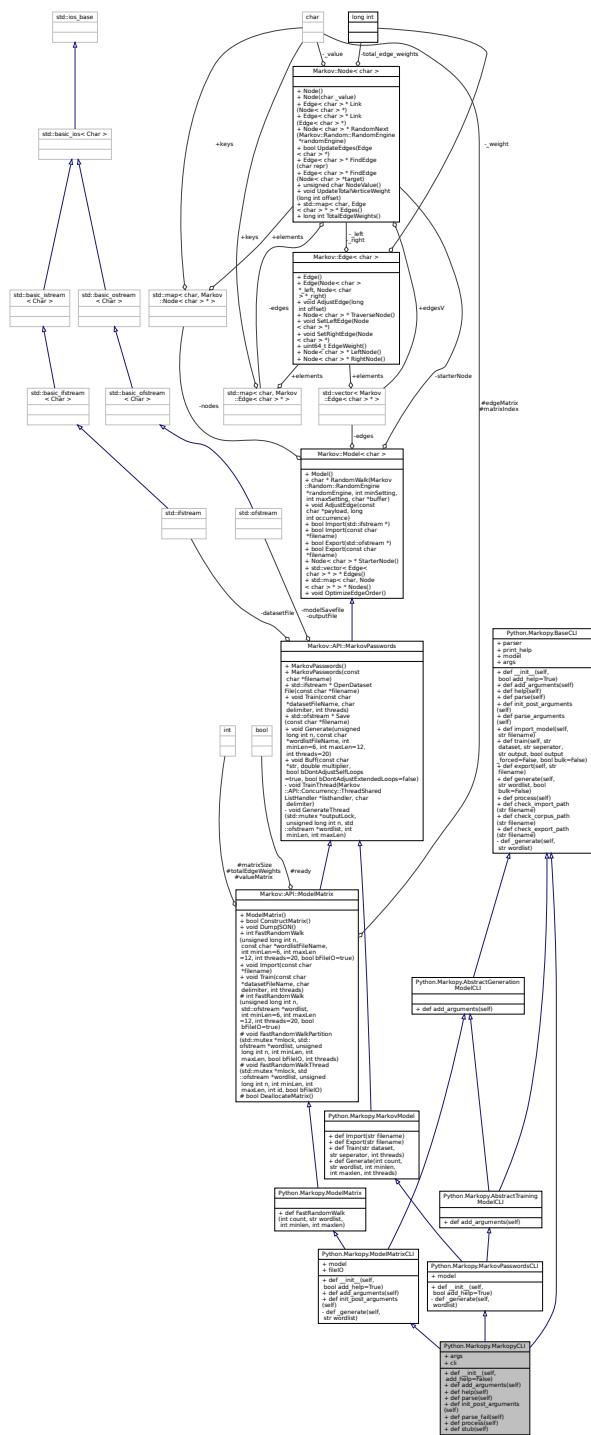
9.15 Python.Markopy.MarkopyCLI Class Reference

Top level model selector for [Markopy](#) CLI.

Inheritance diagram for Python.Markopy.MarkopyCLI:



Collaboration diagram for Python.Markopy.MarkopyCLI:



Public Member Functions

- def `__init__` (self, add_help=False)
default constructor
 - def `add_arguments` (self)
add -mt/-model_type constructor
 - def `help` (self)
overload help function to print submodel helps

- def `parse` (self)
 - def `init_post_arguments` (self)
 - def `parse_fail` (self)
 - def `process` (self)
 - Process parameters for operation.*
- def `stub` (self)
- def `parse_arguments` (self)
- def `import_model` (self, str filename)
 - Import a model file.*
- def `train` (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)
 - Train a model via CLI parameters.*
- def `export` (self, str filename)
 - Export model to a file.*
- def `generate` (self, str wordlist, bool bulk=False)
 - Generate strings from the model.*
- def `parse_arguments` (self)
- def `import_model` (self, str filename)
 - Import a model file.*
- def `train` (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)
 - Train a model via CLI parameters.*
- def `export` (self, str filename)
 - Export model to a file.*
- def `generate` (self, str wordlist, bool bulk=False)
 - Generate strings from the model.*
- def `FastRandomWalk` (int count, str wordlist, int minlen, int maxlen)
 - Random walk on the Matrix-reduced `Markov::Model`.*
- int `FastRandomWalk` (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)
- bool `ConstructMatrix` ()
 - Construct the related Matrix data for the model.*
- void `DumpJSON` ()
 - Debug function to dump the model to a JSON file.*
- void `Import` (const char *filename)
 - Open a file to import with filename, and call `bool Model::Import` with `std::ifstream`.*
- bool `Import` (`std::ifstream` *)
 - Import a file to construct the model.*
- void `Train` (const char *datasetFileName, char delimiter, int threads)
 - Train the model with the dataset file.*
- `std::ifstream` * `OpenDatasetFile` (const char *filename)
 - Open dataset file and return the `ifstream` pointer.*
- `std::ofstream` * `Save` (const char *filename)
 - Export model to file.*
- void `Generate` (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)
 - Call `Markov::Model::RandomWalk` n times, and collect output.*
- void `Buff` (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)
 - Buff expression of some characters in the model.*
- char * `RandomWalk` (`Markov::Random::RandomEngine` *randomEngine, int minSetting, int maxSetting, char *buffer)
 - Do a random walk on this model.*
- void `AdjustEdge` (const char *payload, long int occurrence)

- Adjust the model with a single string.*
- bool **Export** (std::ofstream *)
Export a file of the model.
 - bool **Export** (const char *filename)
Open a file to export with filename, and call bool Model::Export with std::ofstream.
 - Node<char> * **StarterNode** ()
Return starter Node.
 - std::vector<Edge<char> *> * **Edges** ()
Return a vector of all the edges in the model.
 - std::map<char, Node<char> *> * **Nodes** ()
Return starter Node.
 - void **OptimizeEdgeOrder** ()
Sort edges of all nodes in the model ordered by edge weights.
 - def **parse_arguments** (self)
• def **parse_arguments** (self)
• def **import_model** (self, str filename)
Import a model file.
 - def **import_model** (self, str filename)
Import a model file.
 - def **train** (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)
Train a model via CLI parameters.
 - def **train** (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)
Train a model via CLI parameters.
 - def **export** (self, str filename)
Export model to a file.
 - def **export** (self, str filename)
Export model to a file.
 - def **generate** (self, str wordlist, bool bulk=False)
Generate strings from the model.
 - def **generate** (self, str wordlist, bool bulk=False)
Generate strings from the model.
 - def **Import** (str filename)
• bool **Import** (std::ifstream *)
Import a file to construct the model.
 - def **Export** (str filename)
• bool **Export** (std::ofstream *)
Export a file of the model.
 - bool **Export** (const char *filename)
Open a file to export with filename, and call bool Model::Export with std::ofstream.
 - def **Train** (str dataset, str seperator, int threads)
• def **Generate** (int count, str wordlist, int minlen, int maxlen, int threads)
• void **Generate** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)
Call Markov::Model::RandomWalk n times, and collect output.
 - std::ifstream * **OpenDatasetFile** (const char *filename)
Open dataset file and return the ifstream pointer.
 - std::ofstream * **Save** (const char *filename)
Export model to file.
 - void **Buff** (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)
Buff expression of some characters in the model.

- char * [RandomWalk](#) ([Markov::Random::RandomEngine](#) *randomEngine, int minSetting, int maxSetting, char *buffer)
Do a random walk on this model.
- void [AdjustEdge](#) (const char *payload, long int occurrence)
Adjust the model with a single string.
- Node< char > * [StarterNode](#) ()
Return starter Node.
- std::vector< Edge< char > * > * [Edges](#) ()
Return a vector of all the edges in the model.
- std::map< char, Node< char > * > * [Nodes](#) ()
Return starter Node.
- void [OptimizeEdgeOrder](#) ()
Sort edges of all nodes in the model ordered by edge weights.

Static Public Member Functions

- def [check_import_path](#) (str filename)
check import path for validity
- def [check_corpus_path](#) (str filename)
check import path for validity
- def [check_export_path](#) (str filename)
check import path for validity
- def [check_import_path](#) (str filename)
check import path for validity
- def [check_corpus_path](#) (str filename)
check import path for validity
- def [check_export_path](#) (str filename)
check import path for validity
- def [check_import_path](#) (str filename)
check import path for validity
- def [check_import_path](#) (str filename)
check import path for validity
- def [check_corpus_path](#) (str filename)
check import path for validity
- def [check_corpus_path](#) (str filename)
check import path for validity
- def [check_export_path](#) (str filename)
check import path for validity
- def [check_export_path](#) (str filename)
check import path for validity

Public Attributes

- args
- cli
- parser
- print_help
- model
- model
- fileIO
- parser
- print_help

- `model`
- `parser`
- `parser`
- `print_help`
- `print_help`

Protected Member Functions

- int `FastRandomWalk` (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced `Markov::Model`.
- void `FastRandomWalkPartition` (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

A single partition of `FastRandomWalk` event.
- void `FastRandomWalkThread` (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)

A single thread of a single partition of `FastRandomWalk`.
- bool `DeallocateMatrix` ()

Deallocate matrix and make it ready for re-construction.

Protected Attributes

- char ** `edgeMatrix`

2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** `valueMatrix`

2-d Integer array for the value Matrix (For the weights of Edges)
- int `matrixSize`

to hold Matrix size
- char * `matrixIndex`

to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * `totalEdgeWeights`

Array of the Total `Edge` Weights.
- bool `ready`

True when matrix is constructed. False if not.

Private Member Functions

- def `_generate` (self, str wordlist)

wrapper for generate function.
- void `TrainThread` (`Markov::API::Concurrency::ThreadSharedListHandler` *listhandler, char delimiter)

A single thread invoked by the Train function.
- void `GenerateThread` (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)

A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * `datasetFile`
- std::ofstream * `modelSavefile`

Dataset file input of our system
- std::ofstream * `outputFile`

File to save model of our system

- std::map< char, Node< char > * > **nodes**
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > * **starterNode**
Starter Node of this model.
- std::vector< Edge< char > * > **edges**
A list of all edges in this model.

9.15.1 Detailed Description

Top level model selector for [Markopy](#) CLI.

This class is used for injecting the -mt parameter to the CLI, and determining the model type depending on that. Definition at line 59 of file [markopy.py](#).

9.15.2 Constructor & Destructor Documentation

9.15.2.1 __init__()

```
def Python.Markopy.MarkopyCLI.__init__ (
    self,
    add_help = False )
default constructor
Definition at line 69 of file markopy.py.
00069     def __init__(self, add_help=False):
00070         """
00071             @brief default constructor
00072         """
00073
00074     BaseCLI.__init__(self,add_help)
00075     self.args = None
00076     self.parser.epilog = f"""
00077         {colored("Sample runs:", "yellow")}
00078         {__file__.split("/")[-1]} -mt MP generate trained.mdl -n 500 -w output.txt
00079             Import trained.mdl, and generate 500 lines to output.txt
00080
00081         {__file__.split("/")[-1]} -mt MMX generate trained.mdl -n 500 -w output.txt
00082             Import trained.mdl, and generate 500 lines to output.txt
00083         """
00084
```

9.15.3 Member Function Documentation

9.15.3.1 _generate()

```
def Python.Markopy.BaseCLI._generate (
    self,
    str wordlist ) [private], [inherited]
wrapper for generate function.
```

This can be overloaded by other models

Parameters

wordlist	filename to generate to
-----------------	-------------------------

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

Definition at line 161 of file [base.py](#).

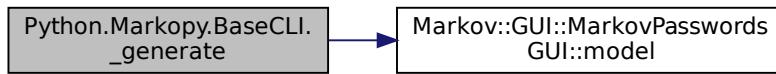
```
00161     def _generate(self, wordlist : str):
00162         """
00163             @brief wrapper for generate function. This can be overloaded by other models
00164             @param wordlist filename to generate to
00165         """
00166         self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
```

```
    int (self.args.threads))
00167
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.2 add_arguments()

```
def Python.Markopy.MarkopyCLI.add_arguments (
    self )
```

add -mt/-model_type constructor

Reimplemented from [Python.Markopy.BaseCLI](#).

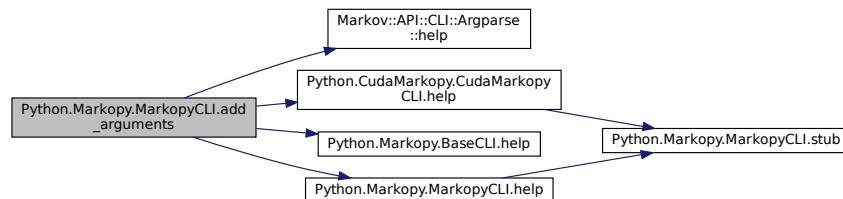
Definition at line 86 of file [markopy.py](#).

```
00086     def add_arguments(self):
00087         """
00088             @brief add -mt/--model_type constructor
00089         """
00090         self.parser.add_argument("-mt", "--model_type", default="_MMX", help="Model type to use.
Accepted values: MP, MMX")
00091         self.parser.add_argument("-h", "--help", action="store_true", help="Model type to use.
Accepted values: MP, MMX")
00092         self.parser.print_help = self.help
00093
```

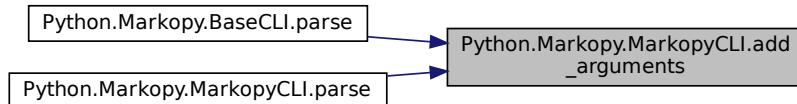
References [Markov::API::CLI::Argparse.help\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), [Python.Markopy.BaseCLI.help\(\)](#), [Python.Markopy.MarkopyCLI.help\(\)](#), and [Python.Markopy.BaseCLI.parser](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.3 AdjustEdge() [1/2]

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.15.3.4 AdjustEdge() [2/2]

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

<i>string</i>	- String that is passed from the training, and will be used to AdjustEdge the model with
<i>occurrence</i>	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.15.3.5 Buff() [1/2]

```
void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]
```

Buff expression of some characters in the model.

Parameters

<i>str</i>	A string containing all the characters to be buffed
<i>multiplier</i>	A constant value to buff the nodes with.
<i>bDontAdjustSelfEdges</i>	Do not adjust weights if target node is same as source node
<i>bDontAdjustExtendedLoops</i>	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

```
00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges) {
00162             if(buffstr.find(targetrepr)!= std::string::npos) {
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops) {
00165                     if(buffstr.find(repr)!= std::string::npos) {
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
```

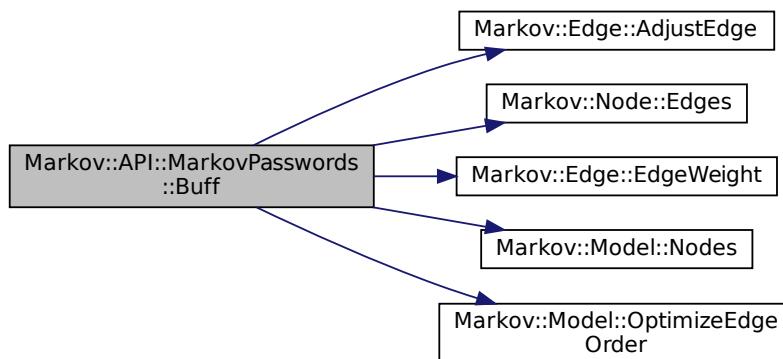
```

00172         }
00173     }
00174     i++;
00175 }
00176 this->OptimizeEdgeOrder();
00179 }
```

References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.6 Buff() [2/2]

```

void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false )  [inherited]
```

Buff expression of some characters in the model.

Parameters

<i>str</i>	A string containing all the characters to be buffed
<i>multiplier</i>	A constant value to buff the nodes with.
<i>bDontAdjustSelfEdges</i>	Do not adjust weights if target node is same as source node

Parameters

<code>bDontAdjustExtendedLoops</code>	Do not adjust if both source and target nodes are in first parameter
---------------------------------------	--

Definition at line 153 of file [markovPasswords.cpp](#).

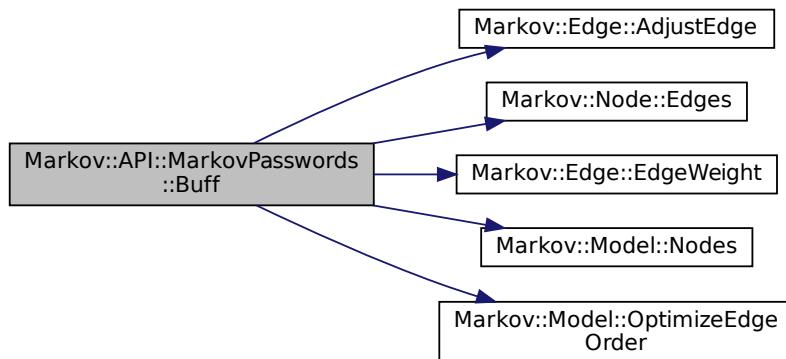
```

00153         {
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos) {
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops) {
00165                     if(buffstr.find(repr)!= std::string::npos) {
00166                         continue;
00167                     }
00168                     long int weight = edge->EdgeWeight();
00169                     weight = weight*multiplier;
00170                     edge->AdjustEdge(weight);
00171                 }
00172             }
00173         }
00174         i++;
00175     }
00176 }
00177
00178 this->OptimizeEdgeOrder();
00179 }
```

References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.7 check_corpus_path() [1/4]

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.15.3.8 check_corpus_path() [2/4]

```
def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190 
```

```

00185      """
00186
00187     if(not os.path.isfile(filename)):
00188         return False
00189     return True
00190

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.15.3.9 check_corpus_path() [3/4]

```

def Python.Markopy.BaseCLI.check_corpus_path (
        str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```

00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187     if(not os.path.isfile(filename)):
00188         return False
00189     return True
00190

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.15.3.10 check_corpus_path() [4/4]

```

def Python.Markopy.BaseCLI.check_corpus_path (
        str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```

00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185

```

```

00185     """
00186
00187     if(not os.path.isfile(filename)):
00188         return False
00189     return True
00190

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.15.3.11 check_export_path() [1/4]

```

def Python.Markopy.BaseCLI.check_export_path (
        str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```

00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.15.3.12 check_export_path() [2/4]

```

def Python.Markopy.BaseCLI.check_export_path (
        str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```

00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check

```

```

00196      """
00197
00198     if(filename and os.path.isfile(filename)):
00199         return True
00200     return True
00201

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.15.3.13 check_export_path() [3/4]

```

def Python.Markopy.BaseCLI.check_export_path (
        str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```

00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.15.3.14 check_export_path() [4/4]

```

def Python.Markopy.BaseCLI.check_export_path (
        str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```

00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196

```

```

00196     """
00197
00198     if(filename and os.path.isfile(filename)):
00199         return True
00200     return False
00201

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.15.3.15 check_import_path() [1/4]

```

def Python.Markopy.BaseCLI.check_import_path (
        str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

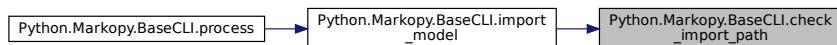
```

00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179

```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.15.3.16 check_import_path() [2/4]

```

def Python.Markopy.BaseCLI.check_import_path (
        str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```

00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172

```

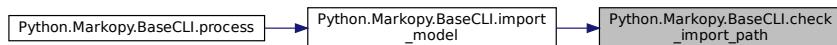
```

00172     @param filename filename to check
00173     """
00174
00175     if(not os.path.isfile(filename)):
00176         return False
00177     else:
00178         return True
00179

```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.15.3.17 check_import_path() [3/4]

```

def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```

00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175     if(not os.path.isfile(filename)):
00176         return False
00177     else:
00178         return True
00179

```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.15.3.18 check_import_path() [4/4]

```

def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```

00169     def check_import_path(filename : str):

```

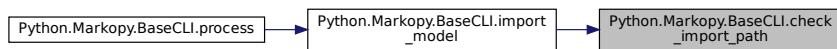
```

00170     """
00171     @brief check import path for validity
00172     @param filename filename to check
00173     """
00174
00175     if(not os.path.isfile(filename)):
00176         return False
00177     else:
00178         return True
00179

```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.15.3.19 ConstructMatrix()

```
bool Markov::API::ModelMatrix::ConstructMatrix () [inherited]
```

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content.

this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

True if constructed. False if already constructed.

Definition at line 31 of file [modelMatrix.cpp](#).

```

00031
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++) {
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++) {
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072         }
00073     }
00074 }

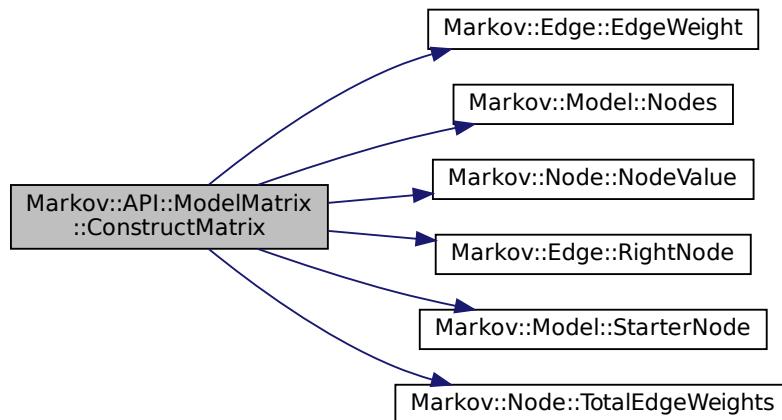
```

```

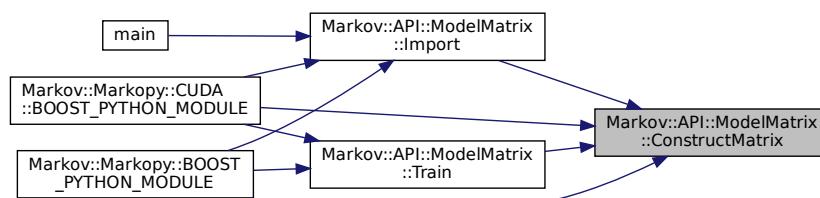
00071         }
00072     }
00073     i++;
00074 }
00075 this->ready = true;
00076 return true;
00077 //this->DumpJSON();
00078 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::API::ModelMatrix::Markov::API::ModelMatrix::matrixSize](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), [Markov::Node< storageType >::NodeValue\(\)](#), [Markov::API::ModelMatrix::ready](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::StarterNode\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), [Markov::Node< storageType >::TotalEdgeWeights\(\)](#), and [Markov::API::ModelMatrix::va](#). Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.20 DeallocateMatrix()

```
bool Markov::API::ModelMatrix::DeallocateMatrix () [protected], [inherited]
```

Deallocate matrix and make it ready for re-construction.

Returns

True if deallocated. False if matrix was not initialized

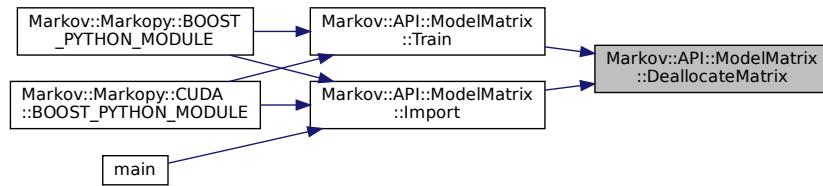
Definition at line 81 of file modelMatrix.cpp.

```
00081     if(!this->ready) return false;
00082     delete[] this->matrixIndex;
00083     delete[] this->totalEdgeWeights;
00084
00085     for(int i=0;i<this->matrixSize;i++) {
00086         delete[] this->edgeMatrix[i];
00087     }
00088     delete[] this->edgeMatrix;
00089
00090     for(int i=0;i<this->matrixSize;i++) {
00091         delete[] this->valueMatrix[i];
00092     }
00093     delete[] this->valueMatrix;
00094
00095     this->matrixSize = -1;
00096     this->ready = false;
00097     return true;
00098 }
00099 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::API::ModelMatrix::ready](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the caller graph for this function:

**9.15.3.21 DumpJSON()**

`void Markov::API::ModelMatrix::DumpJSON () [inherited]`

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

Definition at line 101 of file modelMatrix.cpp.

```
00101     {
00102
00103     std::cout << "{\n      \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++) {
00105         if(this->matrixIndex[i]=='\r') std::cout << "\\\\";";
00106         else if(this->matrixIndex[i]=='\n') std::cout << "\\\\\\";";
00107         else if(this->matrixIndex[i]==0) std::cout << "\\\\\x00";
00108         else if(i==0) std::cout << "\\\\\xff";
00109         else if(this->matrixIndex[i]=='\r\n') std::cout << "\\n";
00110         else std::cout << this->matrixIndex[i];
00111     }
00112     std::cout <<
00113     "\\",\\n"
00114     "  \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++) {
00117         if(this->matrixIndex[i]=='\r') std::cout << "      \\\\\\\\"": "[";
00118         else if(this->matrixIndex[i]=='\n') std::cout << "      \\\\\\\\"": "[";
00119         else if(this->matrixIndex[i]==0) std::cout << "      \\\\\x00": "[";
00120         else if(this->matrixIndex[i]<0) std::cout << "      \\\\\x0f": "[";
00121         else std::cout << "      \"": "[";
00122         for(int j=0;j<this->matrixSize;j++) {
00123             if(this->edgeMatrix[i][j]=='\r') std::cout << "\\\\"": "";
00124             else if(this->edgeMatrix[i][j]=='\n') std::cout << "\\\\"": "";
00125             else if(this->edgeMatrix[i][j]==0) std::cout << "\\\x00": "";
00126             else if(this->edgeMatrix[i][j]<0) std::cout << "\\\x0f": "";
00127             else if(this->matrixIndex[i]=='\r\n') std::cout << "\\\n": "
```

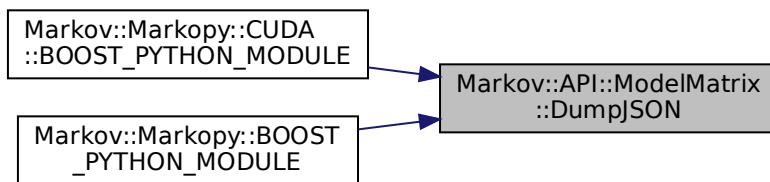
```

00128     else std::cout << "\\" << this->edgeMatrix[i][j] << "\\";
00129     if(j!=this->matrixSize-1) std::cout << ", ";
00130   }
00131   std::cout << "],\n";
00132 }
00133 std::cout << "},\n";
00134
00135 std::cout << "\" weightmap\": {\n";
00136 for(int i=0;i<this->matrixSize;i++){
00137   if(this->matrixIndex[i]=='"') std::cout << "      \"\\\\\"\\": [";
00138   else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\\\\\\": [";
00139   else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\\\\\\\x00\\": [";
00140   else if(this->matrixIndex[i]<0) std::cout << "      \"\\\\\\\\\\xff\\": [";
00141   else std::cout << "      \" \" << this->matrixIndex[i] << "\": [";
00142
00143   for(int j=0;j<this->matrixSize;j++){
00144     std::cout << this->valueMatrix[i][j];
00145     if(j!=this->matrixSize-1) std::cout << ", ";
00146   }
00147   std::cout << "],\n";
00148 }
00149 std::cout << " }\n}\n";
00150 }

```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the caller graph for this function:



9.15.3.22 Edges() [1/2]

`std::vector<Edge<char *>>* Markov::Model< char >::Edges () [inline], [inherited]`
Return a vector of all the edges in the model.

Returns

`vector of edges`

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.15.3.23 Edges() [2/2]

`std::vector<Edge<char *>>* Markov::Model< char >::Edges () [inline], [inherited]`
Return a vector of all the edges in the model.

Returns

`vector of edges`

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.15.3.24 Export() [1/5]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
```

```
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.15.3.25 Export() [2/5]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
```

```
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.15.3.26 export() [1/4]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

Definition at line 138 of file [base.py](#).

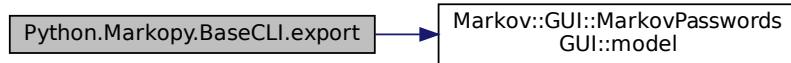
```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """
00143         self.model.Export(filename)
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.mo](#)

[Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.27 `export()` [2/4]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<code>filename</code>	filename to export to
-----------------------	-----------------------

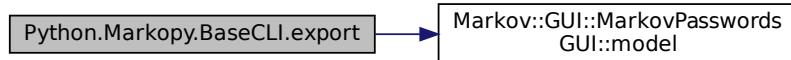
Definition at line 138 of file [base.py](#).

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142             """
00143         self.model.Export(filename)
00144 
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.28 export() [3/4]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

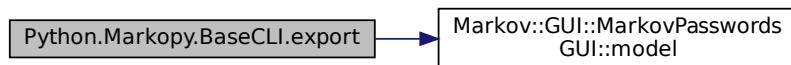
Definition at line 138 of file [base.py](#).

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142             """
00143         self.model.Export(filename)
00144
```

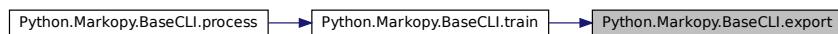
References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.29 export() [4/4]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

Definition at line 138 of file [base.py](#).

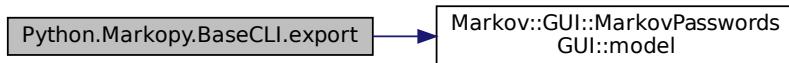
```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142             """
00143         self.model.Export(filename)
```

00144

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.30 Export() [3/5]

```
bool Markov::Model< char >::Export (
    std::ofstream * f )  [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file [model.h](#).

```
00288     {
00289         Markov::Edge<NodeStorageType>* e;
00290         for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291             e = this->edges[i];
00292             //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
00293             *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() << "\n";
00294         }
00295         return true;
00296     }
```

9.15.3.31 Export() [4/5]

```
bool Markov::Model< char >::Export (
    std::ofstream * f )  [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ostream

```
Markov::Model<char> model;
std::ostream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file model.h.

```
00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <
00294         "\n";
00295
00296     return true;
00297 }
```

9.15.3.32 Export() [5/5]

```
def Python.Markopy.MarkovModel.Export (
    str filename) [inherited]
```

Definition at line 26 of file mm.py.

```
00026     def Export(filename : str):
00027         pass
00028
```

9.15.3.33 FastRandomWalk() [1/3]

```
def Python.Markopy.ModelMatrix.FastRandomWalk (
    int count,
    str wordlist,
    int minlen,
    int maxlen) [inherited]
```

Definition at line 48 of file mm.py.

```
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

Referenced by [Python.Markopy.ModelMatrixCLI._generate\(\)](#).

Here is the caller graph for this function:

**9.15.3.34 FastRandomWalk() [2/3]**

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true) [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

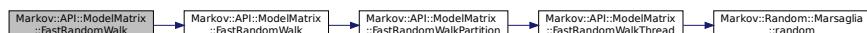
Definition at line 217 of file [modelMatrix.cpp](#).

```
00217
00218     std::ifstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

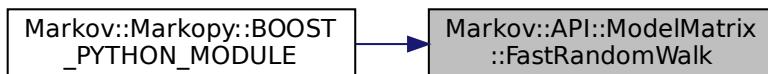
References [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.35 FastRandomWalk() [3/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [protected], [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import ("models/finished.mdl");
mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
```

Definition at line 204 of file [modelMatrix.cpp](#).

```
00204
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
00209     threads);
00210     else{
00211         int numberOfPartitions = n/50000000ull;
00212         for(int i=0;i<numberOfPartitions;i++)
00213             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
00214             threads);
00215 }
```

References [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.36 FastRandomWalkPartition()

```
void Markov::API::ModelMatrix::FastRandomWalkPartition (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    bool bFileIO,
    int threads ) [protected], [inherited]
```

A single partition of FastRandomWalk event.

Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation
- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 225 of file [modelMatrix.cpp](#).

```

00225
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++) {
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00235                                         mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00236         id++;
00237     }
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00239                                         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00240     for(int i=0;i<threads;i++) {
00241         threadsV[i]->join();
00242     }
00243 }
```

References [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.37 FastRandomWalkThread()

```
void Markov::API::ModelMatrix::FastRandomWalkThread (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    int id,
    bool bFileIO ) [protected], [inherited]
```

A single thread of a single partition of FastRandomWalk.

A FastRandomWalkPartition will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>id</i>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

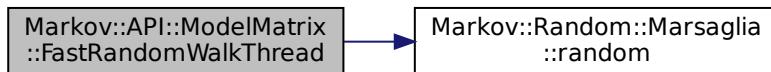
Definition at line 153 of file [modelMatrix.cpp](#).

```
00153
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179             if (len >= maxLen) break;
00180             else if ((next < 0) && (len < minLen)) continue;
00181             else if (next < 0) break;
00182             cur = next;
00183             res[bufferctr + len++] = cur;
00184         }
00185     }
00186     res[bufferctr + len++] = '\n';
00187     bufferctr+=len;
00188
00189 }
00190 if(bFileIO){
00191     *wordlist << res;
00192     mlock->unlock();
00193 }else{
00194     mlock->lock();
00195     std::cout << res;
00196     mlock->unlock();
00197 }
```

```
00199     delete res;
00200
00201 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::Random::Marsaglia::random\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#). Referenced by [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.38 Generate() [1/3]

```
def Python.Markopy.MarkovModel.Generate (
    int count,
    str wordlist,
    int minlen,
    int maxlen,
    int threads ) [inherited]
```

Definition at line 34 of file [mm.py](#).

```
00034     def Generate(count : int, wordlist : str, minlen : int, maxlen: int, threads : int):
00035         pass
00036
00037
```

Referenced by [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#).

Here is the caller graph for this function:



9.15.3.39 generate() [1/4]

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
--------------	----------------

Parameters

<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

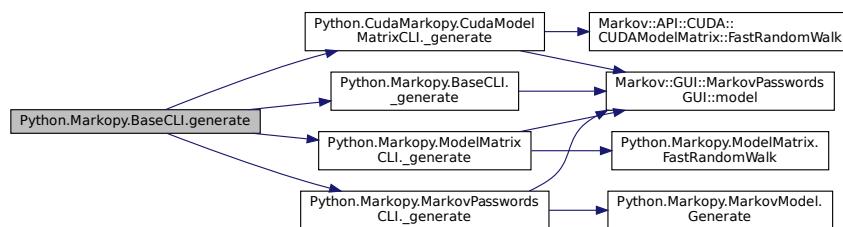
Definition at line 145 of file [base.py](#).

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159 
```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI._generate\(\)](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.40 generate() [2/4]

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
Generate strings from the model.
```

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159
```

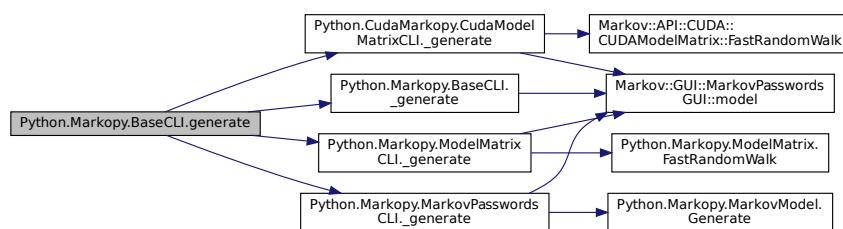
References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#),

[Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopy](#)

[Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.41 generate() [3/4]

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
```

```

00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151             """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155         return False
00156     if(bulk and os.path.isfile(wordlist)):
00157         logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158     self._generate(wordlist)
00159

```

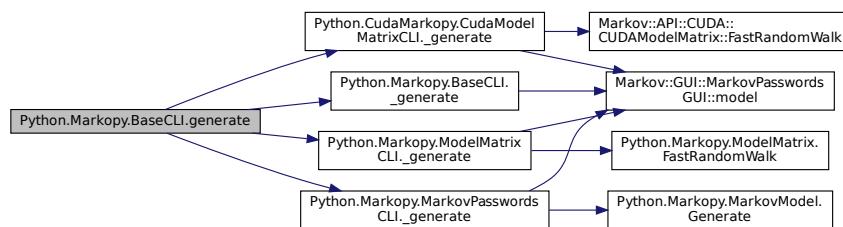
References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#),

[Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI._generate\(\)](#)

[Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.42 generate() [4/4]

```

def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]

```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

```

00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151             """
00152         if not (wordlist or self.args.count):

```

```

00153         logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156     if(bulk and os.path.isfile(wordlist)):
00157         logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158     self._generate(wordlist)
00159

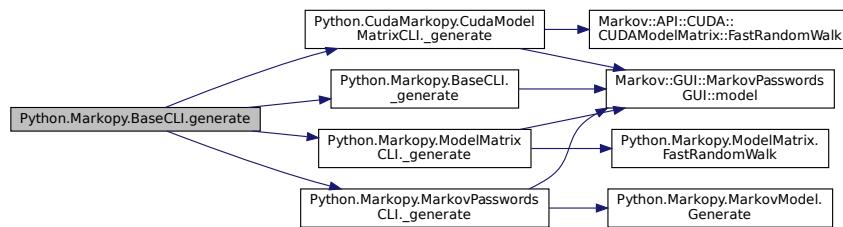
```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#),

[Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopy](#),
[Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.43 Generate() [2/3]

```

void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]

```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. [FastRandomWalk](#) will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

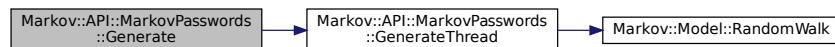
```

00118     {
00119         char* res;
00120         char print[100];
00121         std::ifstream wordlist;
00122         wordlist.open(wordlistFileName);
00123         std::mutex mlock;
00124         int iterationsPerThread = n/threads;
00125         int iterationsCarryOver = n%threads;
00126         std::vector<std::thread*> threadsV;
00127         for(int i=0;i<threads;i++) {
00128             threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129             &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130         }
00131         for(int i=0;i<threads;i++) {
00132             threadsV[i]->join();
00133             delete threadsV[i];
00134         }
00135     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137 }
00138 }
```

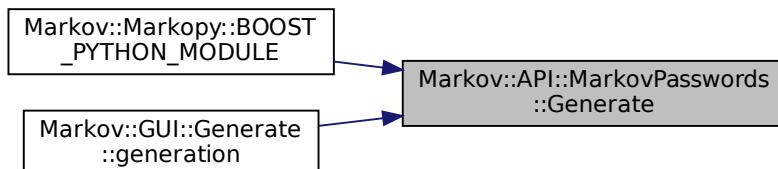
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.44 Generate() [3/3]

```

void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]
```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

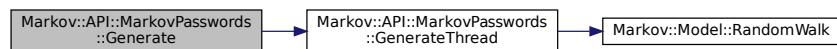
```

00118
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++) {
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129                                         &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130     }
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]~>join();
00133         delete threadsV[i];
00134     }
00135     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00136
00137
00138 }
```

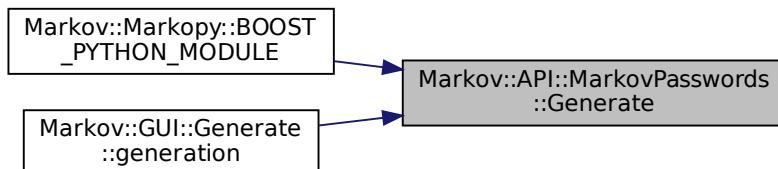
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.45 GenerateThread()

```

void Markov::API::MarkovPasswords::GenerateThread (
    std::mutex * outputLock,
    unsigned long int n,
    std::ofstream * wordlist,
```

```
int minLen,
int maxLen ) [private], [inherited]
```

A single thread invoked by the Generate function.

DEPRECATED: See `Markov::API::MatrixModel::FastRandomWalkThread` for more information. This has been replaced with a much more performance-optimized method. `FastRandomWalk` will reduce the runtime by %96.5 on average.

Parameters

<code>outputLock</code>	- shared mutex lock to lock during output operation. Prevents race condition on write.
<code>n</code>	number of lines to be generated by this thread
<code>wordlist</code>	wordlistfile
<code>minLen</code>	- Minimum password length to generate
<code>maxLen</code>	- Maximum password length to generate

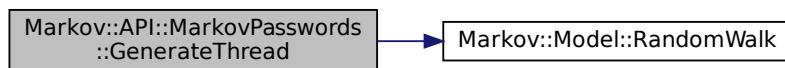
Definition at line 140 of file `markovPasswords.cpp`.

```
00140
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist << res << "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

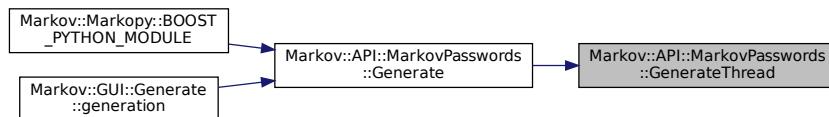
References `Markov::Model< NodeStorageType >::RandomWalk()`.

Referenced by `Markov::API::MarkovPasswords::Generate()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.46 help()

```
def Python.Markopy.MarkopyCLI.help (
    self )
overload help function to print submodel helps
Reimplemented from Python.Markopy.BaseCLI.
Reimplemented in Python.CudaMarkopy.CudaMarkopyCLI.
```

Definition at line 95 of file [markopy.py](#).

```

00095     def help(self):
00096         """
00097             @brief overload help function to print submodel helps
00098         """
00099         self.parser.print_help = self.stub
00100         self.args = self.parser.parse_known_args() [0]
00101         if(self.args.model_type!="_MMX"):
00102             if(self.args.model_type=="MP"):
00103                 mp = MarkovPasswordsCLI()
00104                 mp.add_arguments()
00105                 mp.parser.print_help()
00106             elif(self.args.model_type=="MMX"):
00107                 mp = ModelMatrixCLI()
00108                 mp.add_arguments()
00109                 mp.parser.print_help()
00110         else:
00111             print(colored("Model Mode selection choices:", "green"))
00112             self.print_help()
00113             print(colored("Following are applicable for -mt MP mode:", "green"))
00114             mp = MarkovPasswordsCLI()
00115             mp.add_arguments()
00116             mp.parser.print_help()
00117             print(colored("Following are applicable for -mt MMX mode:", "green"))
00118             mp = ModelMatrixCLI()
00119             mp.add_arguments()
00120             mp.parser.print_help()
00121
00122         exit()
00123
00124

```

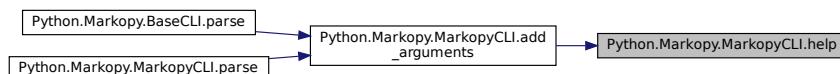
References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.parser](#), [Python.Markopy.BaseCLI.print_help](#), and [Python.Markopy.MarkopyCLI.stub\(\)](#).

Referenced by [Python.Markopy.MarkopyCLI.add_arguments\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.47 Import() [1/4]

```

void Markov::API::ModelMatrix::Import (
    const char * filename ) [inherited]

```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```

Markov::Model<char> model;
model.Import("test.mdl");

```

Construct the matrix when done.

Definition at line 19 of file [modelMatrix.cpp](#).

```

00019

```

```
{
```

```

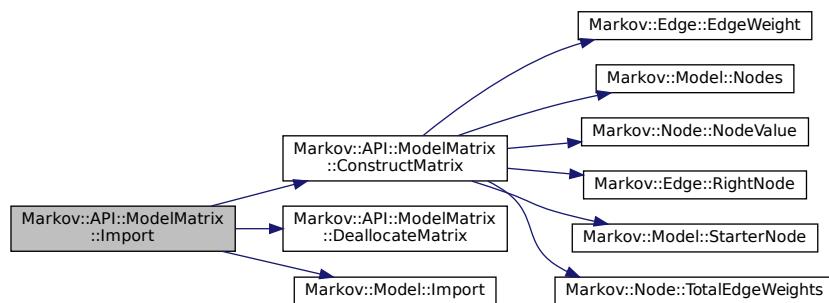
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }

```

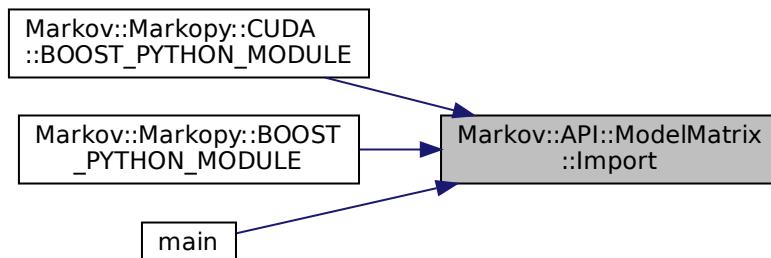
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.48 Import() [2/4]

```

bool Markov::Model< char >::Import (
    std::ifstream * f )  [inherited]

```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```

Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);

```

Definition at line [126](#) of file [model.h](#).

```

00216
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00245             //std::cout << "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00255         int(targetN->NodeValue()) << "\n";
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.15.3.49 Import() [3/4]

```
bool Markov::Model< char >::Import (
    std::ifstream * f ) [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file [model.h](#).

```

00216
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
```

```

00229     //std::cout << oc << "\n";
00230     Markov::Node<NodeStorageType>* srcN;
00231     Markov::Node<NodeStorageType>* targetN;
00232     Markov::Edge<NodeStorageType>* e;
00233     if (this->nodes.find(src) == this->nodes.end()) {
00234         srcN = new Markov::Node<NodeStorageType>(src);
00235         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00236         //std::cout << "Creating new node at start.\n";
00237     }
00238     else {
00239         srcN = this->nodes.find(src)->second;
00240     }
00241
00242     if (this->nodes.find(target) == this->nodes.end()) {
00243         targetN = new Markov::Node<NodeStorageType>(target);
00244         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00245         //std::cout << "Creating new node at end.\n";
00246     }
00247     else {
00248         targetN = this->nodes.find(target)->second;
00249     }
00250     e = srcN->Link(targetN);
00251     e->AdjustEdge(oc);
00252     this->edges.push_back(e);
00253
00254     //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <
00255     int(targetN->NodeValue()) << "\n";
00256
00257 }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.15.3.50 Import() [4/4]

```
def Python.Markopy.MarkovModel.Import (
    str filename ) [inherited]
Definition at line 22 of file mm.py.
00022     def Import(filename : str):
00023         pass
00024
```

9.15.3.51 import_model() [1/4]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
Import a model file.
```

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

Definition at line 77 of file base.py.

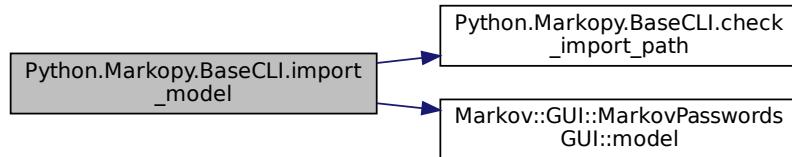
```

00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
00086             directory")
00087             return False
00088
00089         self.model.Import(filename)
00090         logging pprint("Model imported successfully.", 2)
00091
00092
00093
```

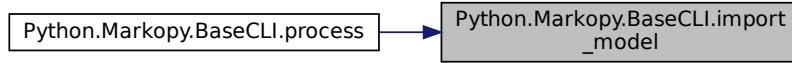
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.52 import_model() [2/4]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

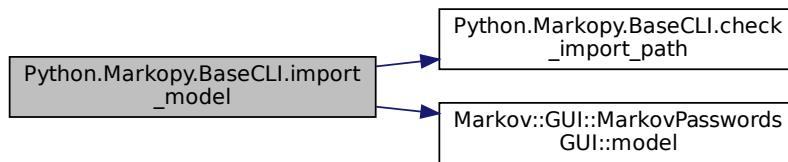
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

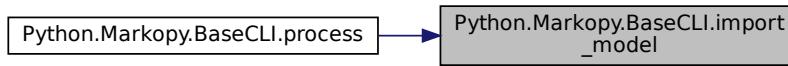
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.53 import_model() [3/4]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

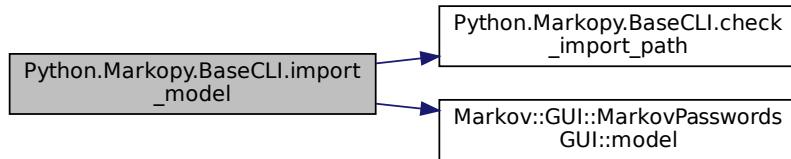
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

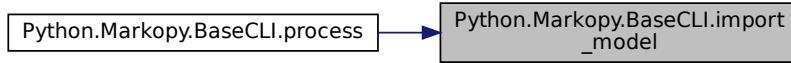
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.54 import_model() [4/4]

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

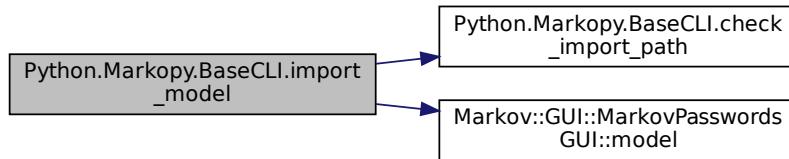
Definition at line 77 of file [base.py](#).

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
directory")
00086             return False
00087
00088         self.model.Import(filename)
00089         logging pprint("Model imported successfully.", 2)
00090         return True
00091
00092
00093
```

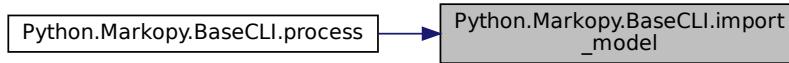
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model\(\)](#), [Python.Markopy.BaseCLI.model\(\)](#), [Python.Markopy.ModelMatrixCLI.model\(\)](#), [Python.Markopy.MarkovPasswordsCLI.model\(\)](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.55 init_post_arguments()

```
def Python.Markopy.MarkopyCLI.init_post_arguments (
    self )
```

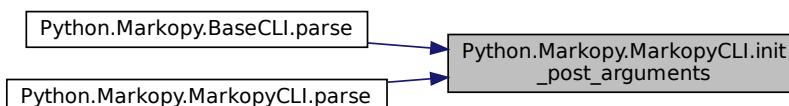
Reimplemented from [Python.Markopy.BaseCLI](#).

Definition at line [143](#) of file [markopy.py](#).

```
00143     def init_post_arguments(self):
00144         pass
00145
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.15.3.56 Nodes() [1/2]

```
std::map<char , Node<char >>* Markov::Model< char >::Nodes ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line [181](#) of file [model.h](#).

```
00181 { return &nodes; }
```

9.15.3.57 Nodes() [2/2]

```
std::map<char , Node<char >*>* Markov::Model< char >::Nodes ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.15.3.58 OpenDatasetFile() [1/2]

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename ) [inherited]
Open dataset file and return the ifstream pointer.
```

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

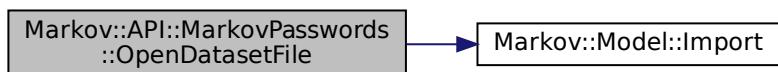
`ifstream*` to the the dataset file

Definition at line 51 of file [markovPasswords.cpp](#).

```
00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References [Markov::Model< NodeStorageType >::Import\(\)](#).

Here is the call graph for this function:



9.15.3.59 OpenDatasetFile() [2/2]

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename ) [inherited]
Open dataset file and return the ifstream pointer.
```

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

`ifstream*` to the dataset file

Definition at line 51 of file `markovPasswords.cpp`.

```
00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References `Markov::Model< NodeStorageType >::Import()`.

Here is the call graph for this function:

**9.15.3.60 OptimizeEdgeOrder() [1/2]**

`void Markov::Model< char >::OptimizeEdgeOrder [inherited]`

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file `model.h`.

```
00265
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType*>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
00269                     Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight() > rhs->EdgeWeight();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         //    std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00278 }
```

9.15.3.61 OptimizeEdgeOrder() [2/2]

`void Markov::Model< char >::OptimizeEdgeOrder [inherited]`

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file `model.h`.

```
00265
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType*>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
00269                     Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight() > rhs->EdgeWeight();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         //    std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00278 }
```

9.15.3.62 parse()

```
def Python.Markopy.MarkopyCLI.parse (
    self)
```

Reimplemented from [Python.Markopy.BaseCLI](#).

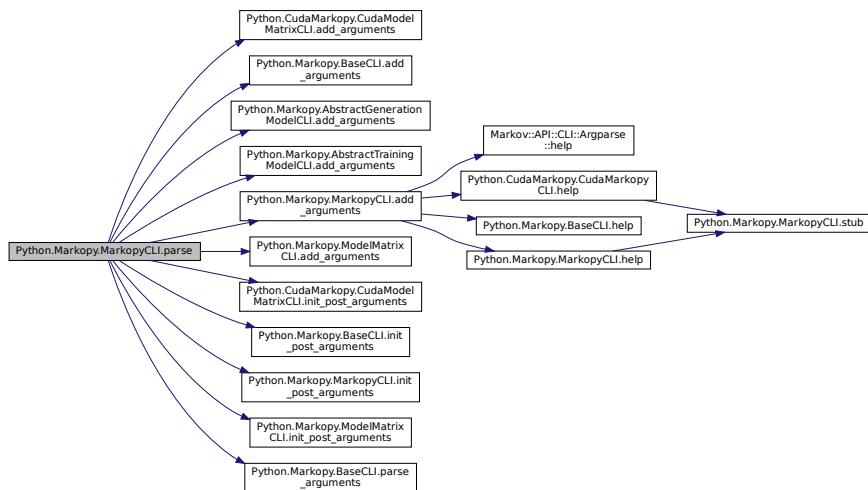
Reimplemented in [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 126 of file [markopy.py](#).

```
00126     def parse(self):
00127         """! @brief overload parse function to parse for submodels"""
00128
00129         self.add_arguments()
00130         self.parse_arguments()
00131         self.init_post_arguments()
00132         if(self.args.model_type == "MP"):
00133             self.cli = MarkovPasswordsCLI()
00134         elif(self.args.model_type == "MMX" or self.args.model_type == "_MMX"):
00135             self.cli = ModelMatrixCLI()
00136         else:
00137             self.parse_fail()
00138
00139         if(self.args.help): return self.help()
00140         self.cli.parse()
00141
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.init_post_arguments\(\)](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.CudaMarkopy.CudaModelMatrixCLI.init_post_arguments\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.init_post_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.init_post_arguments\(\)](#) and [Python.Markopy.BaseCLI.parse_arguments\(\)](#).

Here is the call graph for this function:



9.15.3.63 parse_arguments() [1/4]

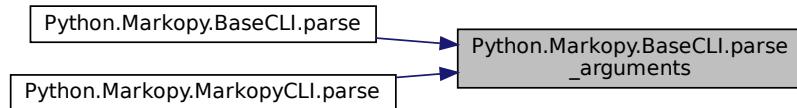
```
def Python.Markopy.BaseCLI.parse_arguments (
    self) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.15.3.64 parse_arguments() [2/4]

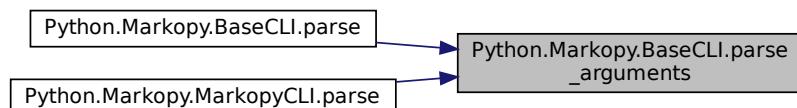
```
def Python.Markopy.BaseCLI.parse_arguments (
    self ) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.15.3.65 parse_arguments() [3/4]

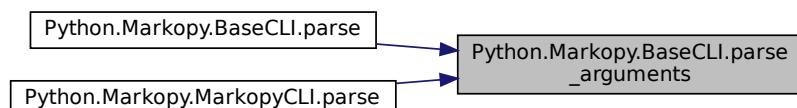
```
def Python.Markopy.BaseCLI.parse_arguments (
    self ) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args() [0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.15.3.66 parse_arguments() [4/4]

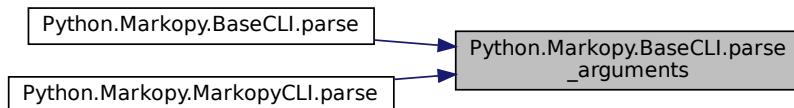
```
def Python.Markopy.BaseCLI.parse_arguments (
    self ) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.15.3.67 parse_fail()

```
def Python.Markopy.MarkopyCLI.parse_fail (
    self )
```

Reimplemented in [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 147 of file [markopy.py](#).

```
00147     def parse_fail(self):
00148         """! @brief failed to parse model type"""
00149         print("Unrecognized model type.")
00150         exit()
00151
```

9.15.3.68 process()

```
def Python.Markopy.MarkopyCLI.process (
    self )
```

Process parameters for operation.

Reimplemented from [Python.Markopy.BaseCLI](#).

Definition at line 152 of file [markopy.py](#).

```
00152     def process(self):
00153         """! @brief pass the process request to selected submodel"""
00154         return self.cli.process()
00155
```

References [Python.CudaMarkopy.CudaMarkopyCLI.cli](#), and [Python.Markopy.MarkopyCLI.cli](#).

9.15.3.69 RandomWalk() [1/2]

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```
00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319         else if (temp_node == NULL) {
00320             break;
00321         }
00322         n = temp_node;
00323         n->RandomNext(randomEngine);
00324         buffer[len++] = n->NodeValue();
00325     }
00326     //null terminate the string
00327     buffer[len] = 0x00;
00328
00329     //do something with the generated string
00330     return buffer; //for now
00331
00332
00333 }
```

9.15.3.70 RandomWalk() [2/2]

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from. This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```
00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319         else if (temp_node == NULL) {
00320             break;
00321         }
00322         n = temp_node;
00323
00324         buffer[len++] = n->NodeValue();
00325     }
00326
00327     //null terminate the string
00328     buffer[len] = 0x00;
00329
00330     //do something with the generated string
00331     return buffer; //for now
00332
00333 }
```

9.15.3.71 Save() [1/2]

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
```

Export model to file.

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

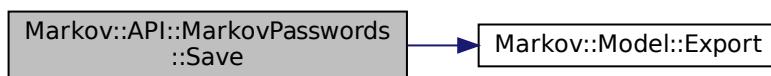
`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

```
00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:

**9.15.3.72 Save() [2/2]**

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
```

Export model to file.

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

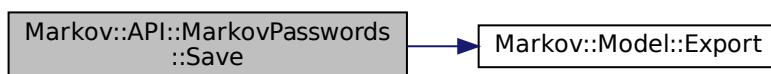
`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

```
00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.15.3.73 StarterNode() [1/2]

```
Node< char >* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.15.3.74 StarterNode() [2/2]

```
Node< char >* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.15.3.75 stub()

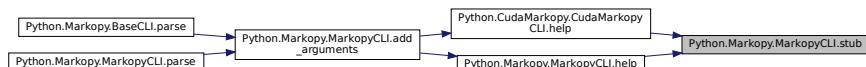
```
def Python.Markopy.MarkopyCLI.stub (
    self )
```

Definition at line 156 of file [markopy.py](#).

```
00156     def stub(self):
00157         """! @brief stub function to hack help requests"""
00158         return
00159
00160
```

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

Here is the caller graph for this function:



9.15.3.76 Train() [1/2]

```
void Markov::API::ModelMatrix::Train (
    const char * datasetFileName,
    char delimiter,
    int threads ) [inherited]
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

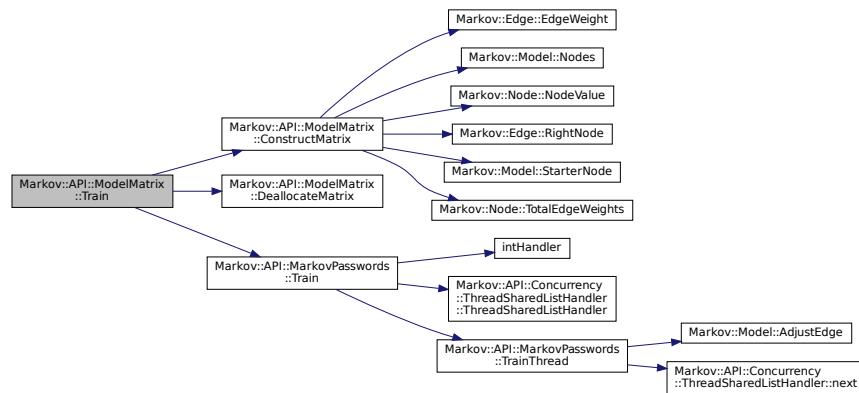
Construct the matrix when done.

Definition at line 25 of file [modelMatrix.cpp](#).

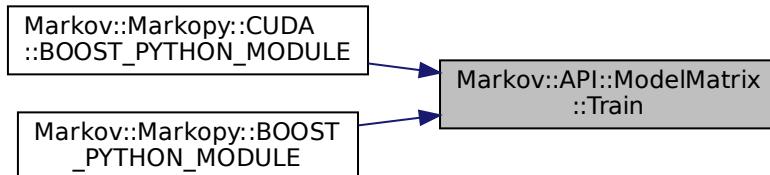
```
00025     this->DeallocateMatrix();
00026     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00027     this->ConstructMatrix();
00028     this->ConstructMatrix();
00029 }
```

References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.77 train() [1/4]

```
def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False ) [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset

Parameters

<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

```

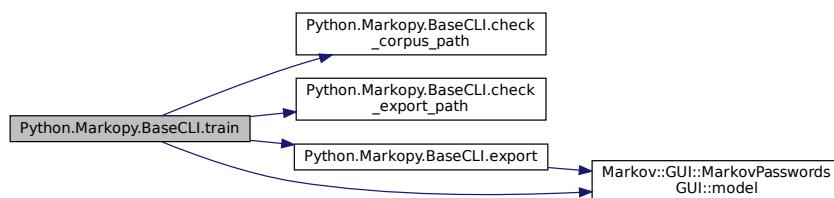
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096             """
00097                 @brief Train a model via CLI parameters
00098                 @param model Model instance
00099                 @param dataset filename for the dataset
00100                 @param seperator seperator used with the dataset
00101                 @param output output filename
00102                 @param output_forced force overwrite
00103                 @param bulk marks bulk operation with directories
00104             """
00105             logging pprint("Training.")
00106             if not (dataset and seperator and (output or not output_forced)):
00107                 logging pprint(f"Training mode requires -d/--dataset(',
00108 else") and -s/--seperator parameters. Exiting.")
00109             return False
00110             if not bulk and not self.check_corpus_path(dataset):
00111                 logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112                 return False
00113             if not self.check_export_path(output):
00114                 logging pprint(f"Cannot create output at {output}")
00115                 return False
00116             if(seperator == '\\t'):
00117                 logging pprint("Escaping seperator.", 3)
00118                 seperator = '\t'
00119             if(len(seperator)!=1):
00120                 logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00121 accepted.')
00122                 exit(4)
00123             logging pprint(f'Starting training.', 3)
00124             self.model.Train(dataset,seperator, int(self.args.threads))
00125             logging pprint(f'Training completed.', 2)
00126             if(output):
00127                 logging pprint(f'Exporting model to {output}', 2)
00128                 self.export(output)
00129             else:
00130                 logging pprint(f'Model will not be exported.', 1)
00131             return True
00132
00133
00134
00135
00136
00137

```

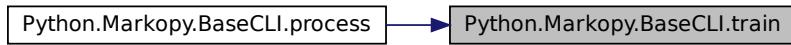
References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.78 train() [2/4]

```
def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False ) [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104
00105             logging pprint("Training.")
00106
00107             if not (dataset and seperator and (output or not output_forced)):
00108                 logging pprint(f"Training mode requires -d/--dataset', -o/--output' if output_forced
00109                 else") and -s/-seperator parameters. Exiting.")
00110             return False
00111
00112             if not bulk and not self.check_corpus_path(dataset):
00113                 logging pprint(f"\{dataset}\ doesn't exists. Check the file path, or working directory")
00114             return False
00115
00116             if not self.check_export_path(output):
00117                 logging pprint(f"Cannot create output at {output}")
00118             return False
00119
00120             if(seperator == '\\t'):
00121                 logging pprint("Escaping seperator.", 3)
00122                 seperator = '\t'
00123
00124             if(len(seperator)!=1):
00125                 logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00126 accepted.')
00127             exit(4)
00128
00129             logging pprint(f'Starting training.', 3)
00130             self.model.Train(dataset, seperator, int(self.args.threads))
```

```

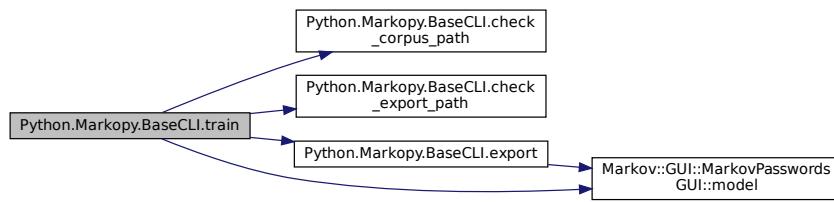
00128     logging pprint(f'Training completed.', 2)
00129
00130     if(output):
00131         logging pprint(f'Exporting model to {output}', 2)
00132         self.export(output)
00133     else:
00134         logging pprint(f'Model will not be exported.', 1)
00135
00136     return True
00137

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.79 train() [3/4]

```

def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False ) [inherited]

```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

```

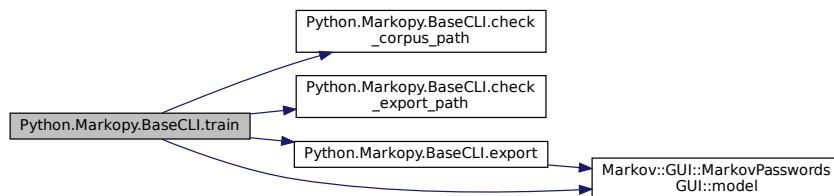
00094     def train(self, dataset : str, separator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096             """
00097                 @brief Train a model via CLI parameters
00098                 @param model Model instance
00099                 @param dataset filename for the dataset
00100                 @param separator separator used with the dataset
00101                 @param output output filename
00102                 @param output_forced force overwrite
00103                 @param bulk marks bulk operation with directories
00104             """
00105             logging pprint("Training.")
00106             if not (dataset and separator and (output or not output_forced)):
00107                 logging pprint(f"Training mode requires -d/--dataset(', -o/--output' if output_forced
00108             else") and -s/-separator parameters. Exiting.")
00109             return False
00110             if not bulk and not self.check_corpus_path(dataset):
00111                 logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00112                 return False
00113             if not self.check_export_path(output):
00114                 logging pprint(f"Cannot create output at {output}")
00115                 return False
00116             if(separator == '\\t'):
00117                 logging pprint("Escaping separator.", 3)
00118                 separator = '\t'
00119             if(len(separator)!=1):
00120                 logging pprint(f'Delimiter must be a single character, and "{separator}" is not
00121             accepted.')
00122                 exit(4)
00123             if(output):
00124                 logging pprint(f'Exporting model to {output}', 2)
00125                 self.export(output)
00126             else:
00127                 logging pprint(f'Model will not be exported.', 1)
00128             return True
00129
00130
00131
00132
00133
00134
00135
00136
00137

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.80 train() [4/4]

```
def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False ) [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

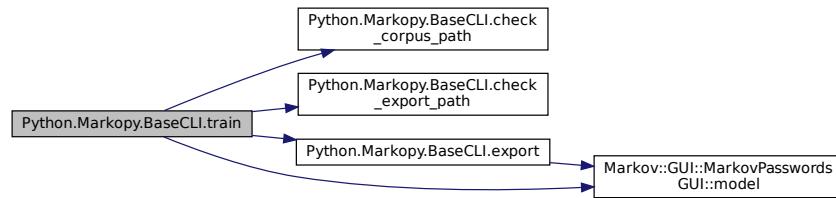
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104         """
00105         logging pprint("Training.")
00106
00107         if not (dataset and seperator and (output or not output_forced)):
00108             logging pprint(f"Training mode requires -d/--dataset', -o/--output' if output_forced
00109             else") and -s/--seperator parameters. Exiting.")
00110             return False
00111
00112         if not bulk and not self.check_corpus_path(dataset):
00113             logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00114             return False
00115
00116         if not self.check_export_path(output):
00117             logging pprint(f"Cannot create output at {output}")
00118             return False
00119
00120         if(seperator == '\\t'):
00121             logging pprint("Escaping seperator.", 3)
00122             seperator = '\t'
00123
00124         if(len(seperator)!=1):
00125             logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00126             accepted.')
00127             exit(4)
00128
00129         if(output):
00130             logging pprint(f'Starting training.', 3)
00131             self.model.Train(dataset, seperator, int(self.args.threads))
00132             logging pprint(f'Training completed.', 2)
00133
00134         if(output):
00135             logging pprint(f'Exporting model to {output}', 2)
00136             self.export(output)
00137         else:
00138             logging pprint(f'Model will not be exported.', 1)
00139
00140     return True
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#),

[Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.3.81 Train() [2/2]

```

def Python.Markopy.MarkovModel.Train (
    str dataset,
    str seperator,
    int threads ) [inherited]
Definition at line 30 of file mm.py.
00030     def Train(dataset: str, seperator : str, threads : int):
00031         pass
00032

```

9.15.3.82 TrainThread()

```

void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private], [inherited]

```

A single thread invoked by the Train function.

Parameters

<code>listhandler</code>	- Listhandler class to read corpus from
<code>delimiter</code>	- a character, same as the delimiter in dataset content

Definition at line 85 of file [markovPasswords.cpp](#).

```

00085
00086     char format_str[] = "%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];

```

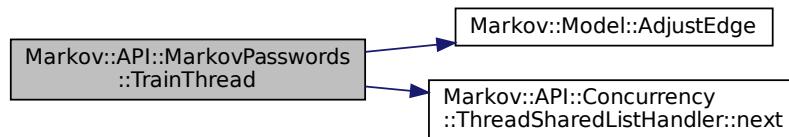
```

00095 #ifdef _WIN32
00096     sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length() + 5); //<== changed format_str to->
00097     "%ld,%s"
00098     sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100     this->AdjustEdge((const char*)linebuf, oc);
00101     delete linebuf;
00102 }
00103 }
```

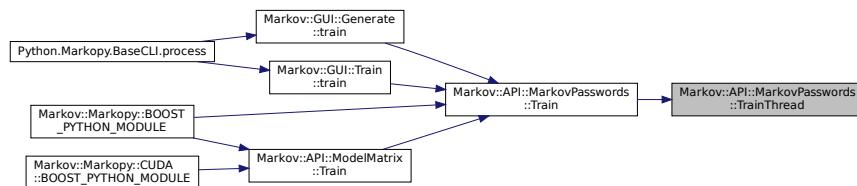
References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), [keepRunning](#), and [Markov::API::Concurrency::ThreadSharedListHandler::next](#).

Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.15.4 Member Data Documentation

9.15.4.1 args

`Python.Markopy.MarkopyCLI.args`

Definition at line 75 of file [markopy.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.MarkopyCLI.help\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.parse\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.parse_fail\(\)](#), [Python.Markopy.BaseCLI.process\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.15.4.2 cli

`Python.Markopy.MarkopyCLI.cli`

Definition at line 133 of file [markopy.py](#).

Referenced by [Python.Markopy.MarkopyCLI.process\(\)](#).

9.15.4.3 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]`

Definition at line 123 of file [markovPasswords.h](#).

9.15.4.4 edgeMatrix

```
char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]  
2-D Character array for the edge Matrix (The characters of Nodes)  
Definition at line 175 of file modelMatrix.h.  
Referenced by Markov::API::ModelMatrix::ConstructMatrix\(\), Markov::API::ModelMatrix::DeallocateMatrix\(\),  
Markov::API::ModelMatrix::DumpJSON\(\), and Markov::API::ModelMatrix::FastRandomWalkThread\(\).
```

9.15.4.5 edges

```
std::vector<Edge<char *>> Markov::Model< char >::edges [private], [inherited]  
A list of all edges in this model.  
Definition at line 204 of file model.h.
```

9.15.4.6 fileIO

```
Python.Markopy.ModelMatrixCLI.fileIO [inherited]  
Definition at line 38 of file mmx.py.  
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI.\_generate\(\), and Python.Markopy.ModelMatrixCLI.\_generate\(\).
```

9.15.4.7 matrixIndex

```
char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]  
to hold the Matrix index (To hold the orders of 2-D arrays)  
Definition at line 190 of file modelMatrix.h.  
Referenced by Markov::API::ModelMatrix::ConstructMatrix\(\), Markov::API::ModelMatrix::DeallocateMatrix\(\),  
Markov::API::ModelMatrix::DumpJSON\(\), and Markov::API::ModelMatrix::FastRandomWalkThread\(\).
```

9.15.4.8 matrixSize

```
int Markov::API::ModelMatrix::matrixSize [protected], [inherited]  
to hold Matrix size  
Definition at line 185 of file modelMatrix.h.  
Referenced by Markov::API::ModelMatrix::ConstructMatrix\(\), Markov::API::ModelMatrix::DeallocateMatrix\(\),  
Markov::API::ModelMatrix::DumpJSON\(\), Markov::API::ModelMatrix::FastRandomWalkThread\(\), and Markov::API::CUDA::CUDAMod.
```

9.15.4.9 model [1/3]

```
Python.Markopy.BaseCLI.model [inherited]  
Definition at line 40 of file base.py.  
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI.\_generate\(\), Python.Markopy.BaseCLI.\_generate\(\),  
Python.Markopy.ModelMatrixCLI.\_generate\(\), Python.Markopy.MarkovPasswordsCLI.\_generate\(\), Python.Markopy.BaseCLI.export\(\),  
Python.Markopy.BaseCLI.import\_model\(\), and Python.Markopy.BaseCLI.train\(\).
```

9.15.4.10 model [2/3]

```
Python.Markopy.ModelMatrixCLI.model [inherited]  
Definition at line 30 of file mmx.py.  
Referenced by Python.CudaMarkopy.CudaModelMatrixCLI.\_generate\(\), Python.Markopy.BaseCLI.\_generate\(\),  
Python.Markopy.ModelMatrixCLI.\_generate\(\), Python.Markopy.MarkovPasswordsCLI.\_generate\(\), Python.Markopy.BaseCLI.export\(\),  
Python.Markopy.BaseCLI.import\_model\(\), and Python.Markopy.BaseCLI.train\(\).
```

9.15.4.11 model [3/3]

`Python.Markopy.MarkovPasswordsCLI.model` [inherited]

Definition at line 29 of file [mp.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.15.4.12 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile` [private], [inherited]

Dataset file input of our system

Definition at line 124 of file [markovPasswords.h](#).

9.15.4.13 nodes

`std::map<char , Node<char >> Markov::Model< char >::nodes` [private], [inherited]

Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.

Definition at line 193 of file [model.h](#).

9.15.4.14 outputFile

`std::ofstream* Markov::API::MarkovPasswords::outputFile` [private], [inherited]

File to save model of our system

Definition at line 125 of file [markovPasswords.h](#).

9.15.4.15 parser [1/4]

`Python.Markopy.BaseCLI.parser` [inherited]

Definition at line 25 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.15.4.16 parser [2/4]

`Python.Markopy.BaseCLI.parser` [inherited]

Definition at line 25 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.15.4.17 parser [3/4]

`Python.Markopy.BaseCLI.parser` [inherited]

Definition at line 25 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.15.4.18 parser [4/4]

Python.Markopy.BaseCLI.parser [inherited]

Definition at line 25 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.15.4.19 print_help [1/4]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.15.4.20 print_help [2/4]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.15.4.21 print_help [3/4]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.15.4.22 print_help [4/4]

Python.Markopy.BaseCLI.print_help [inherited]

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.15.4.23 ready

bool Markov::API::ModelMatrix::ready [protected], [inherited]

True when matrix is constructed. False if not.

Definition at line 200 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::ModelMatrix\(\)](#).

9.15.4.24 starterNode

Node<char *>* Markov::Model< char >::starterNode [private], [inherited]

Starter Node of this model.

Definition at line 198 of file [model.h](#).

9.15.4.25 totalEdgeWeights

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]

Array of the Total Edge Weights.

Definition at line 195 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.15.4.26 valueMatrix

long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]

2-d Integer array for the value Matrix (For the weights of Edges)

Definition at line 180 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

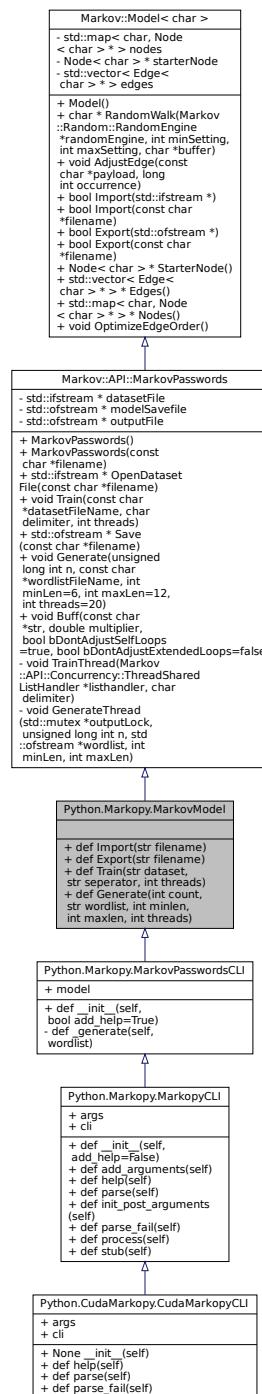
The documentation for this class was generated from the following file:

- [Markopy/Markopy/src/CLI/markopy.py](#)

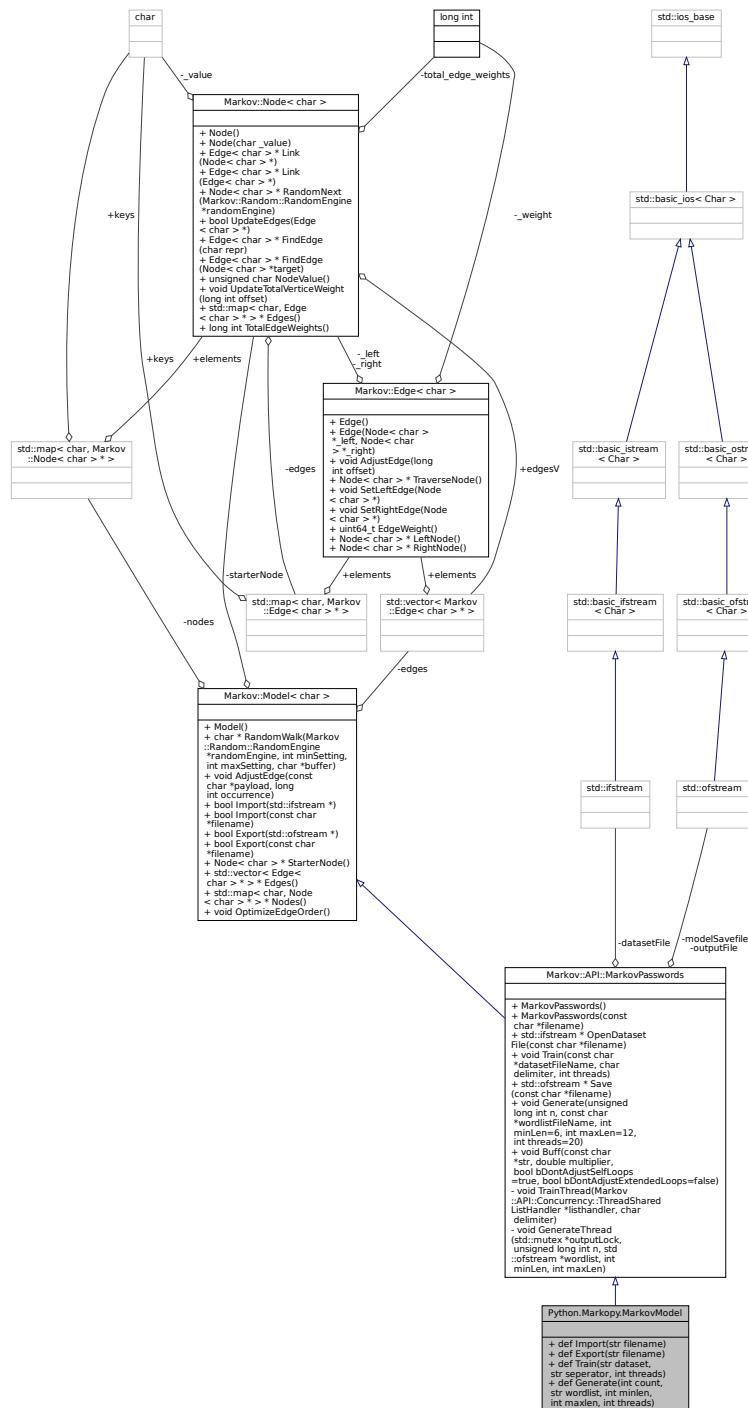
9.16 Python.Markopy.MarkovModel Class Reference

Abstract representation of a markov model.

Inheritance diagram for Python.Markopy.MarkovModel:



Collaboration diagram for Python.Markopy.MarkovModel:



Public Member Functions

- def **Import** (str filename)
 - def **Export** (str filename)
 - def **Train** (str dataset, str seperator, int threads)
 - def **Generate** (int count, str wordlist, int minlen, int maxlen, int threads)
 - std::ifstream * **OpenDatasetFile** (const char *filename)

Open dataset file and return the ifstream pointer.

- void **Train** (const char *datasetFileName, char delimiter, int threads)

Train the model with the dataset file.
- std::ofstream * **Save** (const char *filename)

Export model to file.
- void **Generate** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

Call Markov::Model::RandomWalk n times, and collect output.
- void **Buff** (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)

Buff expression of some characters in the model.
- char * **RandomWalk** (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

Do a random walk on this model.
- void **AdjustEdge** (const char *payload, long int occurrence)

Adjust the model with a single string.
- bool **Import** (std::ifstream *)

Import a file to construct the model.
- bool **Import** (const char *filename)

Open a file to import with filename, and call bool Model::Import with std::ifstream.
- bool **Export** (std::ofstream *)

Export a file of the model.
- bool **Export** (const char *filename)

Open a file to export with filename, and call bool Model::Export with std::ofstream.
- Node< char > * **StarterNode** ()

Return starter Node.
- std::vector< Edge< char > * > * **Edges** ()

Return a vector of all the edges in the model.
- std::map< char, Node< char > * > * **Nodes** ()

Return starter Node.
- void **OptimizeEdgeOrder** ()

Sort edges of all nodes in the model ordered by edge weights.

Private Member Functions

- void **TrainThread** (Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char delimiter)

A single thread invoked by the Train function.
- void **GenerateThread** (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)

A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * **datasetFile**

Dataset file input of our system
- std::ofstream * **outputFile**

File to save model of our system
- std::map< char, Node< char > * > **nodes**

Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > * **starterNode**

Starter Node of this model.
- std::vector< Edge< char > * > **edges**

A list of all edges in this model.

9.16.1 Detailed Description

Abstract representation of a markov model.
To help with the python-cpp gateway documentation.
Definition at line 13 of file [mm.py](#).

9.16.2 Member Function Documentation

9.16.2.1 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.16.2.2 Buff()

```
void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.

Parameters

<code>bDontAdjustSelfEdges</code>	Do not adjust weights if target node is same as source node
<code>bDontAdjustExtendedLoops</code>	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

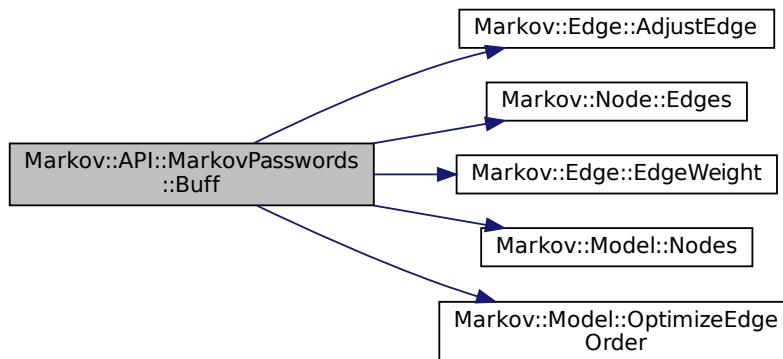
```

00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173         }
00174         i++;
00175     }
00176 }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

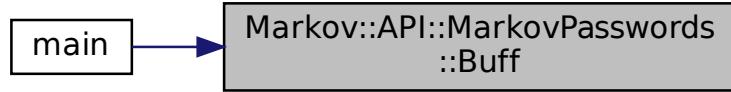
References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.16.2.3 Edges()

`std::vector<Edge<char>*>* Markov::Model< char >::Edges () [inherited]`
Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.16.2.4 Export() [1/3]

`bool Markov::Model< char >::Export (`
 `const char * filename) [inherited]`

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.16.2.5 Export() [2/3]

`bool Markov::Model< char >::Export (`
 `std::ofstream * f) [inherited]`

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ostream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file model.h.

```
00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <
00294         "\n";
00295     }
00296     return true;
00297 }
```

9.16.2.6 Export() [3/3]

```
def Python.Markopy.MarkovModel.Export (
    str filename )
```

Definition at line 26 of file mm.py.

```
00026     def Export(filename : str):
00027         pass
00028
```

9.16.2.7 Generate() [1/2]

```
def Python.Markopy.MarkovModel.Generate (
    int count,
    str wordlist,
    int minlen,
    int maxlen,
    int threads )
```

Definition at line 34 of file mm.py.

```
00034     def Generate(count : int, wordlist : str, minlen : int, maxlen: int, threads : int):
00035         pass
00036
00037
```

Referenced by [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#).

Here is the caller graph for this function:

**9.16.2.8 Generate() [2/2]**

```
void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]
```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

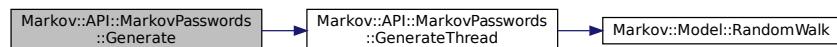
Definition at line 118 of file [markovPasswords.cpp](#).

```
00118     {
00119         char* res;
00120         char print[100];
00121         std::ofstream wordlist;
00122         wordlist.open(wordlistFileName);
00123         std::mutex mlock;
00124         int iterationsPerThread = n/threads;
00125         int iterationsCarryOver = n%threads;
00126         std::vector<std::thread*> threadsV;
00127         for(int i=0;i<threads;i++) {
00128             threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129             &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130         }
00131         for(int i=0;i<threads;i++) {
00132             threadsV[i]->join();
00133             delete threadsV[i];
00134         }
00135         this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00136     }
00137 }
```

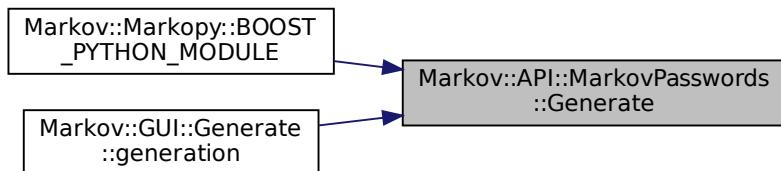
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.16.2.9 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
    std::mutex * outputLock,
```

```

    unsigned long int n,
    std::ofstream * wordlist,
    int minLen,
    int maxLen ) [private], [inherited]

```

A single thread invoked by the Generate function.

DEPRECATED: See `Markov::API::MatrixModel::FastRandomWalkThread` for more information. This has been replaced with a much more performance-optimized method. `FastRandomWalk` will reduce the runtime by %96.5 on average.

Parameters

<code>outputLock</code>	- shared mutex lock to lock during output operation. Prevents race condition on write.
<code>n</code>	number of lines to be generated by this thread
<code>wordlist</code>	wordlistfile
<code>minLen</code>	- Minimum password length to generate
<code>maxLen</code>	- Maximum password length to generate

Definition at line 140 of file `markovPasswords.cpp`.

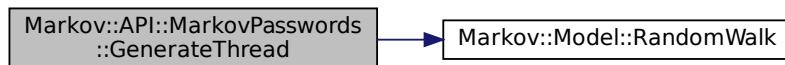
```

00140
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist << res << "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

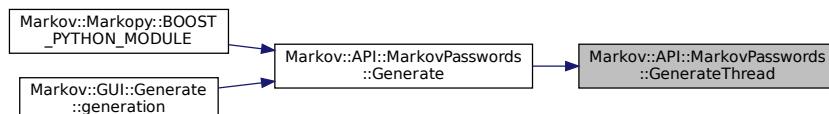
References `Markov::Model< NodeStorageType >::RandomWalk()`.

Referenced by `Markov::API::MarkovPasswords::Generate()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.16.2.10 Import() [1/3]

```

bool Markov::Model< char >::Import (
    const char * filename ) [inherited]

```

Open a file to import with `filename`, and call `bool Model::Import` with `std::ifstream`.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Definition at line 137 of file model.h.

```
00280
00281     std::ifstream importfile;
00282     importfile.open(filename);
00283     return this->Import(&importfile);
00284
00285 }
```

9.16.2.11 Import() [2/3]

```
bool Markov::Model< char >::Import (
    std::ifstream * f )  [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file model.h.

```
00216
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00245             //std::cout << "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00255         int(targetN->NodeValue()) << "\n";
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260 }
```

```
00261     return true;
00262 }
```

9.16.2.12 Import() [3/3]

```
def Python.Markopy.MarkovModel.Import (
    str filename )
Definition at line 22 of file mm.py.
00022     def Import(filename : str):
00023         pass
00024
```

9.16.2.13 Nodes()

`std::map<char , Node<char *>>* Markov::Model< char >::Nodes ()` [inline], [inherited]
Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file model.h.

```
00181 { return &nodes; }
```

9.16.2.14 OpenDatasetFile()

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename )
```

[inherited]
Open dataset file and return the ifstream pointer.

Parameters

<code>filename</code>	- Filename to open
-----------------------	--------------------

Returns

`ifstream*` to the the dataset file

Definition at line 51 of file markovPasswords.cpp.

```
00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References `Markov::Model< NodeStorageType >::Import()`.

Here is the call graph for this function:



9.16.2.15 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder [inherited]
Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.
00265
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
00269             Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight () > rhs->EdgeWeight ();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         // std::cout << x.second->edgesV[i]->EdgeWeight () << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00277 }
```

9.16.2.16 RandomWalk()

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
```

```

00312     temp_node = n->RandomNext (randomEngine);
00313     if (len >= maxSetting) {
00314         break;
00315     }
00316     else if ((temp_node == NULL) && (len < minSetting)) {
00317         continue;
00318     }
00319
00320     else if (temp_node == NULL) {
00321         break;
00322     }
00323
00324     n = temp_node;
00325
00326     buffer[len++] = n->NodeValue ();
00327 }
00328
00329 //null terminate the string
00330 buffer[len] = 0x00;
00331
00332 //do something with the generated string
00333 return buffer; //for now
00334 }
```

9.16.2.17 Save()

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
Export model to file.
```

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

```

00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.16.2.18 StarterNode()

```
Node< char *>* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.16.2.19 Train() [1/2]

```
void Markov::API::MarkovPasswords::Train (
    const char * datasetFileName,
    char delimiter,
    int threads ) [inherited]
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

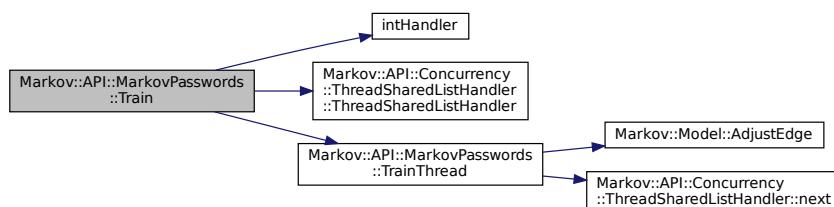
Definition at line 65 of file [markovPasswords.cpp](#).

```
00065 {
00066     signal(SIGINT, intHandler);
00067     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00068     auto start = std::chrono::high_resolution_clock::now();
00069
00070     std::vector<std::thread*> threadsV;
00071     for(int i=0;i<threads;i++) {
00072         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
00073             &listhandler, delimiter));
00074     }
00075     for(int i=0;i<threads;i++) {
00076         threadsV[i]->join();
00077         delete threadsV[i];
00078     }
00079     auto finish = std::chrono::high_resolution_clock::now();
00080     std::chrono::duration<double> elapsed = finish - start;
00081     std::cout << "Elapsed time: " << elapsed.count() << " s\n";
00082
00083 }
```

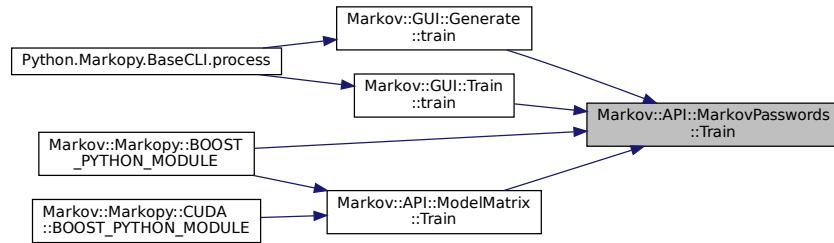
References [intHandler\(\)](#), [Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler\(\)](#), and [Markov::API::MarkovPasswords::TrainThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), [Markov::GUI::Generate::train\(\)](#), [Markov::GUI::Train::train\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.16.2.20 Train() [2/2]

```

def Python.Markopy.MarkovModel.Train (
    str dataset,
    str seperator,
    int threads )
Definition at line 30 of file mm.py.
00030     def Train(dataset: str, seperator : str, threads : int):
00031         pass
00032
  
```

9.16.2.21 TrainThread()

```

void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private], [inherited]
  
```

A single thread invoked by the Train function.

Parameters

<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

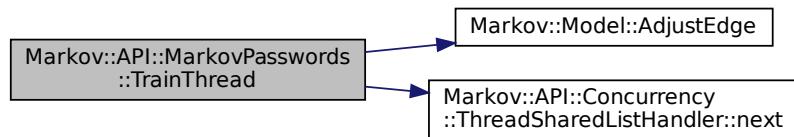
Definition at line 85 of file markovPasswords.cpp.

```

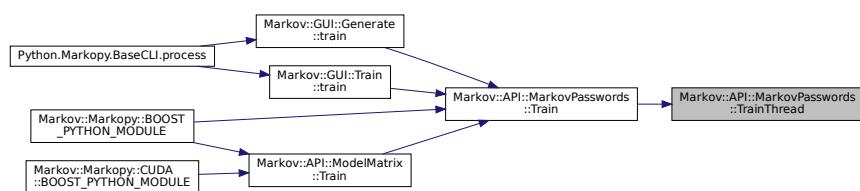
00085
00086     char format_str[] = "%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
00097         "%ld,%s"
00098     else
00099         sscanf(line.c_str(), format_str, &oc, linebuf);
00100     #endif
00101     this->AdjustEdge((const char*)linebuf, oc);
00102     delete linebuf;
00103 }
  
```

References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), [keepRunning](#), and [Markov::API::Concurrency::ThreadSharedListHandler](#). Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.16.3 Member Data Documentation

9.16.3.1 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]`
 Definition at line 123 of file [markovPasswords.h](#).

9.16.3.2 edges

`std::vector<Edge<char *>> Markov::Model< char >::edges [private], [inherited]`
 A list of all edges in this model.
 Definition at line 204 of file [model.h](#).

9.16.3.3 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`
 Dataset file input of our system

Definition at line 124 of file [markovPasswords.h](#).

9.16.3.4 nodes

`std::map<char , Node<char *>> Markov::Model< char >::nodes [private], [inherited]`
 Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
 Definition at line 193 of file [model.h](#).

9.16.3.5 outputFile

```
std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]  
File to save model of our system
```

Definition at line 125 of file [markovPasswords.h](#).

9.16.3.6 starterNode

```
Node<char>* Markov::Model< char >::starterNode [private], [inherited]  
Starter Node of this model.
```

Definition at line 198 of file [model.h](#).

The documentation for this class was generated from the following file:

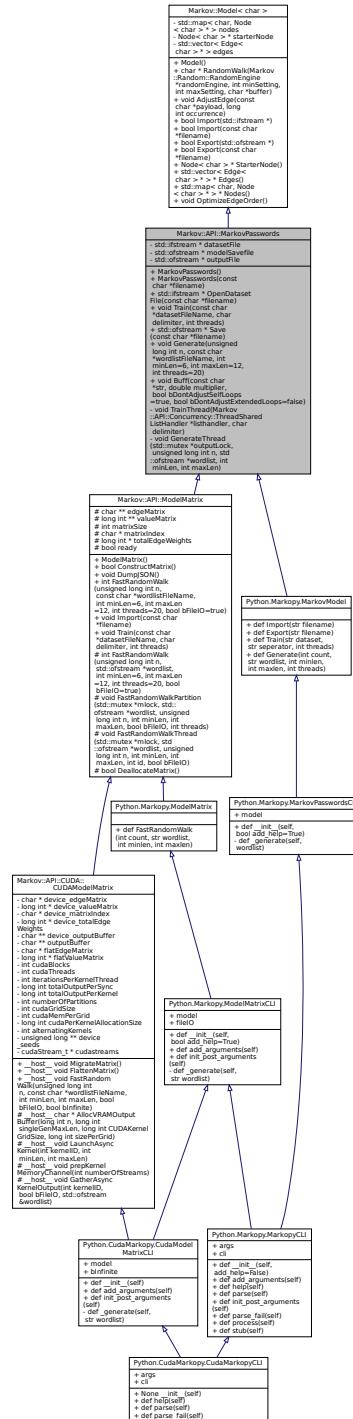
- [Markopy/Markopy/src/CLI/mm.py](#)

9.17 Markov::API::MarkovPasswords Class Reference

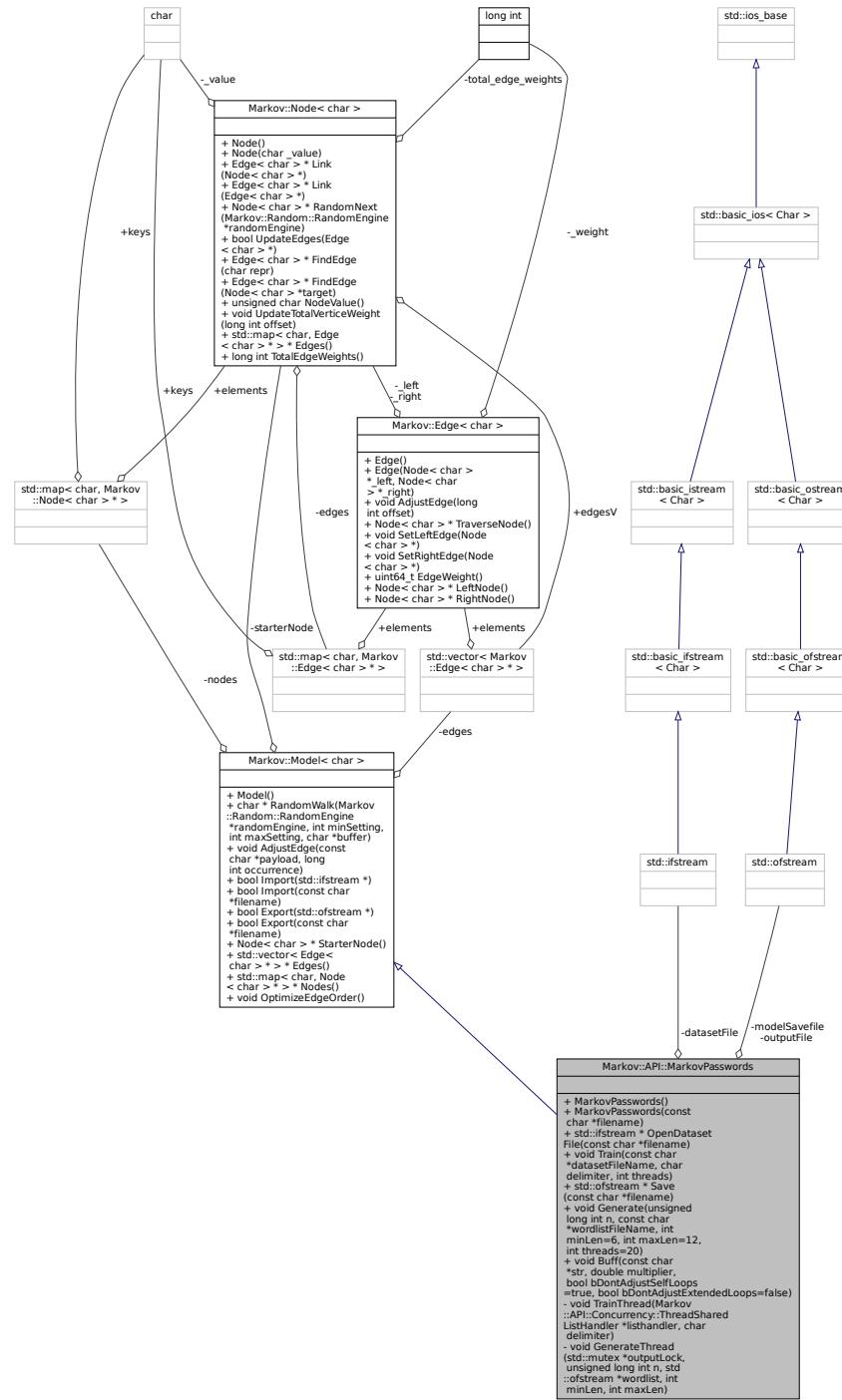
[Markov::Model](#) with char represented nodes.

```
#include <markovPasswords.h>
```

Inheritance diagram for Markov::API::MarkovPasswords:



Collaboration diagram for Markov::API::MarkovPasswords::



Public Member Functions

- **MarkovPasswords ()**
Initialize the markov model from MarkovModel::Markov::Model.
 - **MarkovPasswords (const char *filename)**
Initialize the markov model from MarkovModel::Markov::Model, with an import file.
 - **std::ifstream * OpenDatasetFile (const char *filename)**
Open dataset file and return the ifstream pointer.

- void **Train** (const char *datasetFileName, char delimiter, int threads)

Train the model with the dataset file.
- std::ofstream * **Save** (const char *filename)

Export model to file.
- void **Generate** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

Call Markov::Model::RandomWalk n times, and collect output.
- void **Buff** (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)

Buff expression of some characters in the model.
- char * **RandomWalk** (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

Do a random walk on this model.
- void **AdjustEdge** (const char *payload, long int occurrence)

Adjust the model with a single string.
- bool **Import** (std::ifstream *)

Import a file to construct the model.
- bool **Import** (const char *filename)

Open a file to import with filename, and call bool Model::Import with std::ifstream.
- bool **Export** (std::ofstream *)

Export a file of the model.
- bool **Export** (const char *filename)

Open a file to export with filename, and call bool Model::Export with std::ofstream.
- **Node< char > * StarterNode ()**

Return starter Node.
- std::vector< **Edge< char > * > * Edges ()**

Return a vector of all the edges in the model.
- std::map< char, **Node< char > * > * Nodes ()**

Return starter Node.
- void **OptimizeEdgeOrder ()**

Sort edges of all nodes in the model ordered by edge weights.

Private Member Functions

- void **TrainThread** (Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char delimiter)

A single thread invoked by the Train function.
- void **GenerateThread** (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)

A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * **datasetFile**

Dataset file input of our system
- std::ofstream * **outputFile**

File to save model of our system
- std::map< char, **Node< char > * > nodes**

Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- **Node< char > * starterNode**

Starter Node of this model.
- std::vector< **Edge< char > * > edges**

A list of all edges in this model.

9.17.1 Detailed Description

[Markov::Model](#) with char represented nodes.

Includes wrappers for [Markov::Model](#) and additional helper functions to handle file I/O

This class is an extension of [Markov::Model<char>](#), with higher level abstractions such as train and generate.

Definition at line 26 of file [markovPasswords.h](#).

9.17.2 Constructor & Destructor Documentation

9.17.2.1 MarkovPasswords() [1/2]

`Markov::API::MarkovPasswords::MarkovPasswords ()`

Initialize the markov model from [MarkovModel::Markov::Model](#).

Parent constructor. Has no extra functionality.

Definition at line 34 of file [markovPasswords.cpp](#).

```
00034 : Markov::Model<char> () {  
00035  
00036  
00037 }
```

9.17.2.2 MarkovPasswords() [2/2]

`Markov::API::MarkovPasswords::MarkovPasswords (`
 `const char * filename)`

Initialize the markov model from [MarkovModel::Markov::Model](#), with an import file.

This function calls the [Markov::Model::Import](#) on the filename to construct the model. Same thing as creating an empty model, and calling [MarkovPasswords::Import](#) on the filename.

Parameters

<code>filename</code>	- Filename to import
-----------------------	----------------------

Example Use: Construction via filename

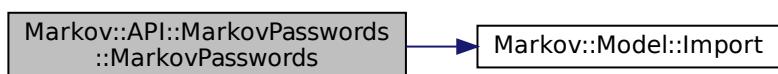
`MarkovPasswords mp("test.mdl");`

Definition at line 39 of file [markovPasswords.cpp](#).

```
00039 {  
00040  
00041     std::ifstream* importFile;  
00042  
00043     this->Import(filename);  
00044  
00045     //std::ifstream* newFile(filename);  
00046  
00047     //importFile = newFile;  
00048  
00049 }
```

References [Markov::Model< NodeStorageType >::Import\(\)](#).

Here is the call graph for this function:



9.17.3 Member Function Documentation

9.17.3.1 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
    const char * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.17.3.2 Buff()

```
void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false )
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

```
00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
```

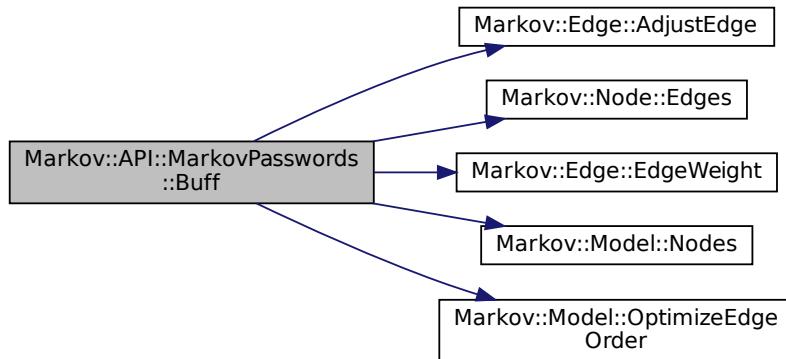
```

00160     edges = node->Edges();
00161     for (auto const& [targetrepr, edge] : *edges){
00162         if(buffstr.find(targetrepr) != std::string::npos) {
00163             if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164             if(buffstr.find(repr) != std::string::npos) {
00165                 continue;
00166             }
00167         }
00168         long int weight = edge->EdgeWeight();
00169         weight = weight*multiplier;
00170         edge->AdjustEdge(weight);
00171     }
00172     i++;
00173 }
00174 this->OptimizeEdgeOrder();
00175
00176 }
00177
00178 }
```

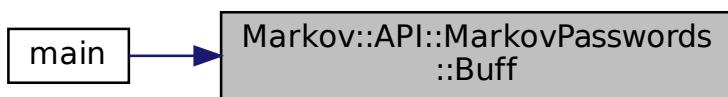
References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.17.3.3 Edges()

```
std::vector<Edge<char *>*>* Markov::Model< char >::Edges ( ) [inline], [inherited]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.17.3.4 Export() [1/2]

```
bool Markov::Model<char>::Export (
    const char * filename ) [inherited]
```

Open a file to export with filename, and call bool [Model::Export](#) with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.17.3.5 Export() [2/2]

```
bool Markov::Model<char>::Export (
    std::ofstream * f ) [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file [model.h](#).

```
00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
00293         e->RightNode()->NodeValue() << "\n";
00294         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
00295         "\n";
00296     }
00297     return true;
00298 }
```

9.17.3.6 Generate()

```
void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 )
```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

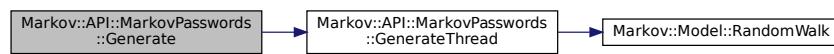
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

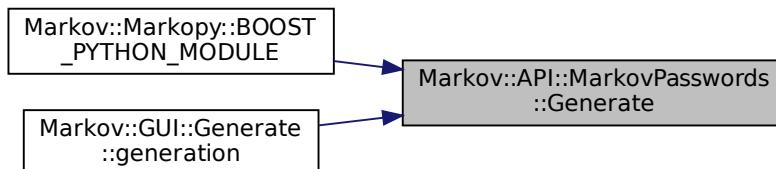
```
00118     {
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++) {
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129             &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130     }
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00136 }
00137 }
```

References [GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).
Here is the call graph for this function:



Here is the caller graph for this function:



9.17.3.7 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
    std::mutex * outputLock,
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen,
    int maxLen ) [private]
```

A single thread invoked by the Generate function.

DEPRECATED: See `Markov::API::MatrixModel::FastRandomWalkThread` for more information. This has been replaced with a much more performance-optimized method. `FastRandomWalk` will reduce the runtime by %96.5 on average.

Parameters

<code>outputLock</code>	- shared mutex lock to lock during output operation. Prevents race condition on write.
<code>n</code>	number of lines to be generated by this thread
<code>wordlist</code>	wordlistfile
<code>minLen</code>	- Minimum password length to generate
<code>maxLen</code>	- Maximum password length to generate

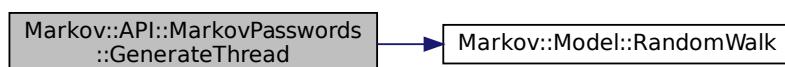
Definition at line 140 of file [markovPasswords.cpp](#).

```
00140
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist << res << "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

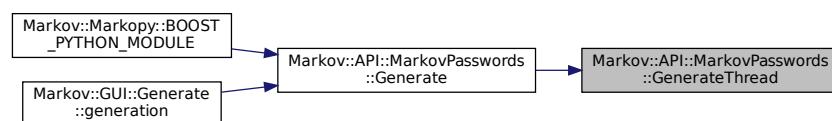
References [Markov::Model< NodeStorageType >::RandomWalk\(\)](#).

Referenced by [Generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.17.3.8 Import() [1/2]

```
bool Markov::Model< char >::Import (
    const char * filename ) [inherited]
```

Open a file to import with filename, and call bool `Model::Import` with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Definition at line 137 of file `model.h`.

```
00280
00281     std::ifstream importfile;
00282     importfile.open(filename);
00283     return this->Import(&importfile);
00284
00285 }
```

9.17.3.9 Import() [2/2]

```
bool Markov::Model< char >::Import (
    std::ifstream * f ) [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file `model.h`.

```
00216
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00245             //std::cout << "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253 }
```

```

00254     //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00255     int(targetN->NodeValue()) << "\n";
00256 }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.17.3.10 Nodes()

`std::map<char , Node<char *>>* Markov::Model< char >::Nodes () [inline], [inherited]`
 Return starter [Node](#).

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.17.3.11 OpenDatasetFile()

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename )
```

Open dataset file and return the ifstream pointer.

Parameters

<code>filename</code>	- Filename to open
-----------------------	--------------------

Returns

`ifstream*` to the the dataset file

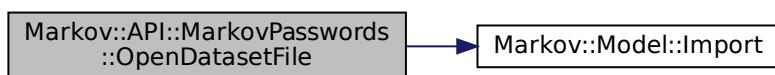
Definition at line 51 of file [markovPasswords.cpp](#).

```

00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References [Markov::Model< NodeStorageType >::Import\(\)](#).

Here is the call graph for this function:



9.17.3.12 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder [inherited]
```

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file [model.h](#).

```
00265     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00266         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00267         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
00268             Edge<NodeStorageType> *rhs) ->bool{
00269                 return lhs->EdgeWeight() > rhs->EdgeWeight();
00270             });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00273         //std::cout << "\n";
00274     }
00275     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00276     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00277 }
```

9.17.3.13 RandomWalk()

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    char * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```
00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316     }
00317 }
```

```

00316     else if ((temp_node == NULL) && (len < minSetting)) {
00317         continue;
00318     }
00319
00320     else if (temp_node == NULL) {
00321         break;
00322     }
00323
00324     n = temp_node;
00325
00326     buffer[len++] = n->NodeValue();
00327 }
00328
00329 //null terminate the string
00330 buffer[len] = 0x00;
00331
00332 //do something with the generated string
00333 return buffer; //for now
00334 }
```

9.17.3.14 Save()

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename )
```

Export model to file.

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

`std::ofstream*` of the exported file.

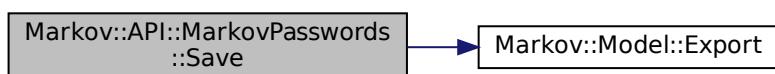
Definition at line 106 of file [markovPasswords.cpp](#).

```

00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.17.3.15 StarterNode()

```
Node< char *>* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
```

Return starter [Node](#).

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.17.3.16 Train()

```
void Markov::API::MarkovPasswords::Train (
    const char * datasetFileName,
    char delimiter,
    int threads )
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

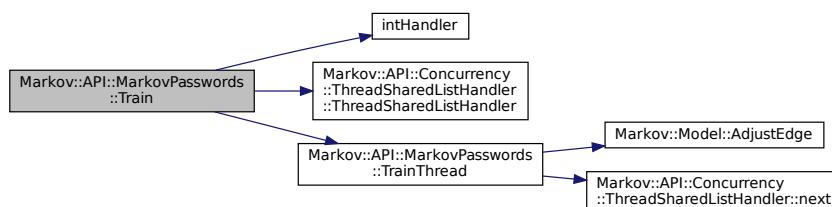
Definition at line 65 of file [markovPasswords.cpp](#).

```
00065 {
00066     signal(SIGINT, intHandler);
00067     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00068     auto start = std::chrono::high_resolution_clock::now();
00069
00070     std::vector<std::thread*> threadsV;
00071     for(int i=0;i<threads;i++) {
00072         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
00073             &listhandler, delimiter));
00074     }
00075     for(int i=0;i<threads;i++) {
00076         threadsV[i]->join();
00077         delete threadsV[i];
00078     }
00079     auto finish = std::chrono::high_resolution_clock::now();
00080     std::chrono::duration<double> elapsed = finish - start;
00081     std::cout << "Elapsed time: " << elapsed.count() << " s\n";
00082
00083 }
```

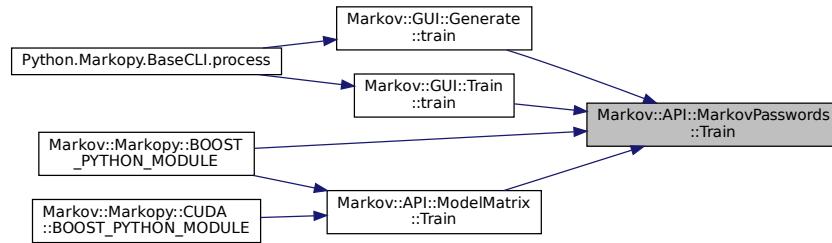
References [intHandler\(\)](#), [Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler\(\)](#), and [TrainThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), [Markov::GUI::Generate::train\(\)](#), [Markov::GUI::Train::train\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.17.3.17 TrainThread()

```
void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private]
```

A single thread invoked by the Train function.

Parameters

<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

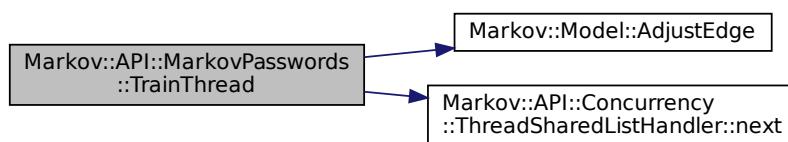
Definition at line 85 of file [markovPasswords.cpp](#).

```
00085
00086     {
00087         char format_str[] = "%ld,%s";
00088         format_str[3]=delimiter;
00089         std::string line;
00090         while (listhandler->next(&line) && keepRunning) {
00091             long int oc;
00092             if (line.size() > 100) {
00093                 line = line.substr(0, 100);
00094             }
00095             char* linebuf = new char[line.length()+5];
00096             sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
00097             "%ld,%s"
00098         else
00099             sscanf(line.c_str(), format_str, &oc, linebuf);
00100     #endif
00101         this->AdjustEdge((const char*)linebuf, oc);
00102         delete linebuf;
00103     }
```

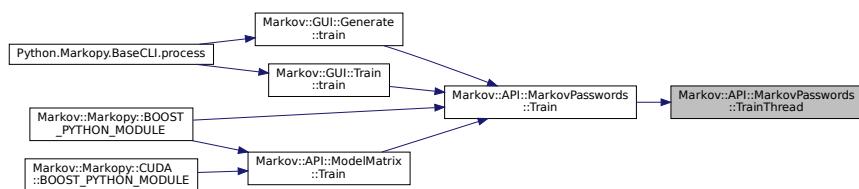
References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), [keepRunning](#), and [Markov::API::Concurrency::ThreadSharedListHandler::next](#).

Referenced by [Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.17.4 Member Data Documentation

9.17.4.1 datasetFile

```
std::ifstream* Markov::API::MarkovPasswords::datasetFile [private]
Definition at line 123 of file markovPasswords.h.
```

9.17.4.2 edges

```
std::vector<Edge<char *>> Markov::Model< char >::edges [private], [inherited]
A list of all edges in this model.
Definition at line 204 of file model.h.
```

9.17.4.3 modelSavefile

```
std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private]
Dataset file input of our system
```

Definition at line 124 of file markovPasswords.h.

9.17.4.4 nodes

```
std::map<char , Node<char *>> Markov::Model< char >::nodes [private], [inherited]
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.
```

9.17.4.5 outputFile

```
std::ofstream* Markov::API::MarkovPasswords::outputFile [private]
File to save model of our system
```

Definition at line 125 of file markovPasswords.h.

9.17.4.6 starterNode

```
Node<char *>* Markov::Model< char >::starterNode [private], [inherited]
Starter Node of this model.
Definition at line 198 of file model.h.
```

The documentation for this class was generated from the following files:

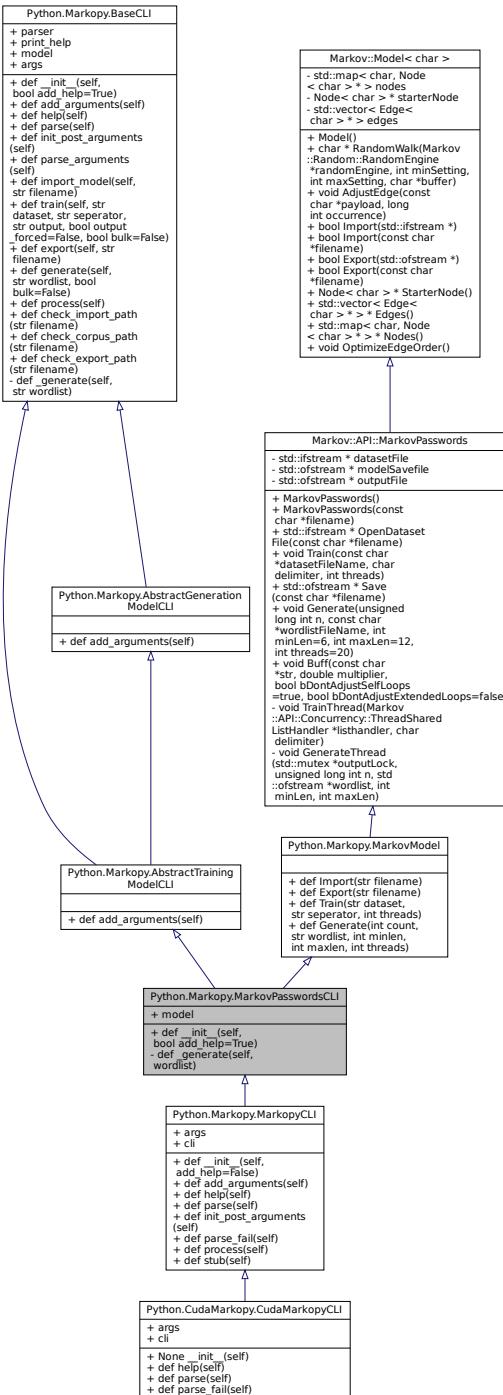
- Markopy/MarkovAPI/src/markovPasswords.h

- [Markopy/MarkovAPI/src/markovPasswords.cpp](#)

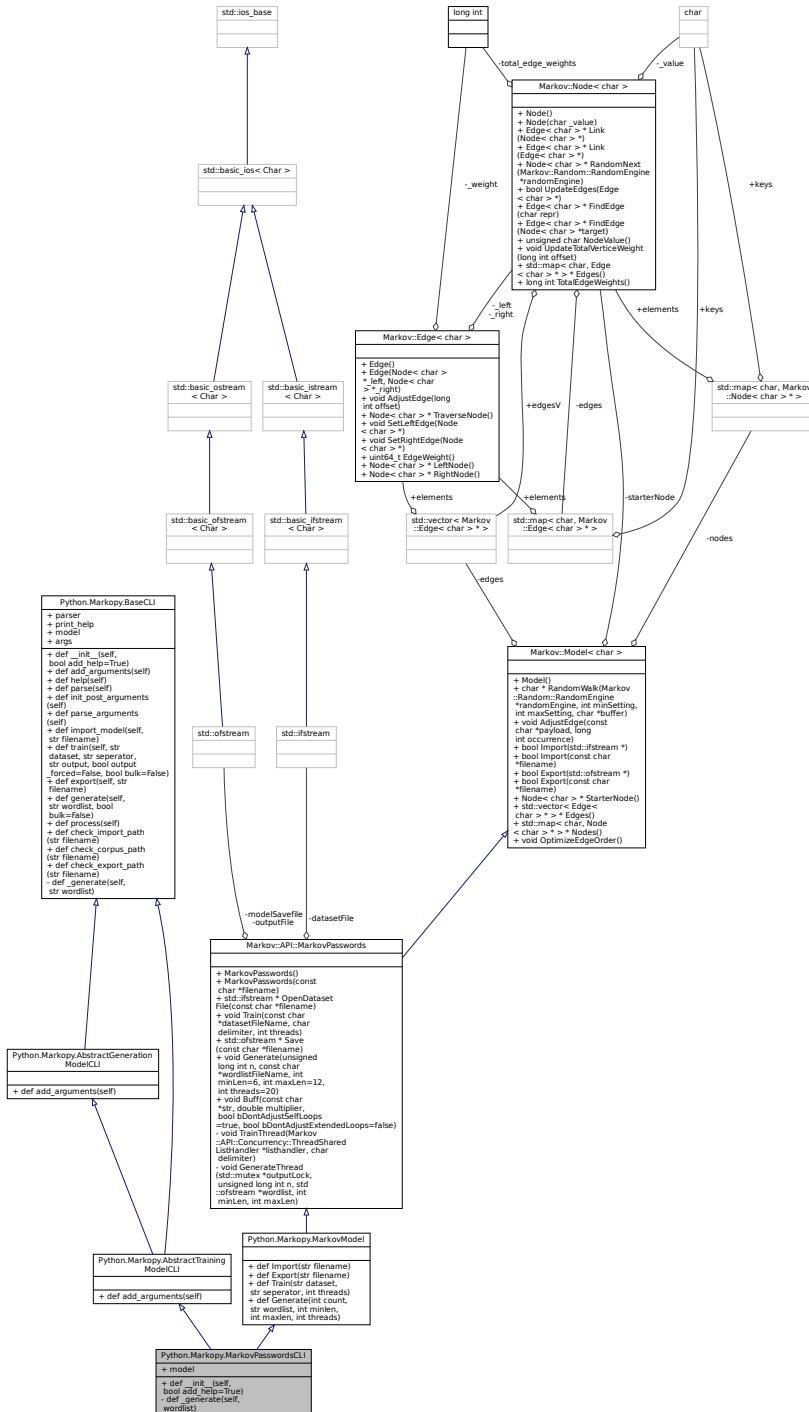
9.18 Python.Markopy.MarkovPasswordsCLI Class Reference

Extension of `Python.Markopy.Base.BaseCLI` for [Markov::API::MarkovPasswords](#).

Inheritance diagram for `Python.Markopy.MarkovPasswordsCLI`:



Collaboration diagram for Python.Markopy.MarkovPasswordsCLI:



Public Member Functions

- def `__init__` (self, bool add_help=True)
initialize base CLI
 - def `add_arguments` (self)
 - def `help` (self)
 - def `help` (self)
 - def `parse` (self)

- def **parse** (self)
- def **init_post_arguments** (self)
- def **init_post_arguments** (self)
- def **parse_arguments** (self)
- def **parse_arguments** (self)
- def **import_model** (self, str filename)

Import a model file.
- def **import_model** (self, str filename)

Import a model file.
- def **train** (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def **train** (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def **export** (self, str filename)

Export model to a file.
- def **export** (self, str filename)

Export model to a file.
- def **generate** (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def **generate** (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def **process** (self)

Process parameters for operation.
- def **process** (self)

Process parameters for operation.
- def **Import** (str filename)
- bool **Import** (std::ifstream *)

Import a file to construct the model.
- bool **Import** (const char *filename)

Open a file to import with filename, and call bool Model::Import with std::ifstream.
- def **Export** (str filename)
- bool **Export** (std::ofstream *)

Export a file of the model.
- bool **Export** (const char *filename)

Open a file to export with filename, and call bool Model::Export with std::ofstream.
- def **Train** (str dataset, str seperator, int threads)
- void **Train** (const char *datasetFileName, char delimiter, int threads)

Train the model with the dataset file.
- def **Generate** (int count, str wordlist, int minlen, int maxlen, int threads)
- void **Generate** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

Call Markov::Model::RandomWalk n times, and collect output.
- std::ifstream * **OpenDatasetFile** (const char *filename)

Open dataset file and return the ifstream pointer.
- std::ofstream * **Save** (const char *filename)

Export model to file.
- void **Buff** (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)

Buff expression of some characters in the model.
- char * **RandomWalk** (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, char *buffer)

Do a random walk on this model.

- void [AdjustEdge](#) (const char *payload, long int occurrence)
Adjust the model with a single string.
- Node< char > * [StarterNode](#) ()
Return starter Node.
- std::vector< Edge< char > * > * [Edges](#) ()
Return a vector of all the edges in the model.
- std::map< char, Node< char > * > * [Nodes](#) ()
Return starter Node.
- void [OptimizeEdgeOrder](#) ()
Sort edges of all nodes in the model ordered by edge weights.

Static Public Member Functions

- def [check_import_path](#) (str filename)
check import path for validity
- def [check_import_path](#) (str filename)
check import path for validity
- def [check_corpus_path](#) (str filename)
check import path for validity
- def [check_corpus_path](#) (str filename)
check import path for validity
- def [check_export_path](#) (str filename)
check import path for validity
- def [check_export_path](#) (str filename)
check import path for validity

Public Attributes

- model
- parser
- parser
- print_help
- print_help
- args
- args

Private Member Functions

- def [_generate](#) (self, wordlist)
- void [TrainThread](#) (Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char delimiter)
A single thread invoked by the Train function.
- void [GenerateThread](#) (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)
A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * [datasetFile](#)
Dataset file input of our system
- std::ofstream * [modelSavefile](#)
File to save model of our system
- std::ofstream * [outputFile](#)
File to save model of our system

- std::map< char, Node< char > * > nodes
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > * starterNode
Starter Node of this model.
- std::vector< Edge< char > * > edges
A list of all edges in this model.

9.18.1 Detailed Description

Extension of Python.Markopy.Base.BaseCLI for [Markov::API::MarkovPasswords](#).
adds -st/-stdout arguement to the command line.
Definition at line 17 of file [mp.py](#).

9.18.2 Constructor & Destructor Documentation

9.18.2.1 __init__()

```
def Python.Markopy.MarkovPasswordsCLI.__init__ (
    self,
    bool add_help = True )
initialize base CLI
```

Parameters

<code>add_help</code>	decide to overload the help function or not
-----------------------	---

Reimplemented from [Python.Markopy.BaseCLI](#).

Definition at line 26 of file [mp.py](#).

```
00026     def __init__(self, add_help=True):
00027         """! @brief initialize model with Markov::API::MarkovPasswords"""
00028         super().__init__(add_help)
00029         self.model = markopy.MarkovPasswords()
00030
```

9.18.3 Member Function Documentation

9.18.3.1 _generate()

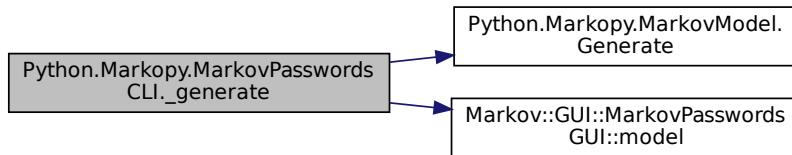
```
def Python.Markopy.MarkovPasswordsCLI._generate (
    self,
    wordlist ) [private]
```

Definition at line 31 of file [mp.py](#).

```
00031     def _generate(self, wordlist):
00032         """! @brief map generation function to Markov::API::MarkovPasswords::Generate"""
00033         self.model.Generate(int(self.args.count), wordlist, int(self.args.min), int(self.args.max),
00034             int(self.args.threads))
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.MarkovModel.Generate\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model](#).
Referenced by [Python.Markopy.BaseCLI.generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.2 add_arguments()

```
def Python.Markopy.AbstractTrainingModelCLI.add_arguments (
    self ) [inherited]
```

Reimplemented from [Python.Markopy.BaseCLI](#).

Reimplemented in [Python.Markopy.MarkopyCLI](#).

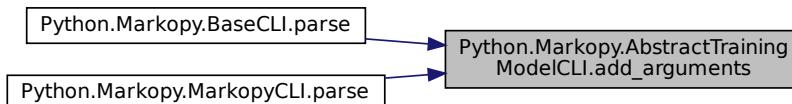
Definition at line 282 of file [base.py](#).

```
00282     def add_arguments(self):
00283         "Add command line arguments to the parser"
00284         self.parser.add_argument("-o", "--output",
00285             help="Output model file. This
00286             model will be exported when done. Will be ignored for generation mode.")
00287         self.parser.add_argument("-d", "--dataset",
00288             help="Dataset file to read input
00289             from for training. Will be ignored for generation mode.")
00290         self.parser.add_argument("-s", "--separator",
00291             help="Separator character to use
00292             with training data.(character between occurrence and value)")
00293         super().add_arguments()
```

References [Python.Markopy.BaseCLI.parser](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.18.3.3 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

<i>string</i>	- String that is passed from the training, and will be used to AdjustEdge the model with
<i>occurrence</i>	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354
00355 }
```

9.18.3.4 Buff()

```
void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]
```

Buff expression of some characters in the model.

Parameters

<i>str</i>	A string containing all the characters to be buffed
<i>multiplier</i>	A constant value to buff the nodes with.
<i>bDontAdjustSelfEdges</i>	Do not adjust weights if target node is same as source node
<i>bDontAdjustExtendedLoops</i>	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

```
00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes){
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges){
00162             if(buffstr.find(targetrepr)!= std::string::npos){
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops){
00165                     if(buffstr.find(repr)!= std::string::npos){
00166                         continue;
00167                     }
00168                 }
00169             }
00170         }
00171     }
00172 }
```

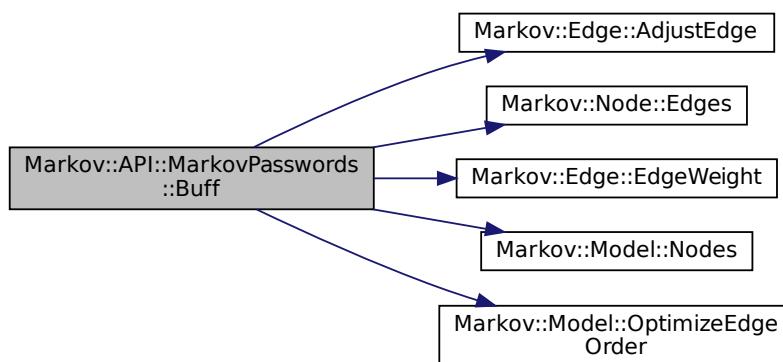
```

00168         }
00169         long int weight = edge->EdgeWeight ();
00170         weight = weight*multiplier;
00171         edge->AdjustEdge (weight);
00172     }
00173 }
00174     i++;
00175 }
00176 }
00177 this->OptimizeEdgeOrder ();
00178 }
```

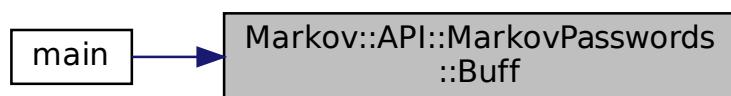
References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.5 check_corpus_path() [1/2]

```

def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```

00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.18.3.6 check_corpus_path() [2/2]

```

def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```

00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.18.3.7 check_export_path() [1/2]

```

def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

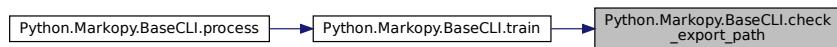
```

00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.18.3.8 check_export_path() [2/2]

```

def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```

00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201

```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.18.3.9 check_import_path() [1/2]

```

def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity

```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

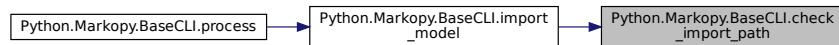
```

00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179

```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.18.3.10 check_import_path() [2/2]

```

def Python.Markopy.BaseCLI.check_import_path (
        str filename ) [static], [inherited]
check import path for validity

```

Parameters

<code>filename</code>	filename to check
-----------------------	-------------------

Definition at line 169 of file [base.py](#).

```

00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179

```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:



9.18.3.11 Edges()

```

std::vector<Edge<char *>>* Markov::Model< char >::Edges ( ) [inline], [inherited]
Return a vector of all the edges in the model.

```

Returns

vector of edges

Definition at line 176 of file [model.h](#).

```

00176 { return &edges; }

```

9.18.3.12 Export() [1/3]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file model.h.

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.18.3.13 export() [1/2]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

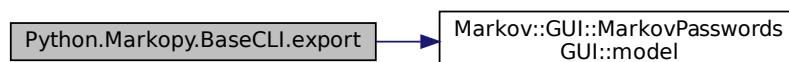
Definition at line 138 of file base.py.

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142             """
00143         self.model.Export(filename)
00144
```

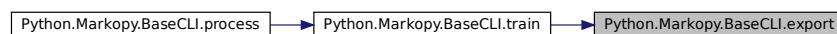
References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.14 `export()` [2/2]

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<code>filename</code>	filename to export to
-----------------------	-----------------------

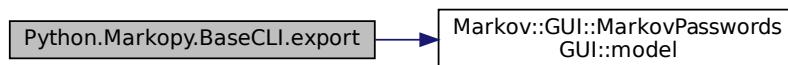
Definition at line 138 of file [base.py](#).

```
00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.15 `Export()` [2/3]

```
bool Markov::Model< char >::Export (
    std::ofstream * f ) [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file [model.h](#).

```
00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
```

```

00292     //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
00293     e->RightNode() ->NodeValue() << "\n";
00294     *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode() ->NodeValue() <<
00295     "\n";
00296     return true;
00297 }
```

9.18.3.16 Export() [3/3]

```

def Python.Markopy.MarkovModel.Export (
    str filename ) [inherited]
Definition at line 26 of file mm.py.
00026     def Export(filename : str):
00027         pass
00028 
```

9.18.3.17 Generate() [1/2]

```

def Python.Markopy.MarkovModel.Generate (
    int count,
    str wordlist,
    int minlen,
    int maxlen,
    int threads ) [inherited]
Definition at line 34 of file mm.py.
00034     def Generate(count : int, wordlist : str, minlen : int, maxlen: int, threads : int):
00035         pass
00036
00037 
```

Referenced by [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#).

Here is the caller graph for this function:



9.18.3.18 generate() [1/2]

```

def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
Generate strings from the model.

```

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file base.py.

```

00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150                 @param bulk marks bulk operation with directories
00151             """
  
```

```

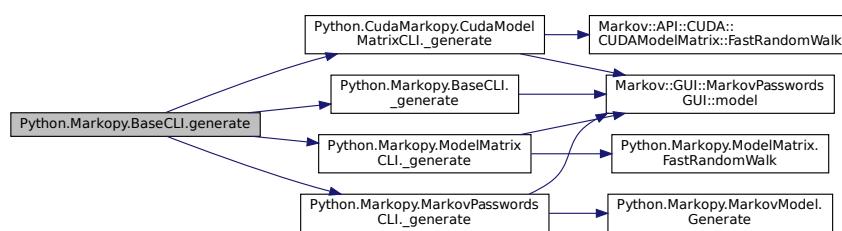
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
00159

```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopy](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.19 generate() [2/2]

```

def Python.Markopy.BaseCLI.generate (
        self,
        str wordlist,
        bool bulk = False )  [inherited]

```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

Definition at line 145 of file [base.py](#).

```

00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00154             return False
00155

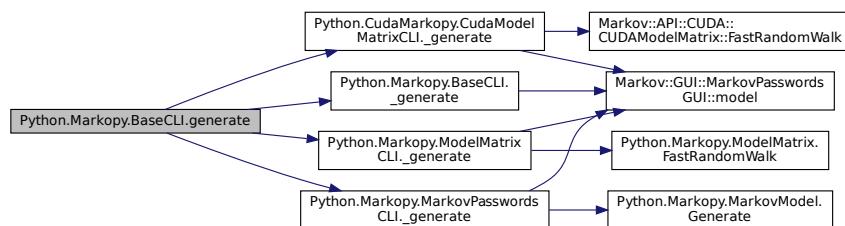
```

```
00156     if(bulk and os.path.isfile(wordlist)):
00157         logging pprint(f"{{wordlist}} exists and will be overwritten.", 1)
00158     self._generate(wordlist)
```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopy](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.20 Generate() [2/2]

```
void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]
```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

```
00118
00119     char* res;
```

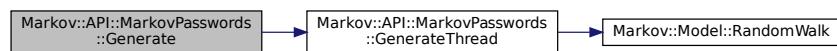
```

00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++) {
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129             &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130     }
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i] ->join();
00133         delete threadsV[i];
00134     }
00135     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00136 }
00137
00138 }
```

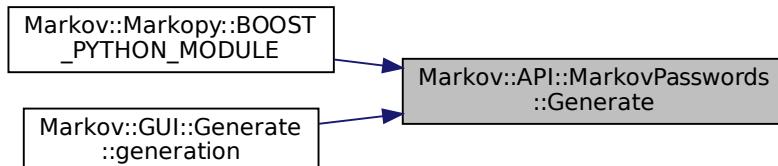
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.21 GenerateThread()

```

void Markov::API::MarkovPasswords::GenerateThread (
    std::mutex * outputLock,
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen,
    int maxLen ) [private], [inherited]
```

A single thread invoked by the Generate function.

DEPRECATED: See [Markov::API::MatrixModel::FastRandomWalkThread](#) for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Parameters

<i>outputLock</i>	- shared mutex lock to lock during output operation. Prevents race condition on write.
<i>n</i>	number of lines to be generated by this thread
<i>wordlist</i>	wordlistfile

Parameters

<code>minLen</code>	- Minimum password length to generate
<code>maxLen</code>	- Maximum password length to generate

Definition at line 140 of file [markovPasswords.cpp](#).

```
00140
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist << res << "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

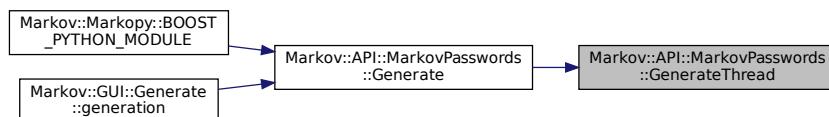
References [Markov::Model< NodeStorageType >::RandomWalk\(\)](#).

Referenced by [Markov::API::MarkovPasswords::Generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.22 help() [1/2]

```
def Python.Markopy.BaseCLI.help (
    self) [inherited]
```

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 51 of file [base.py](#).

```
00051     def help(self):
00052         """! @brief Handle help strings. Defaults to argparse's help"""
00053         self.print_help()
00054
```

References [Python.Markopy.BaseCLI.print_help](#).

Referenced by [Python.Markopy.MarkopyCLI.add_arguments\(\)](#).

Here is the caller graph for this function:



9.18.3.23 help() [2/2]

```
def Python.Markopy.BaseCLI.help (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 51 of file [base.py](#).

```
00051     def help(self):
00052         """! @brief Handle help strings. Defaults to argparse's help"""
00053         self.print_help()
00054
```

References [Python.Markopy.BaseCLI.print_help](#).

Referenced by [Python.Markopy.MarkopyCLI.add_arguments\(\)](#).

Here is the caller graph for this function:



9.18.3.24 Import() [1/3]

```
bool Markov::Model< char >::Import (
    const char * filename ) [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Definition at line 137 of file [model.h](#).

```
00280
00281     std::ifstream importfile;
00282     importfile.open(filename);
00283     return this->Import(&importfile);
00284
00285 }
```

9.18.3.25 Import() [2/3]

```
bool Markov::Model< char >::Import (
    std::ifstream * f ) [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file [model.h](#).

```
00216 {
```

```

00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>*>(target, targetN));
00245             //std::cout << "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00255         int(targetN->NodeValue()) << "\n";
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.18.3.26 Import() [3/3]

```

def Python.Markopy.MarkovModel.Import (
    str filename ) [inherited]
Definition at line 22 of file mm.py.
00022     def Import(filename : str):
00023         pass
00024
```

9.18.3.27 import_model() [1/2]

```

def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]

```

Import a model file.

Parameters

<code>filename</code>	filename to import
-----------------------	--------------------

Definition at line 77 of file base.py.

```

00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081             """
00082         logging pprint("Importing model file.", 1)
```

```

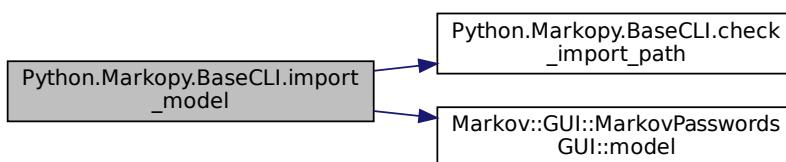
00083
00084     if not self.check_import_path(filename):
00085         logging pprint(f"Model file at {filename} not found. Check the file path, or working
00086         directory")
00087         return False
00088
00089     self.model.Import(filename)
00090     logging pprint("Model imported successfully.", 2)
00091
00092
00093

```

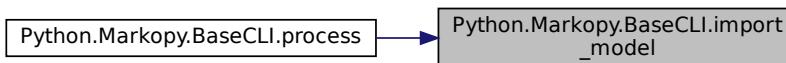
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.28 import_model() [2/2]

```

def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]

```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

Definition at line 77 of file [base.py](#).

```

00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
00085             logging pprint(f"Model file at {filename} not found. Check the file path, or working
00086             directory")
00087             return False
00088
00089         self.model.Import(filename)

```

```

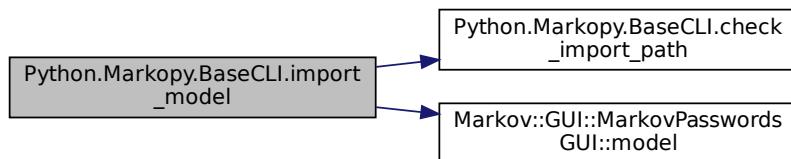
00089     logging pprint("Model imported successfully.", 2)
00090     return True
00091
00092
00093

```

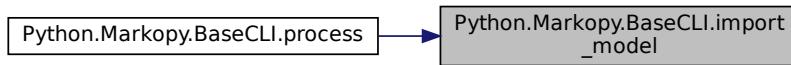
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.29 init_post_arguments() [1/2]

```

def Python.Markopy.BaseCLI.init_post_arguments (
    self ) [inherited]

```

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

Definition at line 62 of file [base.py](#).

```

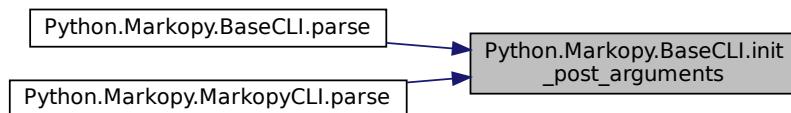
00062     def init_post_arguments(self):
00063         """! @brief set up stuff that is collected from command line arguments"""
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.args.verbosity:
00067                 logging.VERBOSITY = self.args.verbosity
00068                 logging pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
00070             pass
00071

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.18.3.30 init_post_arguments() [2/2]

```
def Python.Markopy.BaseCLI.init_post_arguments (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.ModelMatrixCLI](#), [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

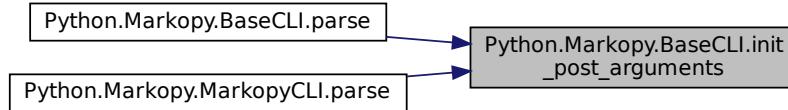
Definition at line 62 of file [base.py](#).

```
00062     def init_post_arguments(self):
00063         """! @brief set up stuff that is collected from command line arguments"""
00064         logging.VERBOSITY = 0
00065         try:
00066             if self.args.verbosity:
00067                 logging.VERBOSITY = self.args.verbosity
00068                 logging pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00069         except:
00070             pass
00071 
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.18.3.31 Nodes()

```
std::map<char , Node<char >>* Markov::Model< char >::Nodes ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.18.3.32 OpenDatasetFile()

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename ) [inherited]
```

Open dataset file and return the ifstream pointer.

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

*ifstream** to the the dataset file

Definition at line 51 of file [markovPasswords.cpp](#).

```
00051
00052
00053     std::ifstream* datasetFile;
```

{

```

00054     std::ifstream newFile(filename);
00055     datasetFile = &newFile;
00056
00057     this->Import(datasetFile);
00058
00059     return datasetFile;
00060 }
00061 }
```

References [Markov::Model< NodeStorageType >::Import\(\)](#).

Here is the call graph for this function:



9.18.3.33 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder [inherited]
```

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file [model.h](#).

```

00265
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType*>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
00269             Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight() > rhs->EdgeWeight();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00278 }
```

9.18.3.34 parse() [1/2]

```
def Python.Markopy.BaseCLI.parse (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

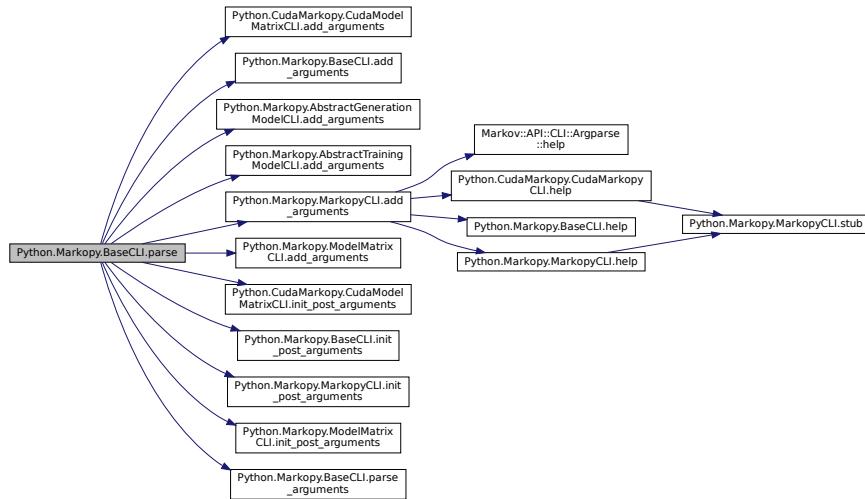
Definition at line 55 of file [base.py](#).

```

00055     def parse(self):
00056         """! @brief add, parse and hook arguments"""
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060 }
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.init_post_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.init_post_arguments\(\)](#), and [Python.Markopy.BaseCLI.parse_arguments\(\)](#).

Here is the call graph for this function:



9.18.3.35 parse() [2/2]

```
def Python.Markopy.BaseCLI.parse (
    self ) [inherited]
```

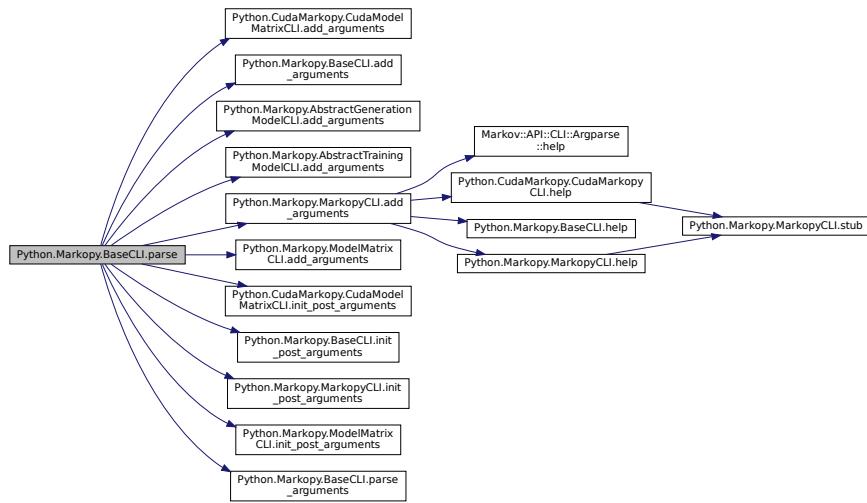
Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 55 of file [base.py](#).

```
00055     def parse(self):
00056         """! @brief add, parse and hook arguments"""
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.init_post_arguments\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.init_post_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.init_post_arguments\(\)](#) and [Python.Markopy.BaseCLI.parse_arguments\(\)](#).

Here is the call graph for this function:



9.18.3.36 parse_arguments() [1/2]

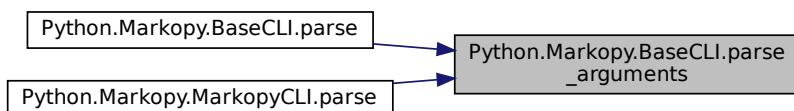
```
def Python.Markopy.BaseCLI.parse_arguments (
    self) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.18.3.37 parse_arguments() [2/2]

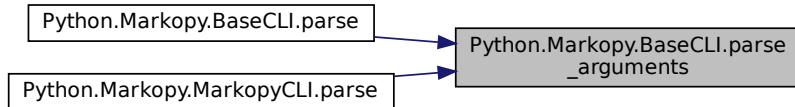
```
def Python.Markopy.BaseCLI.parse_arguments (
    self) [inherited]
```

Definition at line 73 of file [base.py](#).

```
00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args()[0]
00076
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.18.3.38 process() [1/2]

```
def Python.Markopy.BaseCLI.process (
    self ) [inherited]
```

Process parameters for operation.

Reimplemented in [Python.Markopy.MarkopyCLI](#).

Definition at line 202 of file [base.py](#).

```
00202     def process(self):
00203         """
00204             @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging pprint(f"\"Bulk mode operation chosen.\", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
00210                     (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00211                     corpus_list = os.listdir(self.args.dataset)
00212                     for corpus in corpus_list:
00213                         self.import_model(self.args.input)
00214                         logging pprint(f"\"Training {self.args.input} with {corpus}\", 2)
00215                         output_file_name = corpus
00216                         model_extension = ""
00217                         if "." in self.args.input:
00218                             model_extension = self.args.input.split(".")[-1]
00219                             self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
00220                                     f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00221                         else:
00222                             logging pprint("In bulk training, output and dataset should be a directory.")
00223                             exit(1)
00224
00225             elif (self.args.mode.lower() == "generate"):
00226                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
00227                     (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00228                     model_list = os.listdir(self.args.input)
00229                     print(model_list)
00230                     for input in model_list:
00231                         logging pprint(f"\"Generating from {self.args.input}/{input} to
00232                             {self.args.wordlist}/{input}.txt\", 2)
00233                         self.import_model(f"{self.args.input}/{input}")
00234                         model_base = input
00235                         if "." in self.args.input:
00236                             model_base = input.split(".")[-1]
00237                             self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00238                         else:
00239                             logging pprint("In bulk generation, input and wordlist should be directory.")
00240
00241
00242             elif (self.args.mode.lower() == "train"):
00243                 self.train(self.args.dataset, self.args.seperator, self.args.output,
00244                         output_forced=True)
00245
00246             elif(self.args.mode.lower() == "combine"):
00247                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00248                 self.generate(self.args.wordlist)
00249
00250
00251             else:
00252                 logging pprint("Invalid mode arguement given.")
```

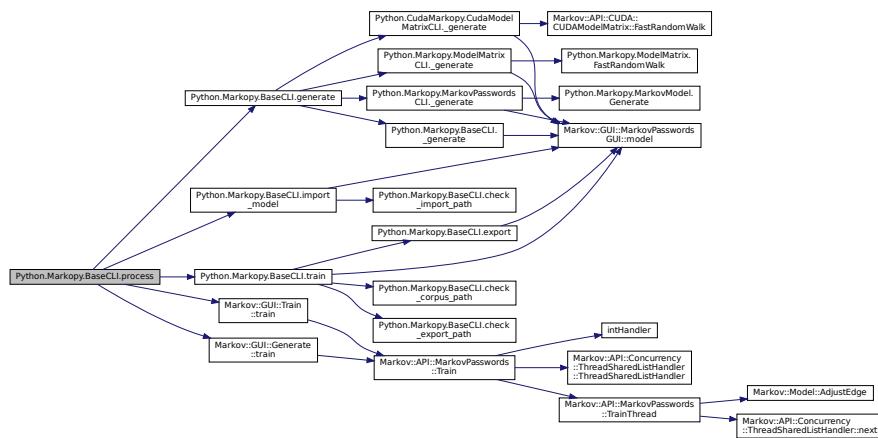
```

00254         logging pprint("Accepted modes: 'Generate', 'Train', 'Combine' ")
00255         exit(5)
00256

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), [Markov::GUI::Generate.train\(\)](#), [Markov::GUI::Train.train\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



9.18.3.39 process() [2/2]

```

def Python.Markopy.BaseCLI.process (
    self ) [inherited]

```

Process parameters for operation.

Reimplemented in [Python.Markopy.MarkopyCLI](#).

Definition at line 202 of file [base.py](#).

```

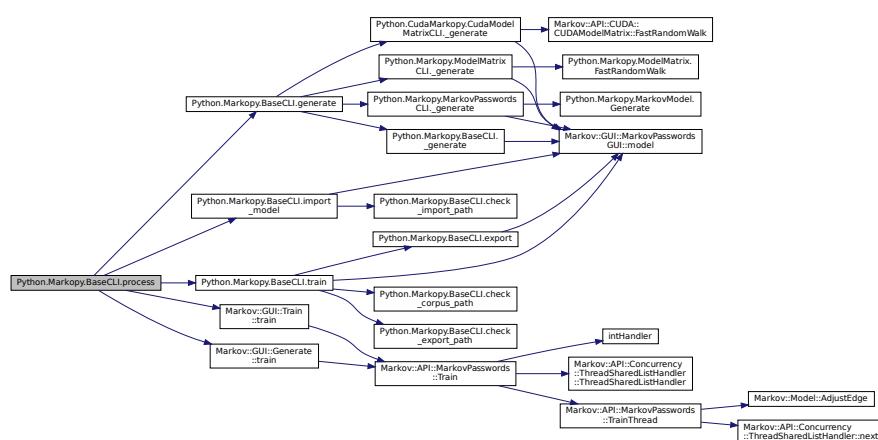
00202     def process(self):
00203         """
00204             @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging pprint(f"\"Bulk mode operation chosen.\", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
00210                     (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00211                     corpus_list = os.listdir(self.args.dataset)
00212                     for corpus in corpus_list:
00213                         self.import_model(self.args.input)
00214                         logging pprint(f"\"Training {self.args.input} with {corpus}\", 2)
00215                         output_file_name = corpus
00216                         model_extension = ""
00217                         if "." in self.args.input:
00218                             model_extension = self.args.input.split(".")[-1]
00219                             self.train(f"{self.args.dataset}/{corpus}", self.args.separator,
00220                                     f"{self.args.output}/{corpus}.{{model_extension}}", output_forced=True, bulk=True)
00221                     else:
00222                         logging pprint("In bulk training, output and dataset should be a directory.")
00223                         exit(1)
00224
00225             elif (self.args.mode.lower() == "generate"):
00226                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
00227                     (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00228                     model_list = os.listdir(self.args.input)
00229                     print(model_list)
00230                     for input in model_list:
00231                         logging pprint(f"\"Generating from {self.args.input}/{input} to
00232 {self.args.wordlist}/{input}.txt\", 2)
00233                         self.import_model(f"{self.args.input}/{input}")
00234                         model_base = input
00235                         if "." in self.args.input:
00236                             model_base = input.split(".")[1]
00237                         self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00238                 else:
00239                     logging pprint("In bulk generation, input and wordlist should be directory.")

```

```
00236
00237     else:
00238         self.import_model(self.args.input)
00239         if (self.args.mode.lower() == "generate"):
00240             self.generate(self.args.wordlist)
00241
00242
00243         elif (self.args.mode.lower() == "train"):
00244             self.train(self.args.dataset, self.args.separator, self.args.output,
00245             output_forced=True)
00246
00247
00248         elif(self.args.mode.lower() == "combine"):
00249             self.train(self.args.dataset, self.args.separator, self.args.output)
00250             self.generate(self.args.wordlist)
00251
00252
00253     else:
00254         logging pprint("Invalid mode argument given.")
00255         logging pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00256         exit(5)
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#),
[Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), [Markov::GUI::Generate.train\(\)](#),
[Markov::GUI::Train.train\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



9.18.3.40 RandomWalk()

```
char * Markov::Model< char >::RandomWalk (  
    Markov::Random::RandomEngine * randomEngine,  
    int minSetting,  
    int maxSetting,  
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia
Marsaglia Model ([Model](#)) model.

```

Model::import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}

```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```

00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319         else if (temp_node == NULL) {
00320             break;
00321         }
00322         n = temp_node;
00323
00324         buffer[len++] = n->NodeValue();
00325
00326     //null terminate the string
00327     buffer[len] = 0x00;
00328
00329     //do something with the generated string
00330     return buffer; //for now
00331
00332
00333
00334 }

```

9.18.3.41 Save()

```

std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]

```

Export model to file.

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

```

00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);

```

```

00110     exportFile = &newFile;
00111     this->Export(exportFile);
00112     return exportFile;
00113 }

```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.18.3.42 StarterNode()

`Node< char *>* Markov::Model< char >::StarterNode () [inline], [inherited]`
Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.18.3.43 Train() [1/2]

```
void Markov::API::MarkovPasswords::Train (
    const char * datasetFileName,
    char delimiter,
    int threads) [inherited]
```

Train the model with the dataset file.

Parameters

<code>datasetFileName</code>	- ifstream* to the dataset. If null, use class member
<code>delimiter</code>	- a character, same as the delimiter in dataset content
<code>threads</code>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

Definition at line 65 of file [markovPasswords.cpp](#).

```

00065
00066     signal(SIGINT, intHandler);
00067     Markov::API::Concurrency::ThreadSharedListHandler lisHandler(datasetFileName);
00068     auto start = std::chrono::high_resolution_clock::now();
00069
00070     std::vector<std::thread*> threadsV;
00071     for(int i=0;i<threads;i++) {
00072         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
00073                                         &lisHandler, delimiter));
00074     }
00075     for(int i=0;i<threads;i++) {
00076         threadsV[i]->join();
00077         delete threadsV[i];
00078     }
00079     auto finish = std::chrono::high_resolution_clock::now();
00080     std::chrono::duration<double> elapsed = finish - start;

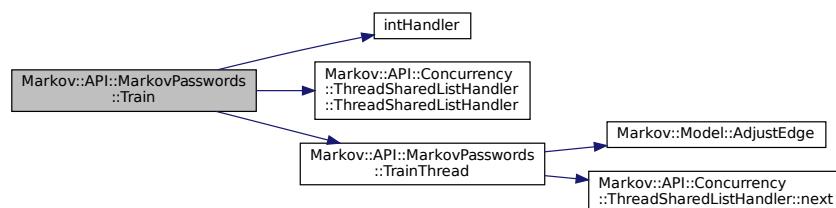
```

```
00081     std::cout << "Elapsed time: " << elapsed.count() << " s\n";
00082
00083 }
```

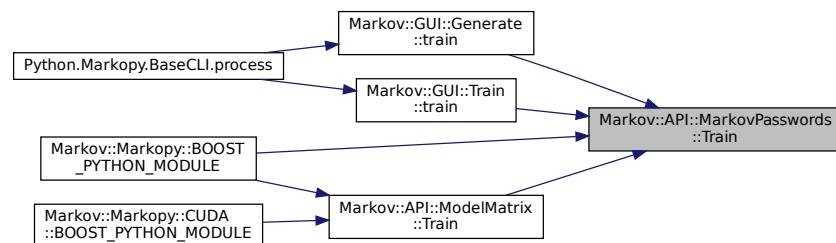
References [intHandler\(\)](#), [Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler\(\)](#), and [Markov::API::MarkovPasswords::TrainThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), [Markov::GUI::Generate::train\(\)](#), [Markov::GUI::Train::train\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.44 train() [1/2]

```
def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False ) [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

```

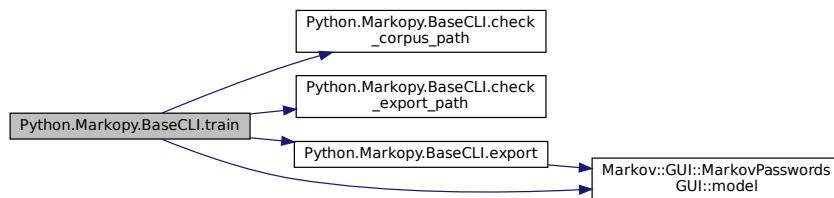
00094     def train(self, dataset : str, separator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param separator separator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104
00105         logging pprint("Training.")
00106
00107         if not (dataset and separator and (output or not output_forced)):
00108             logging pprint(f"Training mode requires -d/--dataset(', -o/--output' if output_forced
00109             else') and -s/--separator parameters. Exiting.")
00110             return False
00111
00112         if not bulk and not self.check_corpus_path(dataset):
00113             logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00114             return False
00115
00116         if not self.check_export_path(output):
00117             logging pprint(f"Cannot create output at {output}")
00118             return False
00119
00120         if(separator == '\\t'):
00121             logging pprint("Escaping separator.", 3)
00122             separator = '\t'
00123
00124         if(len(separator)!=1):
00125             logging pprint(f'Delimiter must be a single character, and "{separator}" is not
00126             accepted.')
00127             exit(4)
00128
00129         logging pprint(f'Starting training.', 3)
00130         self.model.Train(dataset, separator, int(self.args.threads))
00131         logging pprint(f'Training completed.', 2)
00132
00133         if(output):
00134             logging pprint(f'Exporting model to {output}', 2)
00135             self.export(output)
00136         else:
00137             logging pprint(f'Model will not be exported.', 1)
00138
00139     return True
00140

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.45 train() [2/2]

```
def Python.Markopy.BaseCLI.train (
    self,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False ) [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

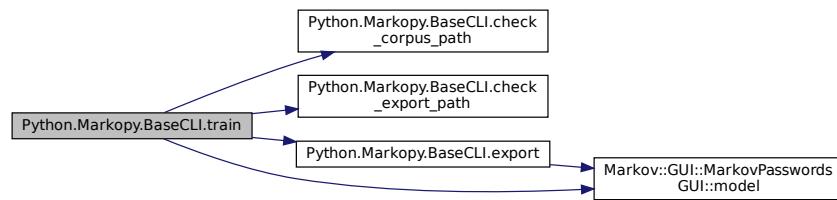
```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104         """
00105         logging pprint("Training.")
00106
00107         if not (dataset and seperator and (output or not output_forced)):
00108             logging pprint(f"Training mode requires -d/--dataset', -o/--output' if output_forced
00109             else") and -s/--seperator parameters. Exiting.")
00110             return False
00111
00112         if not bulk and not self.check_corpus_path(dataset):
00113             logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00114             return False
00115
00116         if not self.check_export_path(output):
00117             logging pprint(f"Cannot create output at {output}")
00118             return False
00119
00120         if(seperator == '\\t'):
00121             logging pprint("Escaping seperator.", 3)
00122             seperator = '\t'
00123
00124         if(len(seperator)!=1):
00125             logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00126             accepted.')
00127             exit(4)
00128
00129         if(output):
00130             logging pprint(f'Starting training.', 3)
00131             self.model.Train(dataset, seperator, int(self.args.threads))
00132             logging pprint(f'Training completed.', 2)
00133
00134         if(output):
00135             logging pprint(f'Exporting model to {output}', 2)
00136             self.export(output)
00137         else:
00138             logging pprint(f'Model will not be exported.', 1)
00139
00140     return True
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#),

[Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.3.46 Train() [2/2]

```

def Python.Markopy.MarkovModel.Train (
    str dataset,
    str seperator,
    int threads ) [inherited]
Definition at line 30 of file mm.py.
00030     def Train(dataset: str, seperator : str, threads : int):
00031         pass
00032
  
```

9.18.3.47 TrainThread()

```

void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private], [inherited]
  
```

A single thread invoked by the Train function.

Parameters

<code>listhandler</code>	- Listhandler class to read corpus from
<code>delimiter</code>	- a character, same as the delimiter in dataset content

Definition at line 85 of file [markovPasswords.cpp](#).

```

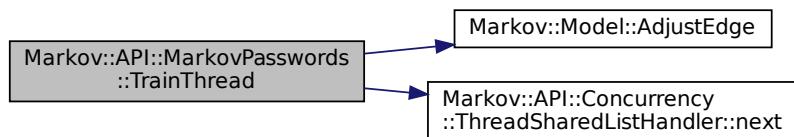
00085
00086     char format_str[] = "%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
  
```

```

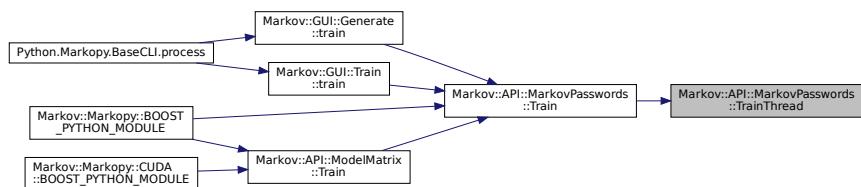
00095 #ifdef _WIN32
00096     sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length() + 5); //<== changed format_str to-
00097     "%ld,%s"
00098     sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100     this->AdjustEdge((const char*)linebuf, oc);
00101     delete linebuf;
00102 }
00103 }
```

References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), [keepRunning](#), and [Markov::API::Concurrency::ThreadSharedListHandler::next](#).
Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.4 Member Data Documentation

9.18.4.1 args [1/2]

`Python.Markopy.BaseCLI.args [inherited]`

Definition at line 75 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.MarkopyCLI.help\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.parse\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.parse_fail\(\)](#), [Python.Markopy.BaseCLI.process\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.18.4.2 args [2/2]

`Python.Markopy.BaseCLI.args [inherited]`

Definition at line 75 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.MarkopyCLI.help\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.parse\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.parse_fail\(\)](#), [Python.Markopy.BaseCLI.process\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.18.4.3 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]`
 Definition at line 123 of file [markovPasswords.h](#).

9.18.4.4 edges

`std::vector<Edge<char *>> Markov::Model< char >::edges [private], [inherited]`
 A list of all edges in this model.
 Definition at line 204 of file [model.h](#).

9.18.4.5 model

`Python.Markopy.MarkovPasswordsCLI.model`
 Definition at line 29 of file [mp.py](#).
 Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.18.4.6 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`
 Dataset file input of our system

Definition at line 124 of file [markovPasswords.h](#).

9.18.4.7 nodes

`std::map<char , Node<char *>> Markov::Model< char >::nodes [private], [inherited]`
 Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
 Definition at line 193 of file [model.h](#).

9.18.4.8 outputFile

`std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]`
 File to save model of our system

Definition at line 125 of file [markovPasswords.h](#).

9.18.4.9 parser [1/2]

`Python.Markopy.BaseCLI.parser [inherited]`
 Definition at line 25 of file [base.py](#).
 Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI](#).

9.18.4.10 parser [2/2]

`Python.Markopy.BaseCLI.parser [inherited]`
 Definition at line 25 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.18.4.11 print_help [1/2]

`Python.Markopy.BaseCLI.print_help [inherited]`

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.18.4.12 print_help [2/2]

`Python.Markopy.BaseCLI.print_help [inherited]`

Definition at line 39 of file [base.py](#).

Referenced by [Python.Markopy.BaseCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.18.4.13 starterNode

`Node<char>*>* Markov::Model< char >::starterNode [private], [inherited]`

Starter Node of this model.

Definition at line 198 of file [model.h](#).

The documentation for this class was generated from the following file:

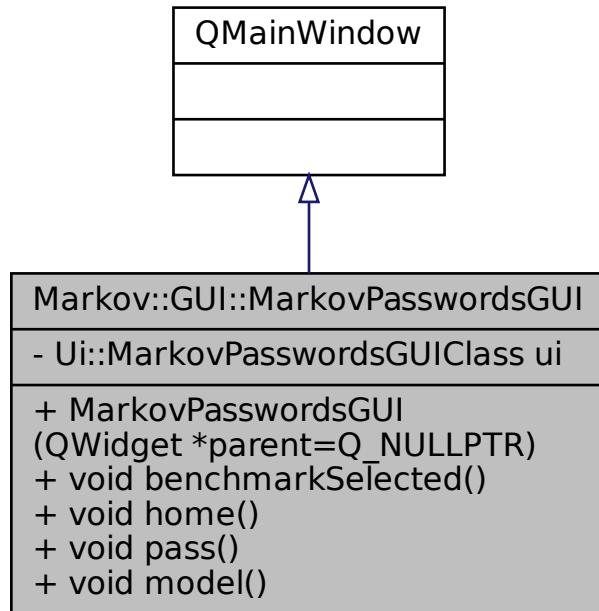
- [Markopy/Markopy/src/CLI/mp.py](#)

9.19 Markov::GUI::MarkovPasswordsGUI Class Reference

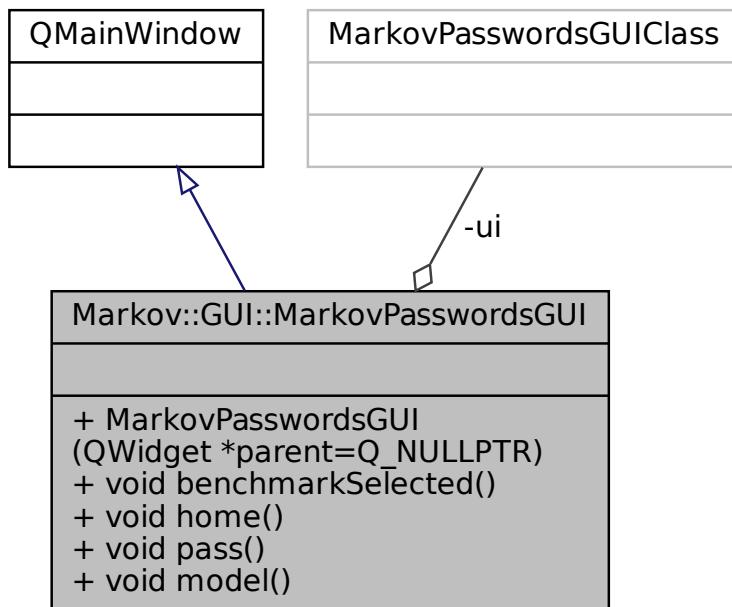
Reporting UI.

```
#include <MarkovPasswordsGUI.h>
```

Inheritance diagram for `Markov::GUI::MarkovPasswordsGUI`:



Collaboration diagram for `Markov::GUI::MarkovPasswordsGUI`:



Public Slots

- void `benchmarkSelected ()`
- void `home ()`
- void `pass ()`
- void `model ()`

Public Member Functions

- `MarkovPasswordsGUI (QWidget *parent=Q_NULLPTR)`

Private Attributes

- `Ui::MarkovPasswordsGUIClass ui`

9.19.1 Detailed Description

Reporting UI.

UI for reporting and debugging tools for MarkovPassword

Definition at line 19 of file `MarkovPasswordsGUI.h`.

9.19.2 Constructor & Destructor Documentation

9.19.2.1 `MarkovPasswordsGUI()`

```
Markov::GUI::MarkovPasswordsGUI::MarkovPasswordsGUI (
    QWidget * parent = Q_NULLPTR )
Definition at line 14 of file MarkovPasswordsGUI.cpp.
00014 : QMainWindow(parent) {
00015     ui.setupUi(this);
00016
00017
00018     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {home(); });
00019     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {model(); });
00020     QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {pass(); });
00021 }
```

References `ui`.

9.19.3 Member Function Documentation

9.19.3.1 `benchmarkSelected`

```
void Markov::GUI::MarkovPasswordsGUI::benchmarkSelected ( ) [slot]
```

9.19.3.2 `home`

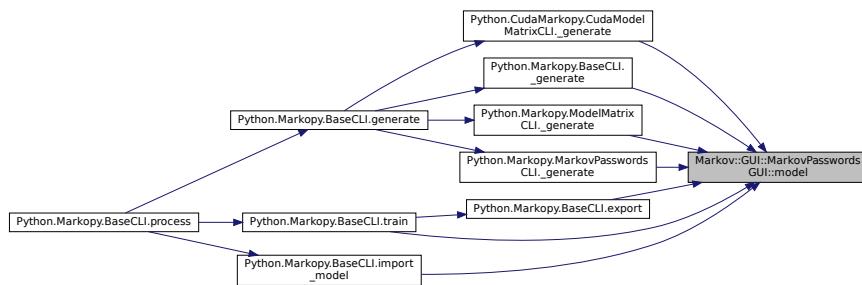
```
void MarkovPasswordsGUI::home ( ) [slot]
Definition at line 24 of file MarkovPasswordsGUI.cpp.
00024 {
00025     CLI* w = new CLI;
00026     w->show();
00027     this->close();
00028 }
```

9.19.3.3 model

```
void MarkovPasswordsGUI::model ( ) [slot]
Definition at line 42 of file MarkovPasswordsGUI.cpp.
00042 {
00043     QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00044
00045     //get working directory
00046     char path[255];
00047     GetCurrentDirectoryA(255, path);
00048
00049     //get absolute path to the layout html
00050     std::string layout = "file:/// " + std::string(path) + "\\views\\index.html";
00051     std::replace(layout.begin(), layout.end(), '\\', '/');
00052     webkit->setUrl(QUrl(layout.c_str()));
00053 }
```

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI::_generate\(\)](#), [Python.Markopy.BaseCLI::_generate\(\)](#), [Python.Markopy.ModelMatrixCLI::_generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI::_generate\(\)](#), [Python.Markopy.BaseCLI::export\(\)](#), [Python.Markopy.BaseCLI::import_model\(\)](#), and [Python.Markopy.BaseCLI::train\(\)](#).

Here is the caller graph for this function:



9.19.3.4 pass

```
void MarkovPasswordsGUI::pass ( ) [slot]
Definition at line 29 of file MarkovPasswordsGUI.cpp.
00029 {
00030     QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00031
00032     //get working directory
00033     char path[255];
00034     GetCurrentDirectoryA(255, path);
00035
00036     //get absolute path to the layout html
00037     std::string layout = "file:/// " + std::string(path) + "\\views\\bar.html";
00038     std::replace(layout.begin(), layout.end(), '\\', '/');
00039     webkit->setUrl(QUrl(layout.c_str()));
00040 }
```

9.19.4 Member Data Documentation

9.19.4.1 ui

`Ui::MarkovPasswordsGUIClass` `Markov::GUI::MarkovPasswordsGUI::ui` [private]

Definition at line 25 of file [MarkovPasswordsGUI.h](#).

Referenced by [MarkovPasswordsGUI\(\)](#).

The documentation for this class was generated from the following files:

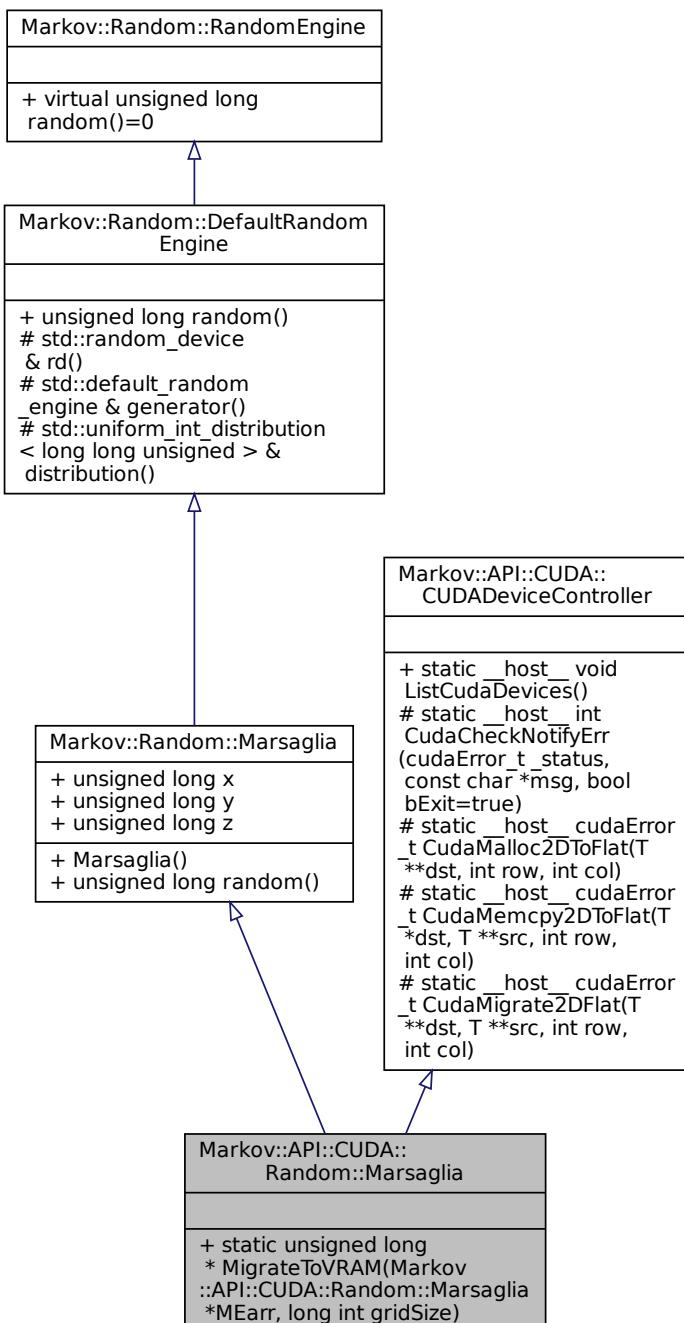
- [Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.h](#)
- [Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.cpp](#)

9.20 Markov::API::CUDA::Random::Marsaglia Class Reference

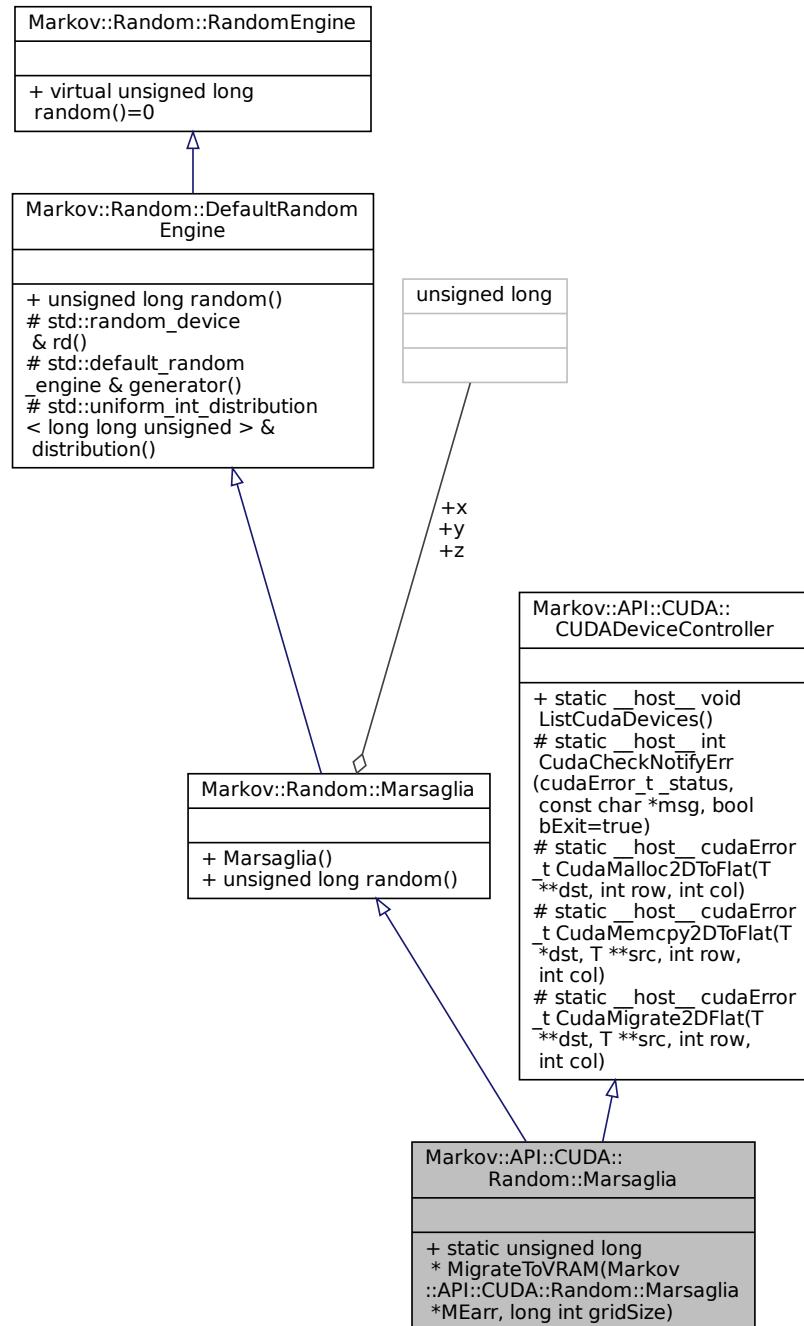
Extension of [Markov::Random::Marsaglia](#) which is capable of working on **device** space.

```
#include <cudarandom.h>
```

Inheritance diagram for Markov::API::CUDA::Random::Marsaglia:



Collaboration diagram for `Markov::API::CUDA::Random::Marsaglia`:



Public Member Functions

- `unsigned long random ()`

Generate Random Number.

Static Public Member Functions

- `static unsigned long * MigrateToVRAM (Markov::API::CUDA::Random::Marsaglia *MEarr, long int gridSize)`

Migrate a [Marsaglia](#)[] to VRAM as seedChunk.

- static __host__ void [ListCudaDevices](#) ()

List CUDA devices in the system.

Public Attributes

- unsigned long [x](#)
- unsigned long [y](#)
- unsigned long [z](#)

Protected Member Functions

- std::random_device & [rd](#) ()

Default random device for seeding.
- std::default_random_engine & [generator](#) ()

Default random engine for seeding.
- std::uniform_int_distribution< long long unsigned > & [distribution](#) ()

Distribution schema for seeding.

Static Protected Member Functions

- static __host__ int [CudaCheckNotifyErr](#) (cudaError_t _status, const char *msg, bool bExit=true)

Check results of the last operation on GPU.
- template<typename T >
 static __host__ cudaError_t [CudaMalloc2DToFlat](#) (T **dst, int row, int col)

Malloc a 2D array in device space.
- template<typename T >
 static __host__ cudaError_t [CudaMemcpy2DToFlat](#) (T *dst, T **src, int row, int col)

Memcpy a 2D array in device space after flattening.
- template<typename T >
 static __host__ cudaError_t [CudaMigrate2DFlat](#) (T **dst, T **src, int row, int col)

Both malloc and memcpy a 2D array into device VRAM.

9.20.1 Detailed Description

Extension of [Markov::Random::Marsaglia](#) which is capable of working on **device** space.

Implementation of [Marsaglia Random](#) Engine. This is an implementation of [Marsaglia Random](#) engine, which for most use cases is a better fit than other solutions. Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.

This implementation of the [Marsaglia](#) Engine is seeded by [random.h](#) default random engine. RandomEngine is only seeded once so its not a performance issue.

Example Use: Using [Marsaglia](#) Engine with RandomWalk

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Example Use: Generating a random number with [Marsaglia](#) Engine

```
Markov::Random::Marsaglia me;
std::cout << me.random();
Definition at line 20 of file cudarandom.h.
```

9.20.2 Member Function Documentation

9.20.2.1 CudaCheckNotifyErr()

```
__host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr (
    cudaError_t _status,
    const char * msg,
    bool bExit = true ) [static], [protected], [inherited]
```

Check results of the last operation on GPU.

Check the status returned from cudaMalloc/cudaMemcpy to find failures.

If a failure occurs, its assumed beyond redemption, and exited.

Parameters

_status	Cuda error status to check
msg	Message to print in case of a failure

Returns

0 if successful, 1 if failure. **Example output:**

```
char *da, a = "test";
cudaStatus = cudaMalloc((char **)&da, 5*sizeof(char *));
CudaCheckNotifyErr(cudaStatus, "Failed to allocate VRAM for *da.\n");
```

Definition at line 32 of file [cudaDeviceController.cu](#).

```
00032
00033     {
00034         if (_status != cudaSuccess) {
00035             std::cerr << "\033[1;31m" << msg << " -> " << cudaGetErrorString(_status) << "(" << _status <
00036             ") " << "\033[0m" << "\n";
00037         if(bExit) {
00038             cudaDeviceReset();
00039             exit(1);
00040         }
00041     }
00042 }
```

9.20.2.2 CudaMalloc2DToFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat (
    T ** dst,
    int row,
    int col ) [inline], [static], [protected], [inherited]
```

Malloc a 2D array in device space.

This function will allocate enough space on VRAM for flattened 2D array.

Parameters

dst	destination pointer
row	row size of the 2d array
col	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
}
```

Definition at line 75 of file [cudaDeviceController.h](#).

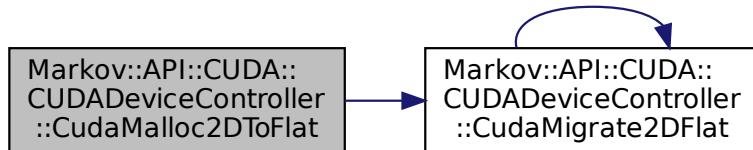
```

00075     cudaError_t cudastatus = cudaMalloc((T **)dst, row*col*sizeof(T));
00076     CudaCheckNotifyErr(cudastatus, "cudaMalloc Failed.", false);
00077     return cudastatus;
00078 }

```

References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:



9.20.2.3 CudaMemcpy2DToFlat()

```

template<typename T >
static __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat (
    T * dst,
    T ** src,
    int row,
    int col ) [inline], [static], [protected], [inherited]

```

Memcpy a 2D array in device space after flattening.

Resulting buffer will not be true 2D array.

Parameters

<i>dst</i>	destination pointer
<i>rc</i>	source pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

```

cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
CudaCheckNotifyErr(cudastatus, " CudaMemcpy2DToFlat Failed.", false);

```

Definition at line 103 of file [cudaDeviceController.h](#).

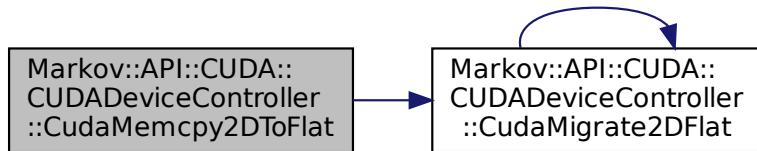
```

00103
00104     T* tempbuf = new T[row*col];
00105     for(int i=0;i<row;i++){
00106         memcpy(&(tempbuf[row*i]), src[i], col);
00107     }
00108     return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109 }
00110 }

```

References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:



9.20.2.4 CudaMigrate2DFlat()

```
template<typename T >
static __host__ cudaError_t Markov::API::CUDADeviceController::CudaMigrate2DFlat (
    T ** dst,
    T ** src,
    int row,
    int col ) [inline], [static], [protected], [inherited]
```

Both malloc and memcpy a 2D array into device VRAM.
Resulting buffer will not be true 2D array.

Parameters

<i>dst</i>	destination pointer
<i>rc</i>	source pointer
<i>row</i>	row size of the 2d array
<i>col</i>	column size of the 2d array

Returns

cudaError_t status of the cudaMalloc operation

Example output:

```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMigrate2DFlat<long int>(
    &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
```

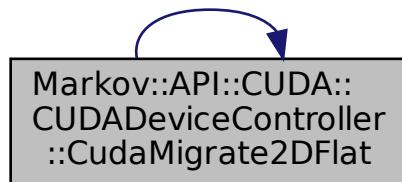
Definition at line 132 of file [cudaDeviceController.h](#).

```
00132
00133     cudaError_t cudastatus;
00134     cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135     if(cudastatus!=cudaSuccess){
00136         CudaCheckNotifyErr(cudastatus, "    CudaMalloc2DToFlat Failed.", false);
00137         return cudastatus;
00138     }
00139     cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140     CudaCheckNotifyErr(cudastatus, "    CudaMemcpy2DToFlat Failed.", false);
00141     return cudastatus;
00142 }
```

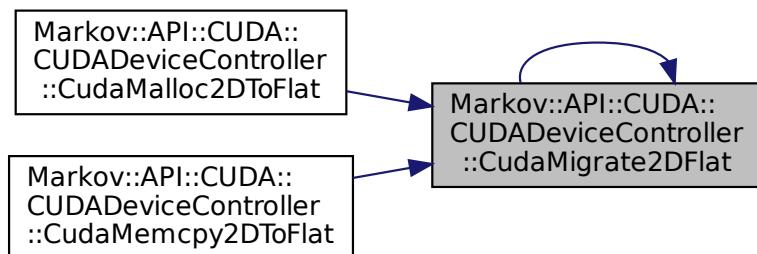
References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Referenced by [Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat\(\)](#), [Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat\(\)](#) and [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.20.2.5 distribution()

```
std::uniform_int_distribution<long long unsigned>& Markov::Random::DefaultRandomEngine::distribution()
() [inline], [protected], [inherited]
```

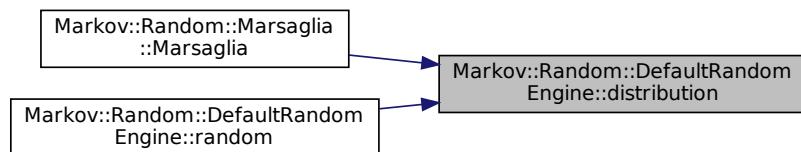
Distribution schema for seeding.

Definition at line 90 of file [random.h](#).

```
00090
00091     static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffffff);
00092     return _distribution;
00093 }
```

Referenced by [Markov::Random::Marsaglia::Marsaglia\(\)](#), and [Markov::Random::DefaultRandomEngine::random\(\)](#).

Here is the caller graph for this function:



9.20.2.6 generator()

```
std::default_random_engine& Markov::Random::DefaultRandomEngine::generator ( ) [inline],  
[protected], [inherited]
```

Default random engine for seeding.

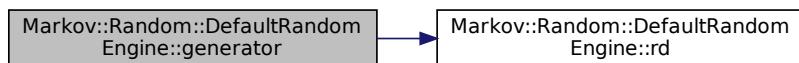
Definition at line 82 of file [random.h](#).

```
00082         static std::default_random_engine _generator(rd()());  
00083         return _generator;  
00084     }  
00085 }
```

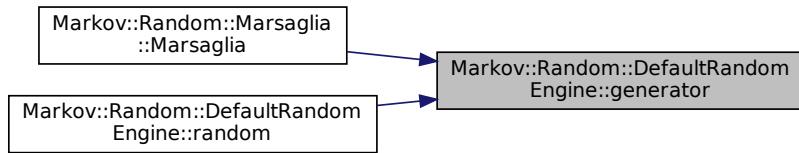
References [Markov::Random::DefaultRandomEngine::rd\(\)](#).

Referenced by [Markov::Random::Marsaglia::Marsaglia\(\)](#), and [Markov::Random::DefaultRandomEngine::random\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.20.2.7 ListCudaDevices()

```
__host__ void Markov::API::CUDA::CUDADeviceController::ListCudaDevices ( ) [static], [inherited]  
List CUDA devices in the system.
```

This function will print details of every CUDA capable device in the system.

Example output:

```
Device Number: 0  
Device name: GeForce RTX 2070  
Memory Clock Rate (KHz): 7001000  
Memory Bus Width (bits): 256  
Peak Memory Bandwidth (GB/s): 448.064  
Max Linear Threads: 1024
```

Definition at line 16 of file [cudaDeviceController.cu](#).

```
00016                                         { //list cuda Capable devices on  
00017     host.  
00018         int nDevices;  
00019         cudaGetDeviceCount(&nDevices);  
00020         for (int i = 0; i < nDevices; i++) {  
00021             cudaDeviceProp prop;  
00022             cudaGetDeviceProperties(&prop, i);  
00023             std::cerr << "Device Number: " << i << "\n";  
00024             std::cerr << "Device name: " << prop.name << "\n";  
00025             std::cerr << "Memory Clock Rate (KHz): " << prop.memoryClockRate << "\n";  
00026             std::cerr << "Memory Bus Width (bits): " << prop.memoryBusWidth << "\n";  
00027             std::cerr << "Peak Memory Bandwidth (GB/s): " << 2.0 * prop.memoryClockRate *  
00028             (prop.memoryBusWidth / 8) / 1.0e6 << "\n";  
00029             std::cerr << "Max Linear Threads: " << prop.maxThreadsDim[0] << "\n";  
00030         }
```

9.20.2.8 MigrateToVRAM()

```
static unsigned long* Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM (
    Markov::API::CUDA::Random::Marsaglia * MEarr,
    long int gridSize ) [inline], [static]
```

Migrate a [Marsaglia](#)[] to VRAM as seedChunk.

Parameters

<i>MEarr</i>	Array of Marsaglia Engines
<i>gridSize</i>	GridSize of the CUDA Kernel, aka size of array

Returns

pointer to the resulting seed chunk in device VRAM.

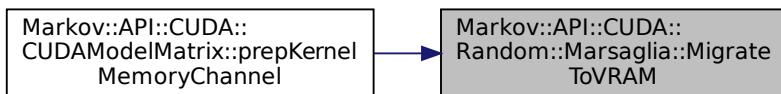
Definition at line 28 of file [cudarandom.h](#).

```
00028
00029     cudaError_t cudastatus;
00030     unsigned long* seedChunk;
00031     cudastatus = cudaMalloc((unsigned long**)&seedChunk, gridSize*3*sizeof(unsigned long));
00032     CudaCheckNotifyErr(cudastatus, "Failed to allocate seed buffer");
00033     unsigned long *temp = new unsigned long[gridSize*3];
00034     for(int i=0;i<gridSize;i++){
00035         temp[i*3] = MEarr[i].x;
00036         temp[i*3+1] = MEarr[i].y;
00037         temp[i*3+2] = MEarr[i].z;
00038     }
00039     //for(int i=0;i<gridSize*3;i++) std::cout << temp[i] << "\n";
00040     cudaMemcpy(seedChunk, temp, gridSize*3*sizeof(unsigned long), cudaMemcpyHostToDevice);
00041     CudaCheckNotifyErr(cudastatus, "Failed to memcpy seed buffer.");
00042
00043     delete[] temp;
00044     return seedChunk;
00045 }
```

References [Markov::Random::Marsaglia::x](#), [Markov::Random::Marsaglia::y](#), and [Markov::Random::Marsaglia::z](#).

Referenced by [Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel\(\)](#).

Here is the caller graph for this function:



9.20.2.9 random()

```
unsigned long Markov::Random::Marsaglia::random () [inline], [virtual], [inherited]
Generate Random Number.
```

Returns

random number in long range.

Reimplemented from [Markov::Random::DefaultRandomEngine](#).

Definition at line 140 of file [random.h](#).

```
00140
00141     unsigned long t;
00142     x ^= x << 16;
00143     x ^= x >> 5;
00144     x ^= x << 1;
00145
00146     t = x;
```

```

00147     x = y;
00148     y = z;
00149     z = t ^ x ^ y;
00150
00151     return z;
00152 }
```

References [Markov::Random::Marsaglia::x](#), [Markov::Random::Marsaglia::y](#), and [Markov::Random::Marsaglia::z](#).
Referenced by [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Here is the caller graph for this function:



9.20.2.10 rd()

```
std::random_device& Markov::Random::DefaultRandomEngine::rd () [inline], [protected], [inherited]
```

Default random device for seeding.

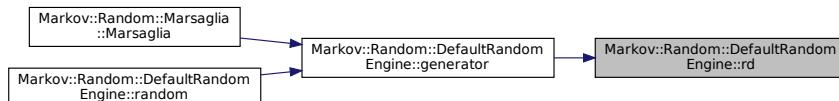
Definition at line 74 of file [random.h](#).

```

00074
00075     static std::random_device _rd;
00076     return _rd;
00077 }
```

Referenced by [Markov::Random::DefaultRandomEngine::generator\(\)](#).

Here is the caller graph for this function:



9.20.3 Member Data Documentation

9.20.3.1 x

```
unsigned long Markov::Random::Marsaglia::x [inherited]
```

Definition at line 155 of file [random.h](#).

Referenced by [Markov::Random::Marsaglia::Marsaglia\(\)](#), [MigrateToVRAM\(\)](#), and [Markov::Random::Marsaglia::random\(\)](#).

9.20.3.2 y

```
unsigned long Markov::Random::Marsaglia::y [inherited]
```

Definition at line 156 of file [random.h](#).

Referenced by [Markov::Random::Marsaglia::Marsaglia\(\)](#), [MigrateToVRAM\(\)](#), and [Markov::Random::Marsaglia::random\(\)](#).

9.20.3.3 z

```
unsigned long Markov::Random::Marsaglia::z [inherited]
```

Definition at line 157 of file [random.h](#).

Referenced by [Markov::Random::Marsaglia::Marsaglia\(\)](#), [MigrateToVRAM\(\)](#), and [Markov::Random::Marsaglia::random\(\)](#).

The documentation for this class was generated from the following file:

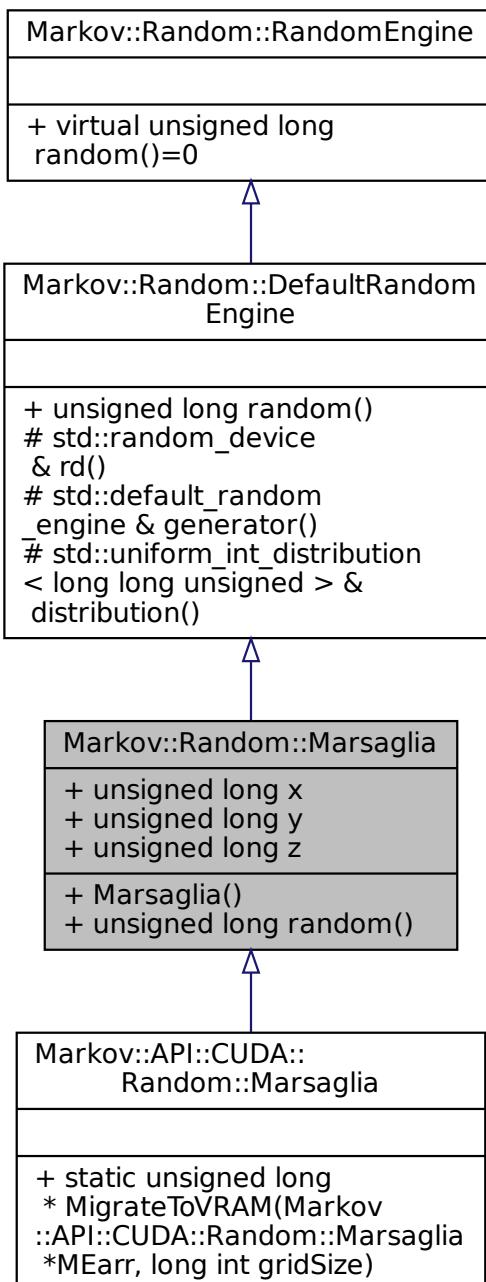
- [Markopy/CudaMarkovAPI/src/cudarandom.h](#)

9.21 Markov::Random::Marsaglia Class Reference

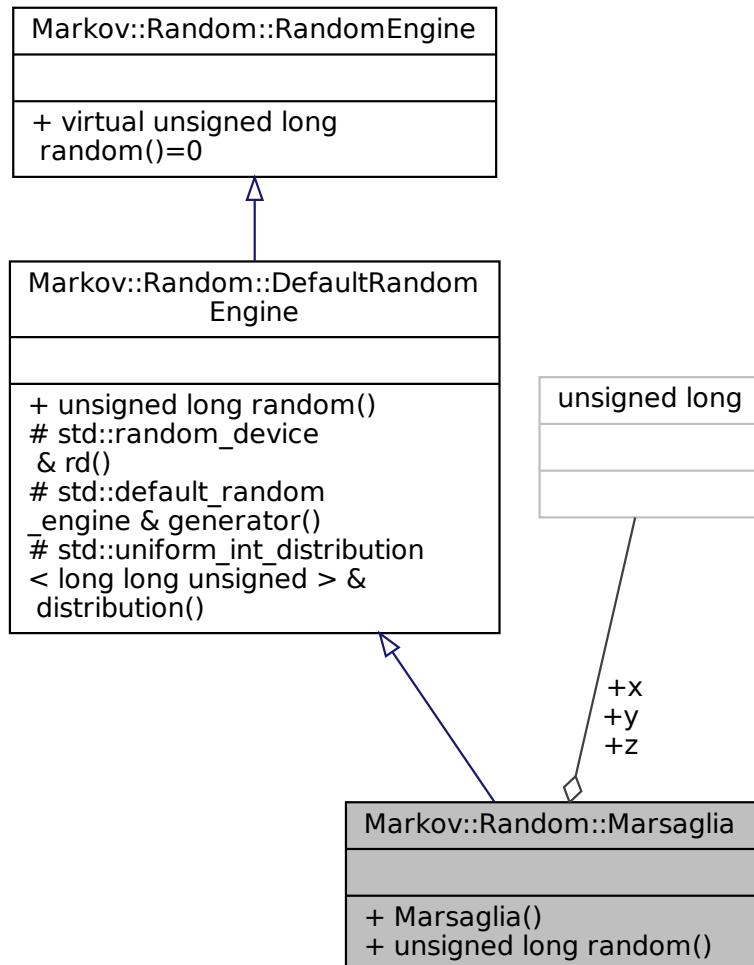
Implementation of [Marsaglia Random Engine](#).

```
#include <random.h>
```

Inheritance diagram for Markov::Random::Marsaglia:



Collaboration diagram for `Markov::Random::Marsaglia`:



Public Member Functions

- `Marsaglia ()`
Construct `Marsaglia` Engine.
- `unsigned long random ()`
Generate `Random` Number.

Public Attributes

- `unsigned long x`
- `unsigned long y`
- `unsigned long z`

Protected Member Functions

- `std::random_device & rd ()`
Default random device for seeding.

- std::default_random_engine & **generator** ()

Default random engine for seeding.
- std::uniform_int_distribution< long long unsigned > & **distribution** ()

Distribution schema for seeding.

9.21.1 Detailed Description

Implementation of **Marsaglia Random** Engine.

This is an implementation of **Marsaglia Random** engine, which for most use cases is a better fit than other solutions. Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.

This implementation of the **Marsaglia** Engine is seeded by **random.h** default random engine. **RandomEngine** is only seeded once so its not a performance issue.

Example Use: Using **Marsaglia** Engine with **RandomWalk**

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Example Use: Generating a random number with **Marsaglia** Engine

```
Markov::Random::Marsaglia me;
std::cout << me.random();
```

Definition at line 125 of file **random.h**.

9.21.2 Constructor & Destructor Documentation

9.21.2.1 Marsaglia()

Markov::Random::Marsaglia::Marsaglia () [inline]

Construct **Marsaglia** Engine.

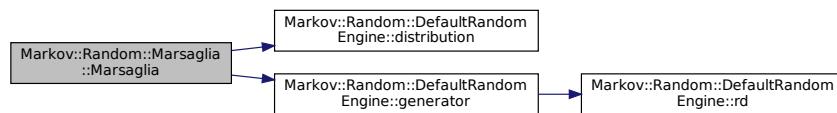
Initialize x,y and z using the default random engine.

Definition at line 132 of file **random.h**.

```
00132     {
00133         this->x = this->distribution()(this->generator());
00134         this->y = this->distribution()(this->generator());
00135         this->z = this->distribution()(this->generator());
00136         //std::cout << "x: " << x << ", y: " << y << ", z: " << z << "\n";
00137     }
```

References **Markov::Random::DefaultRandomEngine::distribution()**, **Markov::Random::DefaultRandomEngine::generator()**, **x**, **y**, and **z**.

Here is the call graph for this function:



9.21.3 Member Function Documentation

9.21.3.1 distribution()

```
std::uniform_int_distribution<long long unsigned> & Markov::Random::DefaultRandomEngine::distribution()
( ) [inline], [protected], [inherited]
```

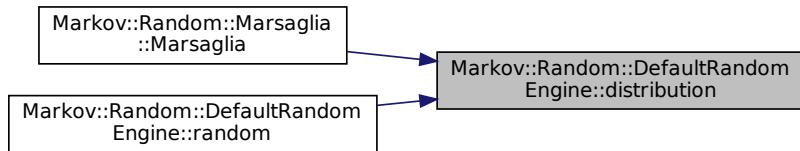
Distribution schema for seeding.

Definition at line 90 of file random.h.

```
00090     static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffffff);
00091     return _distribution;
00092 }
00093 }
```

Referenced by [Marsaglia\(\)](#), and [Markov::Random::DefaultRandomEngine::random\(\)](#).

Here is the caller graph for this function:



9.21.3.2 generator()

`std::default_random_engine& Markov::Random::DefaultRandomEngine::generator () [inline], [protected], [inherited]`

Default random engine for seeding.

Definition at line 82 of file random.h.

```
00082 {
00083     static std::default_random_engine _generator(rnd()());
00084     return _generator;
00085 }
```

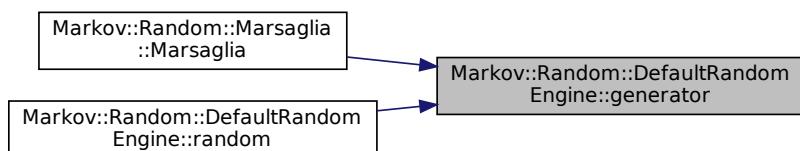
References [Markov::Random::DefaultRandomEngine::rd\(\)](#).

Referenced by [Marsaglia\(\)](#), and [Markov::Random::DefaultRandomEngine::random\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.21.3.3 random()

`unsigned long Markov::Random::Marsaglia::random () [inline], [virtual]`

Generate Random Number.

Returns

random number in long range.

Reimplemented from [Markov::Random::DefaultRandomEngine](#).

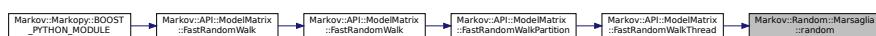
Definition at line 140 of file [random.h](#).

```
00140         unsigned long t;
00141         x ^= x << 16;
00142         x ^= x >> 5;
00143         x ^= x << 1;
00145
00146         t = x;
00147         x = y;
00148         y = z;
00149         z = t ^ x ^ y;
00150
00151     return z;
00152 }
```

References [x](#), [y](#), and [z](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Here is the caller graph for this function:

**9.21.3.4 rd()**

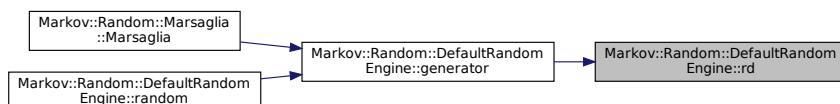
`std::random_device& Markov::Random::DefaultRandomEngine::rd () [inline], [protected], [inherited]`
Default random device for seeding.

Definition at line 74 of file [random.h](#).

```
00074     static std::random_device _rd;
00075     return _rd;
00076 }
00077 }
```

Referenced by [Markov::Random::DefaultRandomEngine::generator\(\)](#).

Here is the caller graph for this function:

**9.21.4 Member Data Documentation****9.21.4.1 x**

`unsigned long Markov::Random::Marsaglia::x`

Definition at line 155 of file [random.h](#).

Referenced by [Marsaglia\(\)](#), [Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM\(\)](#), and [random\(\)](#).

9.21.4.2 y

`unsigned long Markov::Random::Marsaglia::y`

Definition at line 156 of file [random.h](#).

Referenced by [Marsaglia\(\)](#), [Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM\(\)](#), and [random\(\)](#).

9.21.4.3 z

```
unsigned long Markov::Random::Marsaglia::z
```

Definition at line 157 of file [random.h](#).

Referenced by [Marsaglia\(\)](#), [Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM\(\)](#), and [random\(\)](#).

The documentation for this class was generated from the following file:

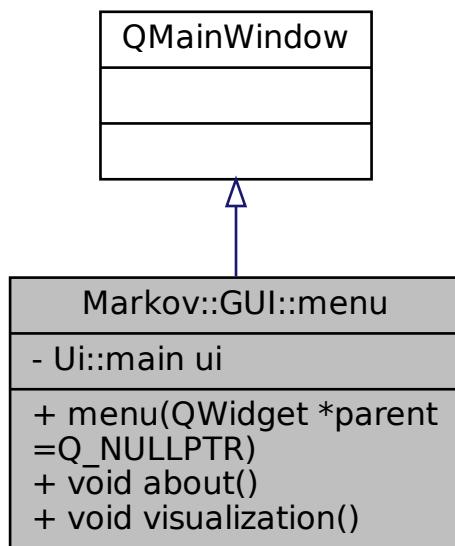
- [Markopy/MarkovModel/src/random.h](#)

9.22 Markov::GUI::menu Class Reference

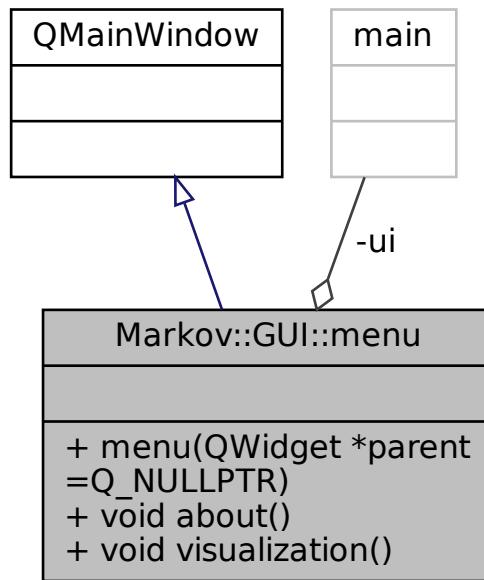
QT Menu class.

```
#include <menu.h>
```

Inheritance diagram for Markov::GUI::menu:



Collaboration diagram for Markov::GUI::menu:



Public Slots

- void [about \(\)](#)
- void [visualization \(\)](#)

Public Member Functions

- [menu \(QWidget *parent=Q_NULLPTR\)](#)

Private Attributes

- [Ui::main ui](#)

9.22.1 Detailed Description

QT Menu class.

Definition at line 15 of file [menu.h](#).

9.22.2 Constructor & Destructor Documentation

9.22.2.1 menu()

```

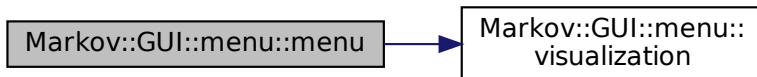
menu::menu (
    QWidget * parent = Q_NULLPTR )
Definition at line 14 of file menu.cpp.
00015     : QMainWidget(parent)
00016 {
00017     ui.setupUi(this);
00018
  
```

```

00019
00020     //QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {about(); });
00021     QObject::connect(ui.visu, &QPushButton::clicked, this, [this] {visualization(); });
00022 }
```

References [ui](#), and [visualization\(\)](#).

Here is the call graph for this function:



9.22.3 Member Function Documentation

9.22.3.1 about

```

void menu::about ( ) [slot]
Definition at line 23 of file menu.cpp.
00023
00024
00025
00026 }
```

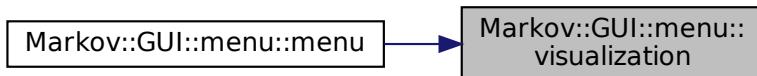
9.22.3.2 visualization

```

void menu::visualization ( ) [slot]
Definition at line 27 of file menu.cpp.
00027
00028     MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
00029     w->show();
00030     this->close();
00031 }
```

Referenced by [menu\(\)](#).

Here is the caller graph for this function:



9.22.4 Member Data Documentation

9.22.4.1 ui

[Ui::main](#) `Markov::GUI::menu::ui` [private]

Definition at line 21 of file [menu.h](#).

Referenced by [menu\(\)](#).

The documentation for this class was generated from the following files:

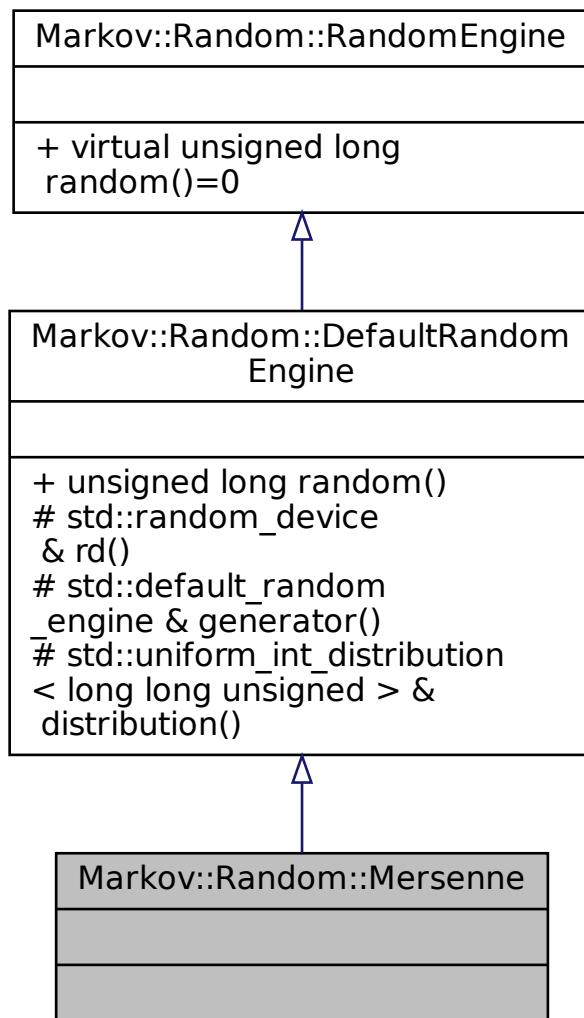
- Markopy/MarkovPasswordsGUI/src/menu.h
- Markopy/MarkovPasswordsGUI/src/menu.cpp

9.23 Markov::Random::Mersenne Class Reference

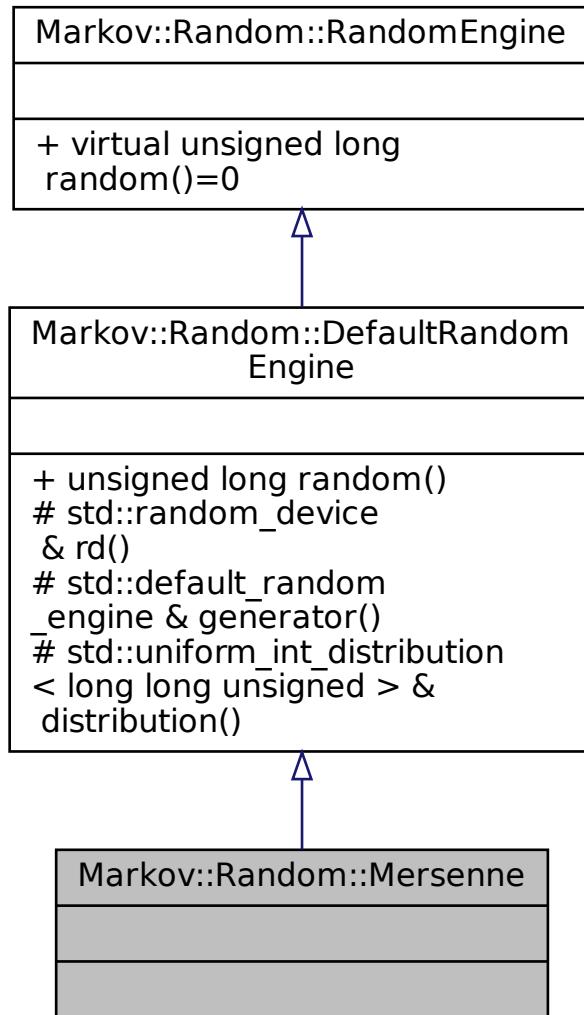
Implementation of Mersenne Twister Engine.

```
#include <random.h>
```

Inheritance diagram for Markov::Random::Mersenne:



Collaboration diagram for Markov::Random::Mersenne:



Public Member Functions

- `unsigned long random ()`

Generate Random Number.

Protected Member Functions

- `std::random_device & rd ()`
Default random device for seeding.
- `std::default_random_engine & generator ()`
Default random engine for seeding.
- `std::uniform_int_distribution< long long unsigned > & distribution ()`
Distribution schema for seeding.

9.23.1 Detailed Description

Implementation of [Mersenne](#) Twister Engine.

This is an implementation of [Mersenne](#) Twister Engine, which is slow but is a good implementation for high entropy pseudorandom.

Example Use: Using [Mersenne](#) Engine with RandomWalk

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Mersenne MersenneTwisterEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MersenneTwisterEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Example Use: Generating a random number with [Marsaglia](#) Engine

```
Markov::Random::Mersenne me;
std::cout << me.random();
```

Definition at line 185 of file [random.h](#).

9.23.2 Member Function Documentation

9.23.2.1 distribution()

```
std::uniform_int_distribution<long long unsigned>& Markov::Random::DefaultRandomEngine::distribution
( ) [inline], [protected], [inherited]
```

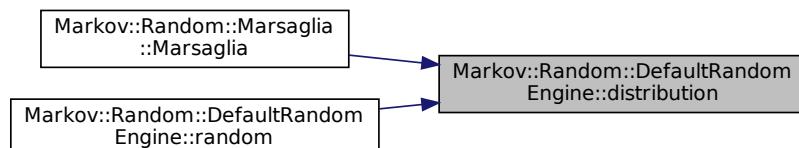
Distribution schema for seeding.

Definition at line 90 of file [random.h](#).

```
00090
00091     static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffffff);
00092     return _distribution;
00093 }
```

Referenced by [Markov::Random::Marsaglia::Marsaglia\(\)](#), and [Markov::Random::DefaultRandomEngine::random\(\)](#).

Here is the caller graph for this function:



9.23.2.2 generator()

```
std::default_random_engine& Markov::Random::DefaultRandomEngine::generator ( ) [inline],
[protected], [inherited]
```

Default random engine for seeding.

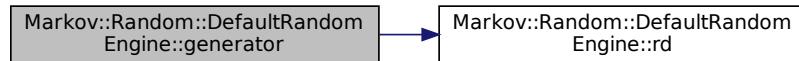
Definition at line 82 of file [random.h](#).

```
00082
00083     static std::default_random_engine _generator(rd()());
00084     return _generator;
00085 }
```

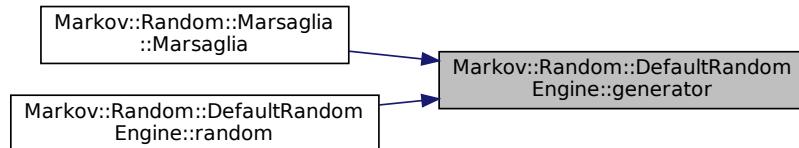
References [Markov::Random::DefaultRandomEngine::rd\(\)](#).

Referenced by [Markov::Random::Marsaglia::Marsaglia\(\)](#), and [Markov::Random::DefaultRandomEngine::random\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.23.2.3 random()

`unsigned long Markov::Random::DefaultRandomEngine::random () [inline], [virtual], [inherited]`
Generate Random Number.

Returns

random number in long range.

Implements [Markov::Random::RandomEngine](#).

Reimplemented in [Markov::Random::Marsaglia](#).

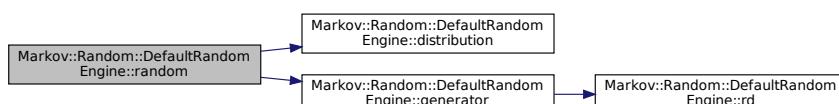
Definition at line 66 of file [random.h](#).

```

00066
00067     return this->distribution() (this->generator());
00068 }
```

References [Markov::Random::DefaultRandomEngine::distribution\(\)](#), and [Markov::Random::DefaultRandomEngine::generator\(\)](#).

Here is the call graph for this function:



9.23.2.4 rd()

`std::random_device& Markov::Random::DefaultRandomEngine::rd () [inline], [protected], [inherited]`
Default random device for seeding.

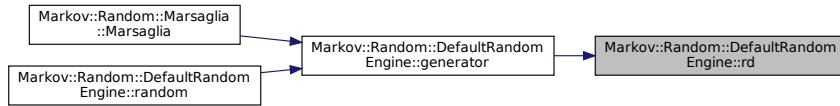
Definition at line 74 of file [random.h](#).

```

00074
00075     static std::random_device _rd;
00076     return _rd;
00077 }
```

Referenced by [Markov::Random::DefaultRandomEngine::generator\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

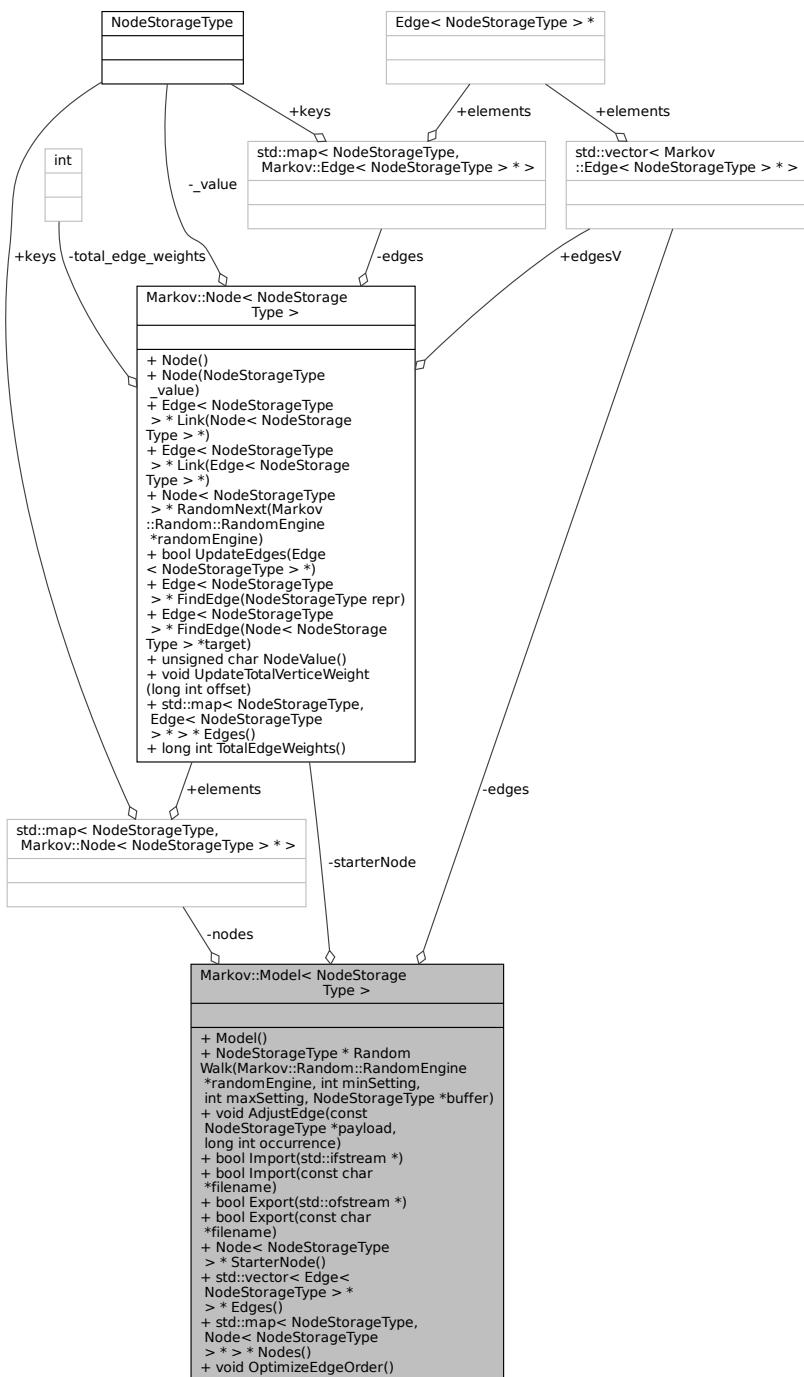
- [Markopy/MarkovModel/src/random.h](#)

9.24 **Markov::Model< NodeStorageType >** Class Template Reference

class for the final [Markov Model](#), constructed from nodes and edges.

```
#include <model.h>
```

Collaboration diagram for `Markov::Model< NodeStorageType >`:



Public Member Functions

- `Model()`
Initialize a model with only start and end nodes.
- `NodeStorageType * RandomWalk (Markov::Random::RandomEngine *randomEngine, int minSetting, int maxSetting, NodeStorageType *buffer)`
Do a random walk on this model.
- `void AdjustEdge (const NodeStorageType *payload, long int occurrence)`

- **bool Import (std::ifstream *)**
Import a file to construct the model.
- **bool Import (const char *filename)**
Open a file to import with filename, and call bool Model::Import with std::ifstream.
- **bool Export (std::ofstream *)**
Export a file of the model.
- **bool Export (const char *filename)**
Open a file to export with filename, and call bool Model::Export with std::ofstream.
- **Node< NodeStorageType > * StarterNode ()**
Return starter Node.
- **std::vector< Edge< NodeStorageType > * > * Edges ()**
Return a vector of all the edges in the model.
- **std::map< NodeStorageType, Node< NodeStorageType > * > * Nodes ()**
Return starter Node.
- **void OptimizeEdgeOrder ()**
Sort edges of all nodes in the model ordered by edge weights.

Private Attributes

- **std::map< NodeStorageType, Node< NodeStorageType > * > nodes**
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- **Node< NodeStorageType > * starterNode**
Starter Node of this model.
- **std::vector< Edge< NodeStorageType > * > edges**
A list of all edges in this model.

9.24.1 Detailed Description

```
template<typename NodeStorageType>
class Markov::Model< NodeStorageType >
```

class for the final **Markov Model**, constructed from nodes and edges.

Each atomic piece of the generation result is stored in a node, while edges contain the relation weights. *Extending:* To extend the class, implement the template and inherit from it, as "class MyModel : public Markov::Model<char>". For a complete demonstration of how to extend the class, see [MarkovPasswords](#).

Whole model can be defined as a list of the edges, as dangling nodes are pointless. This approach is used for the import/export operations. For more information on importing/exporting model, check out the [github readme](#) and [wiki page](#).

Definition at line 45 of file [model.h](#).

9.24.2 Constructor & Destructor Documentation

9.24.2.1 Model()

```
template<typename NodeStorageType >
Markov::Model< NodeStorageType >::Model
Initialize a model with only start and end nodes.

Initialize an empty model with only a starterNode Starter node is a special kind of node that has constant 0x00 value, and will be used to initiate the generation execution from.

Definition at line 210 of file model.h.
00210
00211     this->starterNode = new Markov::Node<NodeStorageType>(0);
00212     this->nodes.insert({ 0, this->starterNode });
00213 }
```

9.24.3 Member Function Documentation

9.24.3.1 AdjustEdge()

```
template<typename NodeStorageType >
void Markov::Model< NodeStorageType >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence )
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

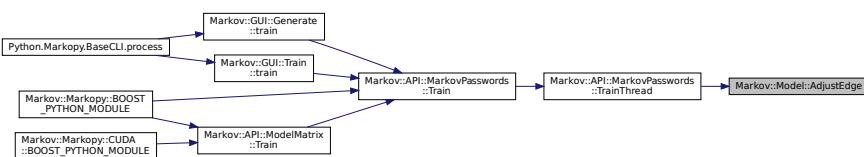
<i>string</i>	- String that is passed from the training, and will be used to AdjustEdge the model with
<i>occurrence</i>	- Occurrence of this string.

Definition at line 337 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

Referenced by [Markov::API::MarkovPasswords::TrainThread\(\)](#).

Here is the caller graph for this function:



9.24.3.2 Edges()

```
template<typename NodeStorageType >
std::vector<Edge<NodeStorageType>*>* Markov::Model< NodeStorageType >::Edges ( ) [inline]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.24.3.3 Export() [1/2]

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Export (
    const char * filename )
```

Open a file to export with filename, and call bool [Model::Export](#) with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

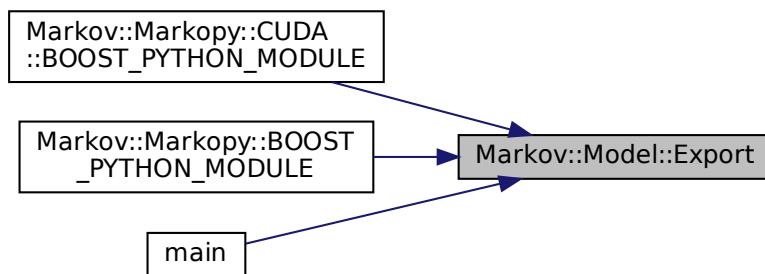
```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 300 of file [model.h](#).

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [main\(\)](#).

Here is the caller graph for this function:

**9.24.3.4 Export() [2/2]**

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Export (
    std::ofstream * f )
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ostream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);

Definition at line 288 of file model.h.
00288     {
00289         Markov::Edge<NodeStorageType>* e;
00290         for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291             e = this->edges[i];
00292             //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <<
00293             e->RightNode()->NodeValue() << "\n";
00294             *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
00295             "\n";
00296         }
00297     }
00298     return true;
00299 }
```

Referenced by [Markov::API::MarkovPasswords::Save\(\)](#).

Here is the caller graph for this function:

**9.24.3.5 Import() [1/2]**

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Import (
    const char * filename )
```

Open a file to import with filename, and call bool [Model::Import](#) with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

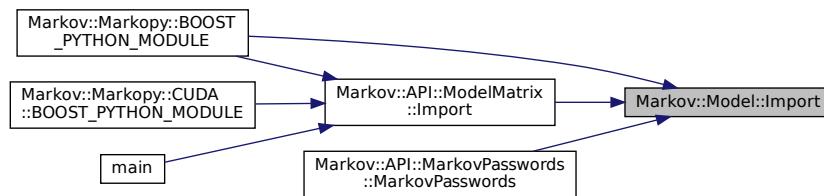
```
Markov::Model<char> model;
model.Import("test.mdl");
```

Definition at line 280 of file model.h.

```
00280     {
00281         std::ifstream importfile;
00282         importfile.open(filename);
00283         return this->Import(&importfile);
00284     }
00285 }
```

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::MarkovPasswords::MarkovPasswords\(\)](#).

Here is the caller graph for this function:



9.24.3.6 Import() [2/2]

```
template<typename NodeStorageType >
bool Markov::Model< NodeStorageType >::Import (
    std::ifstream * f )
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight:right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 216 of file model.h.

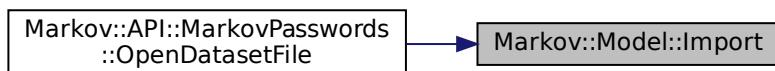
```
00216     std::string cell;                                {
00217     std::string src;                                 src;
00218     std::string target;                            target;
00219     long int oc;                                oc;
00220
00221     while (std::getline(*f, cell)) {
00222         //std::cout << "cell: " << cell << std::endl;
00223         src = cell[0];
00224         target = cell[cell.length() - 1];
00225         char* j;
00226         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00227         //std::cout << oc << "\n";
00228         Markov::Node<NodeStorageType>* srcN;
00229         Markov::Node<NodeStorageType>* targetN;
00230         Markov::Edge<NodeStorageType>* e;
00231         if (this->nodes.find(src) == this->nodes.end()) {
00232             srcN = new Markov::Node<NodeStorageType>(src);
00233             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00234             //std::cout << "Creating new node at start.\n";
00235         }
00236         else {
00237             srcN = this->nodes.find(src)->second;
00238         }
00239
00240         if (this->nodes.find(target) == this->nodes.end()) {
00241             targetN = new Markov::Node<NodeStorageType>(target);
00242             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00243             //std::cout << "Creating new node at end.\n";
00244         }
00245         else {
00246             targetN = this->nodes.find(target)->second;
00247         }
00248         e = srcN->Link(targetN);
00249         e->AdjustEdge(oc);
00250         this->edges.push_back(e);
00251
00252         //std::cout << int(srcN->NodeValue()) << " --> " <<
00253         //int(targetN->NodeValue()) << "\n";
```

```

00255
00256
00257 }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

Referenced by [Markov::API::MarkovPasswords::OpenDatasetFile\(\)](#).

Here is the caller graph for this function:



9.24.3.7 Nodes()

```
template<typename NodeStorageType >
std::map<NodeStorageType, Node<NodeStorageType>*>* Markov::Model< NodeStorageType >::Nodes (
) [inline]
```

Return starter Node.

Returns

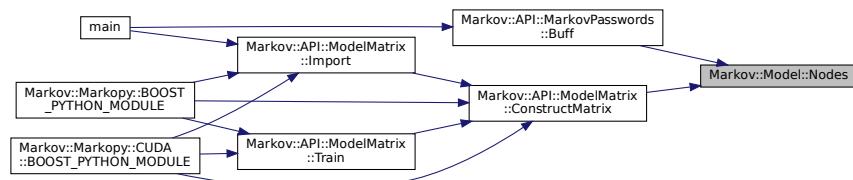
starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

Referenced by [Markov::API::MarkovPasswords::Buff\(\)](#), and [Markov::API::ModelMatrix::ConstructMatrix\(\)](#).

Here is the caller graph for this function:



9.24.3.8 OptimizeEdgeOrder()

```
template<typename NodeStorageType >
void Markov::Model< NodeStorageType >::OptimizeEdgeOrder
```

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 265 of file [model.h](#).

```

00265
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>*> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
00269             Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight() > rhs->EdgeWeight();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";

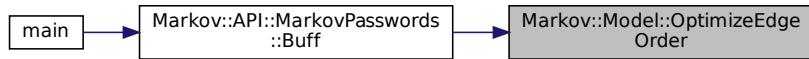
```

```

00273     //std::cout << "\n";
00274 }
00275 //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00276 //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00277 }
```

Referenced by [Markov::API::MarkovPasswords::Buff\(\)](#).

Here is the caller graph for this function:



9.24.3.9 RandomWalk()

```

template<typename NodeStorageType >
NodeStorageType * Markov::Model< NodeStorageType >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer )
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```

Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 307 of file [model.h](#).

```

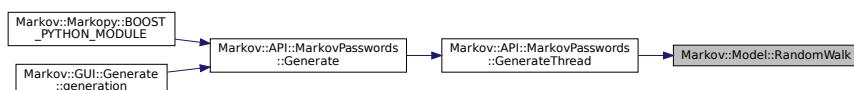
00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
```

```

00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL) {
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

Referenced by [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Here is the caller graph for this function:



9.24.3.10 StarterNode()

```

template<typename NodeStorageType >
Node< NodeStorageType >* Markov::Model< NodeStorageType >::StarterNode ( ) [inline]
Return starter Node.
```

Returns

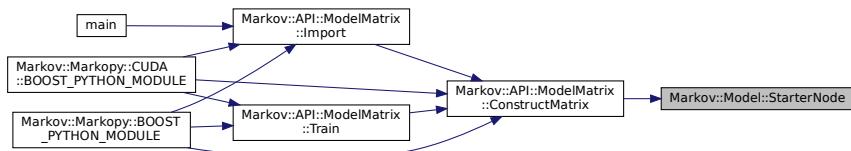
starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#).

Here is the caller graph for this function:



9.24.4 Member Data Documentation

9.24.4.1 edges

```

template<typename NodeStorageType >
std::vector<Edge<NodeStorageType>*>* Markov::Model< NodeStorageType >::edges [private]
```

A list of all edges in this model.

Definition at line 204 of file [model.h](#).

Referenced by [Markov::Model< char >::Edges\(\)](#).

9.24.4.2 nodes

```
template<typename NodeStorageType >
std::map<NodeStorageType, Node<NodeStorageType>*>> Markov::Model< NodeStorageType >::nodes
[private]
```

Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file [model.h](#).
Referenced by [Markov::Model< char >::Nodes\(\)](#).

9.24.4.3 starterNode

```
template<typename NodeStorageType >
Node<NodeStorageType>* Markov::Model< NodeStorageType >::starterNode [private]
```

Starter Node of this model.
Definition at line 198 of file [model.h](#).
Referenced by [Markov::Model< char >::StarterNode\(\)](#).
The documentation for this class was generated from the following file:

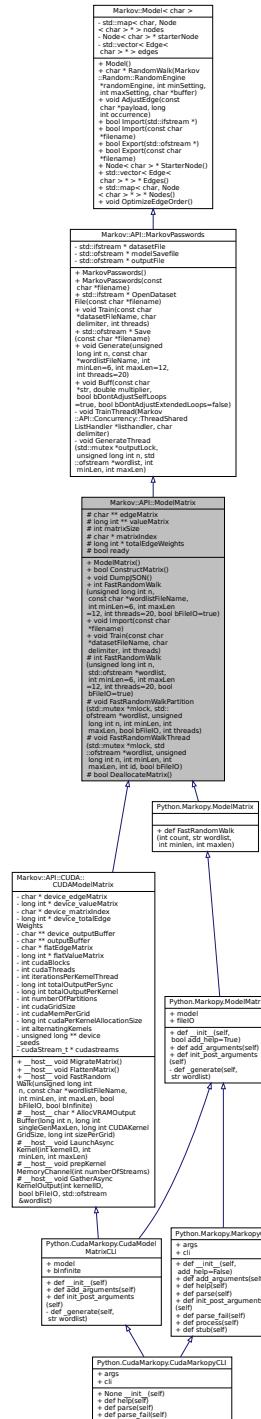
- [Markopy/MarkovModel/src/model.h](#)

9.25 **Markov::API::ModelMatrix** Class Reference

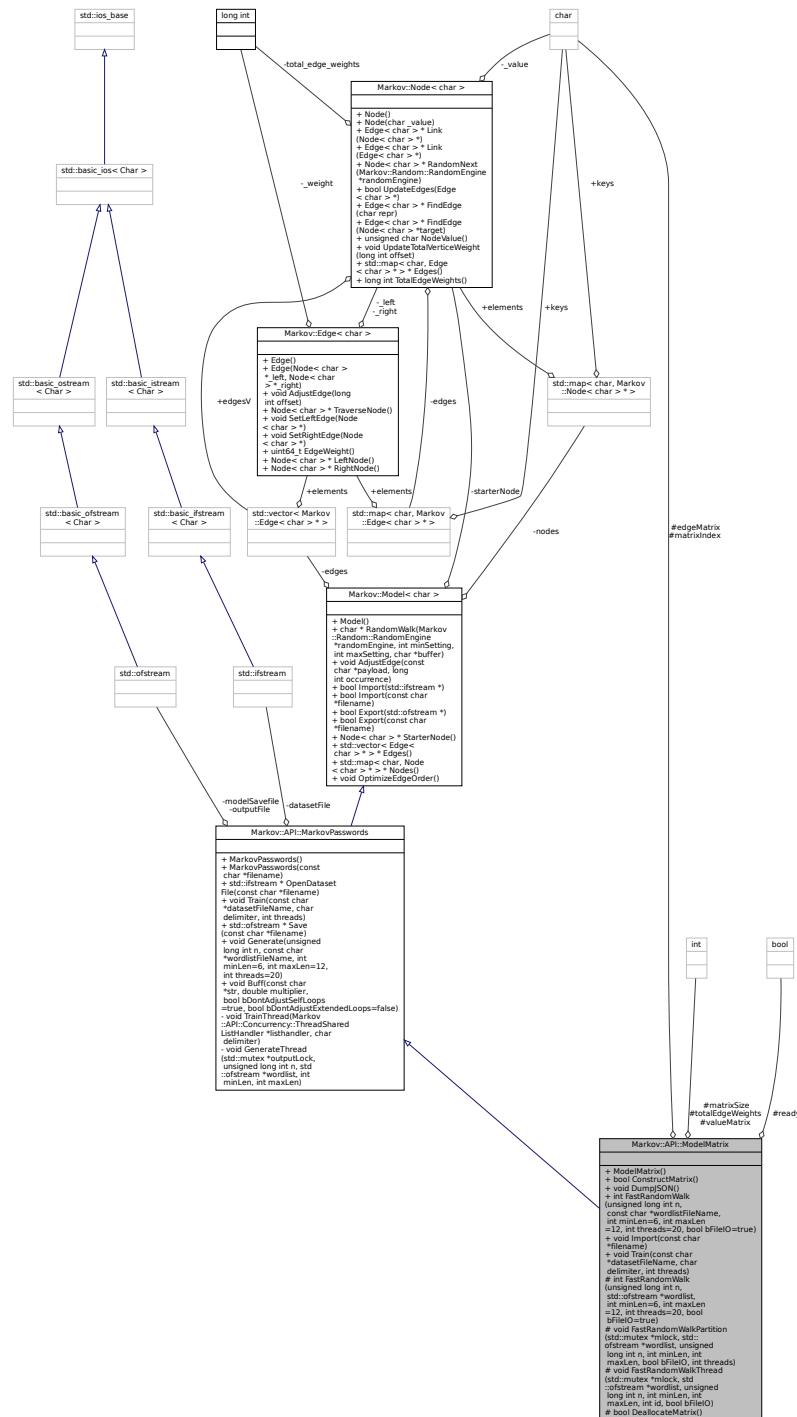
Class to flatten and reduce [Markov::Model](#) to a Matrix.

```
#include <modelMatrix.h>
```

Inheritance diagram for Markov::API::ModelMatrix



Collaboration diagram for Markov::API::ModelMatrix:



Public Member Functions

- `ModelMatrix ()`
 - `bool ConstructMatrix ()`

Construct the related Matrix data for the model.

- void DumpJSON ()

Debug function to dump the model to a JSON file.

- int **FastRandomWalk** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced `Markov::Model`.
- void **Import** (const char *filename)

Open a file to import with filename, and call `bool Model::Import` with `std::ifstream`.
- void **Train** (const char *datasetFileName, char delimiter, int threads)

Train the model with the dataset file.
- std::ifstream * **OpenDatasetFile** (const char *filename)

Open dataset file and return the ifstream pointer.
- std::ofstream * **Save** (const char *filename)

Export model to file.
- void **Generate** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

Call `Markov::Model::RandomWalk` n times, and collect output.
- void **Buff** (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)

Buff expression of some characters in the model.
- char * **RandomWalk** (`Markov::Random::RandomEngine` *randomEngine, int minSetting, int maxSetting, char *buffer)

Do a random walk on this model.
- void **AdjustEdge** (const char *payload, long int occurrence)

Adjust the model with a single string.
- bool **Import** (std::ifstream *)

Import a file to construct the model.
- bool **Export** (std::ofstream *)

Export a file of the model.
- bool **Export** (const char *filename)

Open a file to export with filename, and call `bool Model::Export` with `std::ofstream`.
- **Node<char> * StarterNode()**

Return starter Node.
- std::vector<**Edge<char>**> * * **Edges()**

Return a vector of all the edges in the model.
- std::map<char, **Node<char>**> * * **Nodes()**

Return starter Node.
- void **OptimizeEdgeOrder()**

Sort edges of all nodes in the model ordered by edge weights.

Protected Member Functions

- int **FastRandomWalk** (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced `Markov::Model`.
- void **FastRandomWalkPartition** (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

A single partition of `FastRandomWalk` event.
- void **FastRandomWalkThread** (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)

A single thread of a single partition of `FastRandomWalk`.
- bool **DeallocateMatrix()**

Deallocate matrix and make it ready for re-construction.

Protected Attributes

- char ** **edgeMatrix**
2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** **valueMatrix**
2-d Integer array for the value Matrix (For the weights of Edges)
- int **matrixSize**
to hold Matrix size
- char * **matrixIndex**
to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * **totalEdgeWeights**
Array of the Total Edge Weights.
- bool **ready**
True when matrix is constructed. False if not.

Private Member Functions

- void **TrainThread** (Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char delimiter)
A single thread invoked by the Train function.
- void **GenerateThread** (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)
A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * **datasetFile**
- std::ofstream * **modelSavefile**
Dataset file input of our system
- std::ofstream * **outputFile**
File to save model of our system
- std::map< char, Node< char > * > **nodes**
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > * **starterNode**
Starter Node of this model.
- std::vector< Edge< char > * > **edges**
A list of all edges in this model.

9.25.1 Detailed Description

Class to flatten and reduce [Markov::Model](#) to a Matrix.

Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)

To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are synchronized and files are flushed every 50M operations.

Definition at line [23](#) of file [modelMatrix.h](#).

9.25.2 Constructor & Destructor Documentation

9.25.2.1 ModelMatrix()

```
Markov::API::ModelMatrix::ModelMatrix ( )
Definition at line 15 of file modelMatrix.cpp.
00015     {
00016     this->ready = false;
00017 }
```

References ready.

9.25.3 Member Function Documentation

9.25.3.1 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file model.h.

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.25.3.2 Buff()

```
void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
-----	---

Parameters

<i>multiplier</i>	A constant value to buff the nodes with.
<i>bDontAdjustSelfEdges</i>	Do not adjust weights if target node is same as source node
<i>bDontAdjustExtendedLoops</i>	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

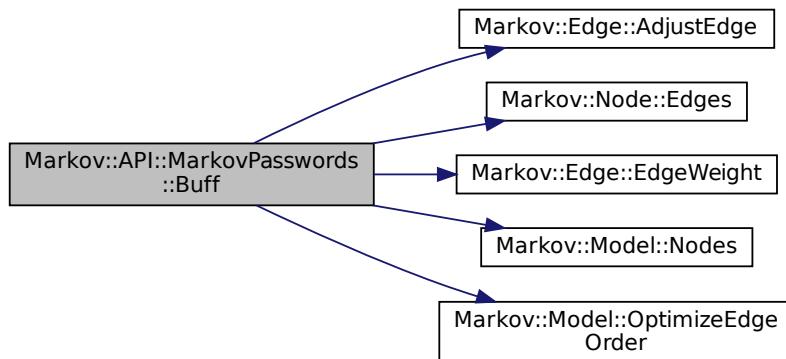
```

00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
00160         edges = node->Edges();
00161         for (auto const& [targetrepr, edge] : *edges) {
00162             if(buffstr.find(targetrepr)!= std::string::npos) {
00163                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164                 if(bDontAdjustExtendedLoops) {
00165                     if(buffstr.find(repr)!= std::string::npos) {
00166                         continue;
00167                     }
00168                 }
00169                 long int weight = edge->EdgeWeight ();
00170                 weight = weight*multiplier;
00171                 edge->AdjustEdge(weight);
00172             }
00173         }
00174     }
00175     i++;
00176 }
00177
00178 this->OptimizeEdgeOrder();
00179 }
```

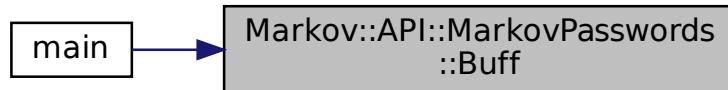
References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.25.3.3 ConstructMatrix()

`bool Markov::API::ModelMatrix::ConstructMatrix ()`

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content.

this will initialize: `char** edgeMatrix ->` a 2D array of mapping left and right connections of each edge. `long int **valueMatrix ->` a 2D array representing the edge weights. `int matrixSize ->` Size of the matrix, aka total number of nodes. `char* matrixIndex ->` order of nodes in the model `long int *totalEdgeWeights ->` total edge weights of each `Node`.

Returns

True if constructed. False if already constructed.

Definition at line 31 of file `modelMatrix.cpp`.

```

00031
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070                 this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072         }
00073     }
00074     i++;
00075 }

```

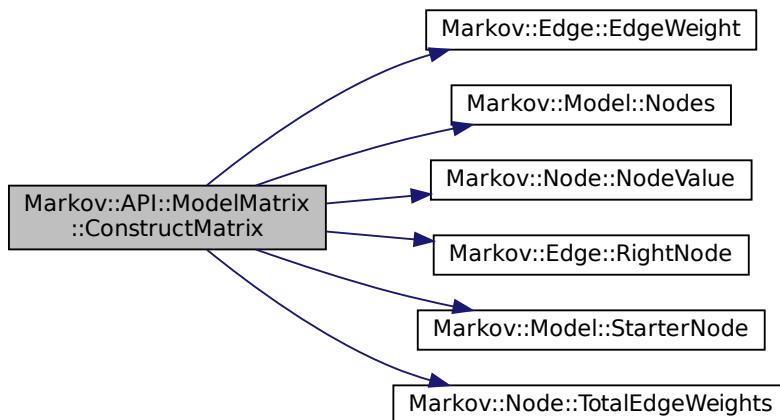
```

00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

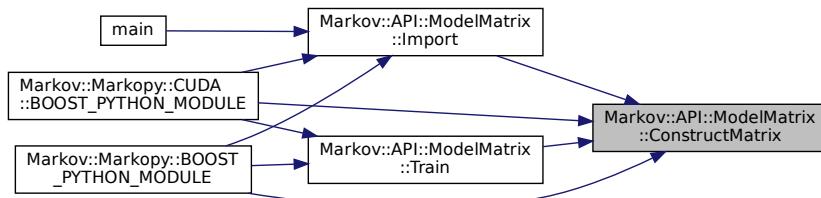
References `edgeMatrix`, `Markov::Edge< NodeStorageType >::EdgeWeight()`, `matrixIndex`, `matrixSize`, `Markov::Model< NodeStorageType >::ModelMatrix`, `Markov::Node< storageType >::NodeValue()`, `ready`, `Markov::Edge< NodeStorageType >::RightNode()`, `Markov::Model< NodeStorageType >::StarterNode()`, `totalEdgeWeights`, `Markov::Node< storageType >::TotalEdgeWeights()`, and `valueMatrix`.

Referenced by `Markov::Markopy::CUDA::BOOST_PYTHON_MODULE()`, `Markov::Markopy::BOOST_PYTHON_MODULE()`, `Import()`, and `Train()`.

Here is the call graph for this function:



Here is the caller graph for this function:



9.25.3.4 DeallocateMatrix()

```
bool Markov::API::ModelMatrix::DeallocateMatrix( ) [protected]
```

Deallocate matrix and make it ready for re-construction.

Returns

True if deallocated. False if matrix was not initialized

Definition at line 81 of file `modelMatrix.cpp`.

```

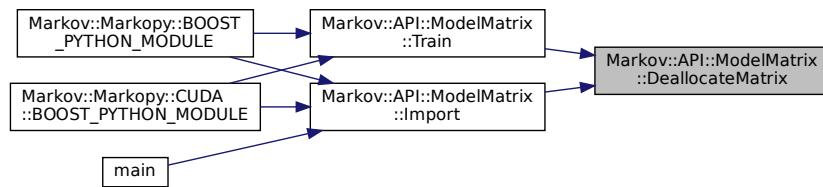
00081
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085
00086     for(int i=0;i<this->matrixSize;i++) {
```

```
00087     delete[] this->edgeMatrix[i];
00088 }
00089 delete[] this->edgeMatrix;
00090
00091 for(int i=0;i<this->matrixSize;i++){
00092     delete[] this->valueMatrix[i];
00093 }
00094 delete[] this->valueMatrix;
00095
00096 this->matrixSize = -1;
00097 this->ready = false;
00098 return true;
00099 }
```

References `edgeMatrix`, `matrixIndex`, `matrixSize`, `ready`, `totalEdgeWeights`, and `valueMatrix`.

Referenced by `Import()`, and `Train()`.

Here is the caller graph for this function:



9.25.3.5 DumpJSON()

void Markov::API::ModelMatrix::DumpJSON ()
Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

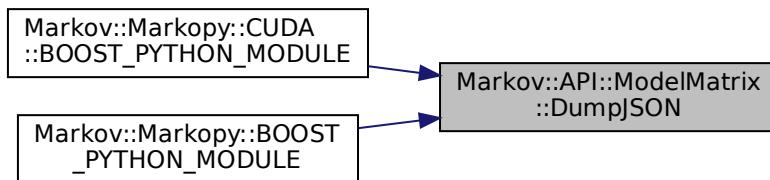
Definition at line 101 of file `modelMatrix.cpp`.

```

00141     else std::cout << "      \\" << this->matrixIndex[i] << "\": [";
00142
00143     for(int j=0;j<this->matrixSize;j++){
00144         std::cout << this->valueMatrix[i][j];
00145         if(j!=this->matrixSize-1) std::cout << ", ";
00146     }
00147     std::cout << "],\n";
00148 }
00149 std::cout << " }\n}\n";
00150 }
```

References [edgeMatrix](#), [matrixIndex](#), [matrixSize](#), and [valueMatrix](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the caller graph for this function:



9.25.3.6 Edges()

`std::vector<Edge<char *>>* Markov::Model<char>::Edges\(\) [inline], [inherited]`
Return a vector of all the edges in the model.

Returns

`vector of edges`

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.25.3.7 Export() [1/2]

`bool Markov::Model<char>::Export\(\) (const char * filename) [inherited]`

Open a file to export with filename, and call `bool Model::Export` with `std::ofstream`.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```

00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.25.3.8 Export() [2/2]

`bool Markov::Model<char>::Export\(\) (std::ofstream * f) [inherited]`

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file [model.h](#).

```
00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode() ->NodeValue() << "," << e->EdgeWeight() << "," <<
00293         *f << e->LeftNode() ->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode() ->NodeValue() <<
00294         "\n";
00295     }
00296     return true;
00297 }
```

9.25.3.9 FastRandomWalk() [1/2]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true )
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

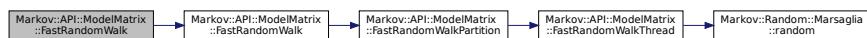
Definition at line 217 of file [modelMatrix.cpp](#).

```
00217
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

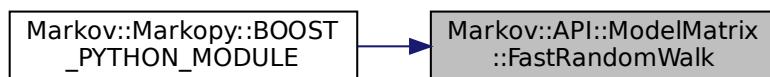
References [FastRandomWalk\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.25.3.10 FastRandomWalk() [2/2]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [protected]
```

[Random walk on the Matrix-reduced Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

Definition at line 204 of file [modelMatrix.cpp](#).

```
00204
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
threads);
```

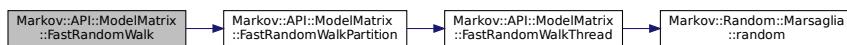
```

00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
00213             threads);
00214     }
00215 }
```

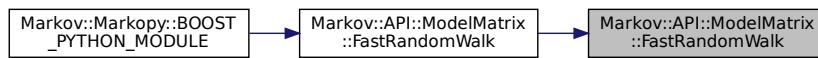
References [FastRandomWalkPartition\(\)](#).

Referenced by [FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.25.3.11 FastRandomWalkPartition()

```

void Markov::API::ModelMatrix::FastRandomWalkPartition (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    bool bFileIO,
    int threads ) [protected]
```

A single partition of FastRandomWalk event.

Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation
- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 225 of file [modelMatrix.cpp](#).

```

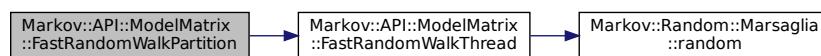
00225
00226     int iterationsPerThread = n/threads;
00227     int iterationsPerThreadCarryOver = n%threads;
00228
00229     std::vector<std::thread*> threadsV;
00230
00231     int id = 0;
00232     for(int i=0;i<threads;i++) {
00233         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00234                                         mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00235         id++;
00236     }
00237
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00239                                         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00240
00241     for(int i=0;i<threads;i++) {
00242         threadsV[i]->join();
00243     }

```

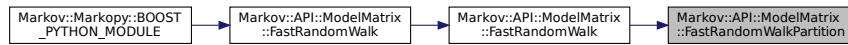
References [FastRandomWalkThread\(\)](#).

Referenced by [FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.25.3.12 FastRandomWalkThread()

```

void Markov::API::ModelMatrix::FastRandomWalkThread (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    int id,
    bool bFileIO ) [protected]

```

A single thread of a single partition of FastRandomWalk.

A FastRandomWalkPartition will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate

Parameters

<i>id</i>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 153 of file [modelMatrix.cpp](#).

```

00153
00154     if(n==0)  return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179             if (len >= maxLen)  break;
00180             else if ((next < 0) && (len < minLen))  continue;
00181             else if (next < 0)  break;
00182             cur = next;
00183             res[bufferctr + len++] = cur;
00184         }
00185     }
00186     res[bufferctr + len++] = '\n';
00187     bufferctr+=len;
00188
00189 }
00190 if(bFileIO){
00191     mlock->lock();
00192     *wordlist << res;
00193     mlock->unlock();
00194 }else{
00195     mlock->lock();
00196     std::cout << res;
00197     mlock->unlock();
00198 }
00199 delete res;
00200
00201 }
```

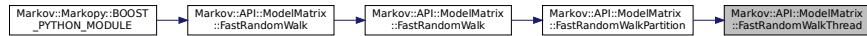
References [edgeMatrix](#), [matrixIndex](#), [matrixSize](#), [Markov::Random::Marsaglia::random\(\)](#), [totalEdgeWeights](#), and [valueMatrix](#).

Referenced by [FastRandomWalkPartition\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.25.3.13 Generate()

```
void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]
```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

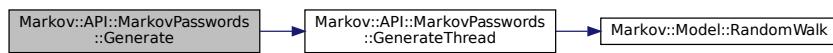
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

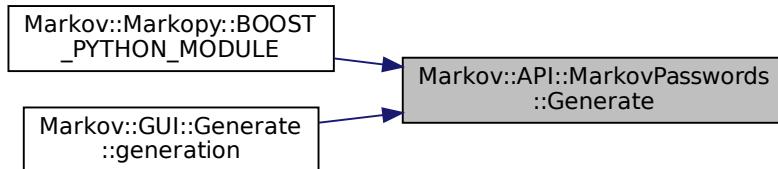
```
00118
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++) {
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129             &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130     }
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00136
00137
00138 }
```

References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).
Here is the call graph for this function:



Here is the caller graph for this function:



9.25.3.14 GenerateThread()

```
void Markov::API::MarkovPasswords::GenerateThread (
    std::mutex * outputLock,
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen,
    int maxLen ) [private], [inherited]
```

A single thread invoked by the Generate function.

DEPRECATED: See [Markov::API::MatrixModel::FastRandomWalkThread](#) for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Parameters

<i>outputLock</i>	- shared mutex lock to lock during output operation. Prevents race condition on write.
<i>n</i>	number of lines to be generated by this thread
<i>wordlist</i>	wordlistfile
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate

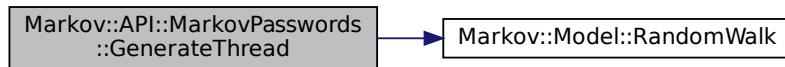
Definition at line 140 of file [markovPasswords.cpp](#).

```
00140
00141     char* res = new char[maxLen+5];
00142     if(n==0) return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist << res << "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

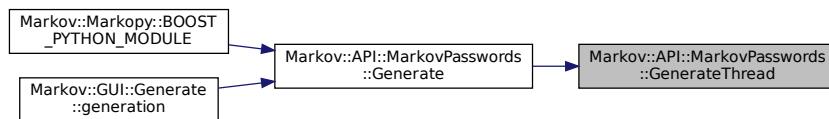
References [Markov::Model< NodeStorageType >::RandomWalk\(\)](#).

Referenced by [Markov::API::MarkovPasswords::Generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.25.3.15 Import() [1/2]

```
void Markov::API::ModelMatrix::Import (
    const char * filename )
```

Open a file to import with filename, and call bool [Model::Import](#) with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.

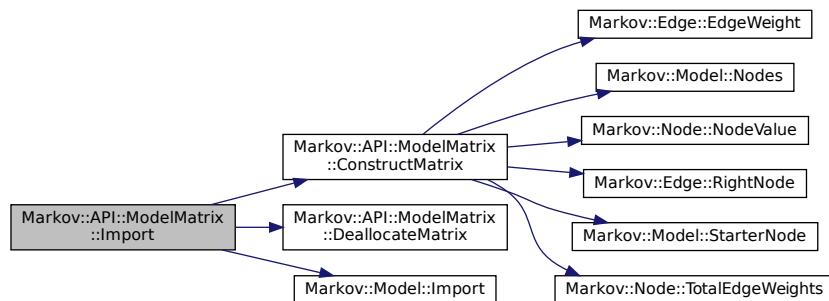
Definition at line 19 of file [modelMatrix.cpp](#).

```
00019
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

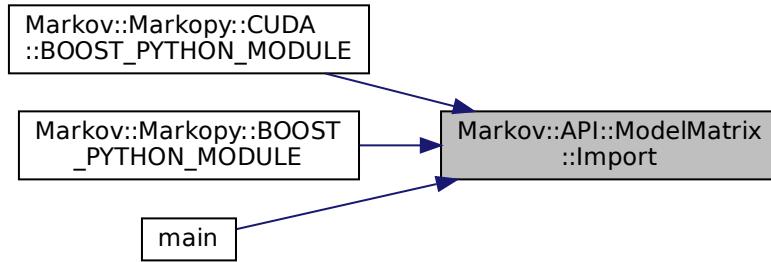
References [ConstructMatrix\(\)](#), [DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.25.3.16 Import() [2/2]

```
bool Markov::Model< char >::Import (
    std::ifstream * f )  [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file model.h.

```
00216     std::string cell;                                {
00217     std::string cell;
00218     char src;
00219     char target;
00220     long int oc;
00221
00222     while (std::getline(*f, cell)) {
00223         //std::cout << "cell: " << cell << std::endl;
00224         src = cell[0];
00225         target = cell[cell.length() - 1];
00226         char* j;
00227         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(),&j,10);
00228         //std::cout << oc << "\n";
00229         Markov::Node<NodeStorageType>* srcN;
00230         Markov::Node<NodeStorageType>* targetN;
00231         Markov::Edge<NodeStorageType>* e;
00232         if (this->nodes.find(src) == this->nodes.end()) {
00233             srcN = new Markov::Node<NodeStorageType>(src);
00234             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00235             //std::cout << "Creating new node at start.\n";
00236         }
00237     }
00238     else {
00239         srcN = this->nodes.find(src)->second;
00240     }
00241
00242     if (this->nodes.find(target) == this->nodes.end()) {
00243         targetN = new Markov::Node<NodeStorageType>(target);
00244         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00245         //std::cout << "Creating new node at end.\n";
00246     }
00247     else {
00248         targetN = this->nodes.find(target)->second;
00249     }
00250     e = srcN->Link(targetN);
00251     e->AdjustEdge(oc);
```

```

00252     this->edges.push_back(e);
00253
00254     //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "-->" <<
00255     int(targetN->NodeValue()) << "\n";
00256
00257 }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.25.3.17 Nodes()

`std::map<char , Node<char *>>* Markov::Model< char >::Nodes () [inline], [inherited]`
 Return starter [Node](#).

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.25.3.18 OpenDatasetFile()

`std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (`
 `const char * filename) [inherited]`

Open dataset file and return the ifstream pointer.

Parameters

<code>filename</code>	- Filename to open
-----------------------	--------------------

Returns

`ifstream*` to the the dataset file

Definition at line 51 of file [markovPasswords.cpp](#).

```

00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References [Markov::Model< NodeStorageType >::Import\(\)](#).

Here is the call graph for this function:



9.25.3.19 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder [inherited]
Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file model.h.
00265
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [](Edge<NodeStorageType> *lhs,
00269             Edge<NodeStorageType> *rhs)->bool{
00270             return lhs->EdgeWeight () > rhs->EdgeWeight ();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         // std::cout << x.second->edgesV[i]->EdgeWeight () << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00278 }
```

9.25.3.20 RandomWalk()

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from

This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```
00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
```

```

00312     temp_node = n->RandomNext (randomEngine);
00313     if (len >= maxSetting) {
00314         break;
00315     }
00316     else if ((temp_node == NULL) && (len < minSetting)) {
00317         continue;
00318     }
00319
00320     else if (temp_node == NULL) {
00321         break;
00322     }
00323
00324     n = temp_node;
00325
00326     buffer[len++] = n->NodeValue ();
00327 }
00328
00329 //null terminate the string
00330 buffer[len] = 0x00;
00331
00332 //do something with the generated string
00333 return buffer; //for now
00334 }
```

9.25.3.21 Save()

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
Export model to file.
```

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

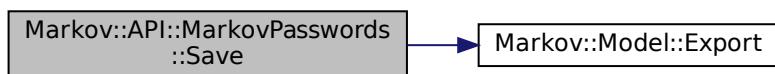
```

00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

{

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.25.3.22 StarterNode()

```
Node< char *>* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.25.3.23 Train()

```
void Markov::API::ModelMatrix::Train (
    const char * datasetFileName,
    char delimiter,
    int threads )
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

Construct the matrix when done.

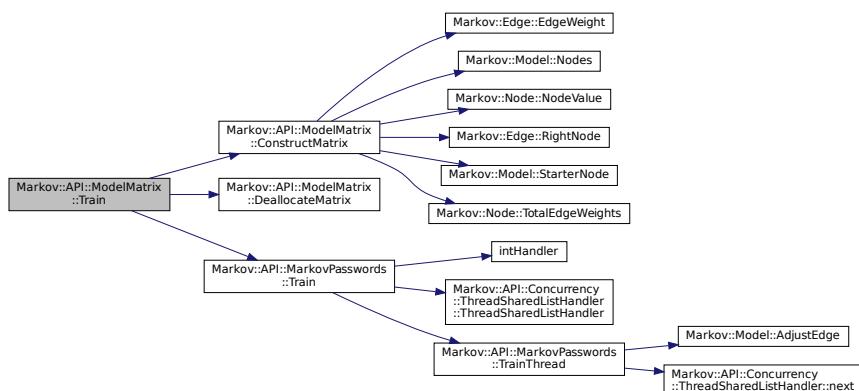
Definition at line 25 of file [modelMatrix.cpp](#).

```
00025
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

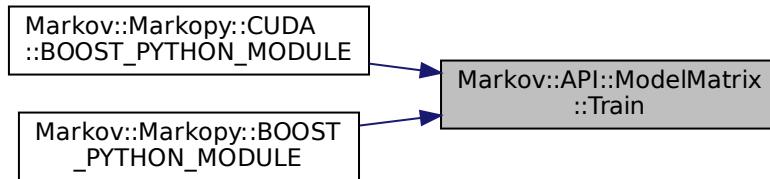
References [ConstructMatrix\(\)](#), [DeallocateMatrix\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.25.3.24 TrainThread()

```

void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private], [inherited]
  
```

A single thread invoked by the Train function.

Parameters

<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

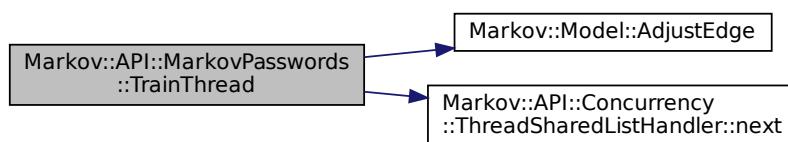
Definition at line 85 of file [markovPasswords.cpp](#).

```

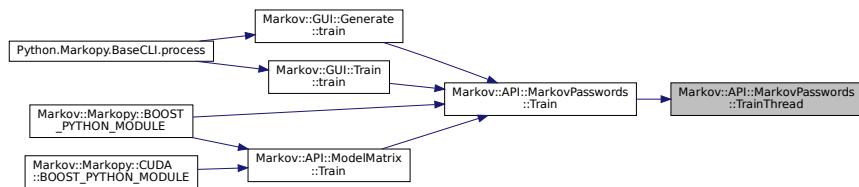
00085     {
00086     char format_str[] = "%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to-
00097         "%ld,%s"
00098     #else
00099         sscanf(line.c_str(), format_str, &oc, linebuf);
00100     #endif
00101     this->AdjustEdge((const char*)linebuf, oc);
00102     delete linebuf;
00103 }
  
```

References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), [keepRunning](#), and [Markov::API::Concurrency::ThreadSharedListHandler::next](#).
Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.25.4 Member Data Documentation

9.25.4.1 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]`
Definition at line 123 of file [markovPasswords.h](#).

9.25.4.2 edgeMatrix

`char** Markov::API::ModelMatrix::edgeMatrix [protected]`
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line 175 of file [modelMatrix.h](#).
Referenced by [ConstructMatrix\(\)](#), [DeallocateMatrix\(\)](#), [DumpJSON\(\)](#), and [FastRandomWalkThread\(\)](#).

9.25.4.3 edges

`std::vector<Edge<char *>> Markov::Model< char >::edges [private], [inherited]`
A list of all edges in this model.
Definition at line 204 of file [model.h](#).

9.25.4.4 matrixIndex

`char* Markov::API::ModelMatrix::matrixIndex [protected]`
to hold the Matrix index (To hold the orders of 2-D arrays)
Definition at line 190 of file [modelMatrix.h](#).
Referenced by [ConstructMatrix\(\)](#), [DeallocateMatrix\(\)](#), [DumpJSON\(\)](#), and [FastRandomWalkThread\(\)](#).

9.25.4.5 matrixSize

`int Markov::API::ModelMatrix::matrixSize [protected]`
to hold Matrix size
Definition at line 185 of file [modelMatrix.h](#).
Referenced by [ConstructMatrix\(\)](#), [DeallocateMatrix\(\)](#), [DumpJSON\(\)](#), [FastRandomWalkThread\(\)](#), and [Markov::API::CUDA::CUDAMode](#).

9.25.4.6 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`
Dataset file input of our system

Definition at line 124 of file [markovPasswords.h](#).

9.25.4.7 nodes

```
std::map<char , Node<char *>> Markov::Model< char >::nodes [private], [inherited]
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file model.h.
```

9.25.4.8 outputFile

```
std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]
File to save model of our system
```

Definition at line 125 of file [markovPasswords.h](#).

9.25.4.9 ready

```
bool Markov::API::ModelMatrix::ready [protected]
True when matrix is constructed. False if not.
Definition at line 200 of file modelMatrix.h.
Referenced by ConstructMatrix\(\), DeallocateMatrix\(\), and ModelMatrix\(\).
```

9.25.4.10 starterNode

```
Node<char *>* Markov::Model< char >::starterNode [private], [inherited]
Starter Node of this model.
Definition at line 198 of file model.h.
```

9.25.4.11 totalEdgeWeights

```
long int* Markov::API::ModelMatrix::totalEdgeWeights [protected]
Array of the Total Edge Weights.
Definition at line 195 of file modelMatrix.h.
Referenced by ConstructMatrix\(\), DeallocateMatrix\(\), and FastRandomWalkThread\(\).
```

9.25.4.12 valueMatrix

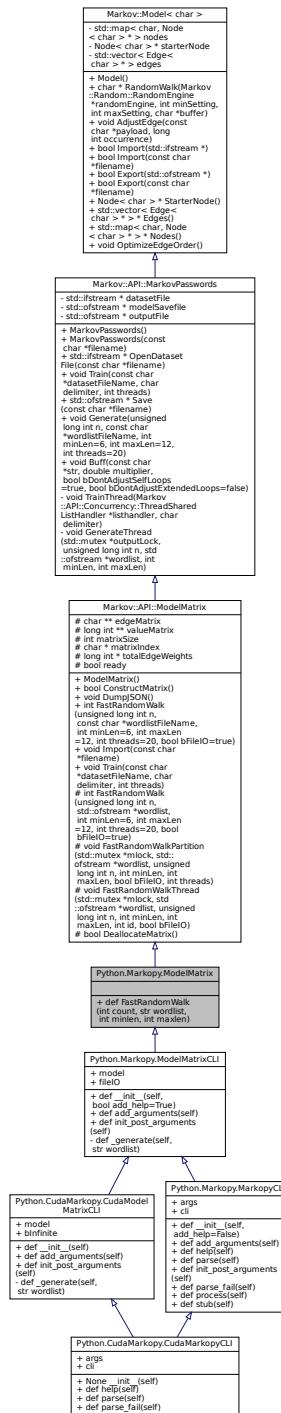
```
long int** Markov::API::ModelMatrix::valueMatrix [protected]
2-d Integer array for the value Matrix (For the weights of Edges)
Definition at line 180 of file modelMatrix.h.
Referenced by ConstructMatrix\(\), DeallocateMatrix\(\), DumpJSON\(\), and FastRandomWalkThread\(\).
The documentation for this class was generated from the following files:
```

- [Markopy/MarkovAPI/src/modelMatrix.h](#)
- [Markopy/MarkovAPI/src/modelMatrix.cpp](#)

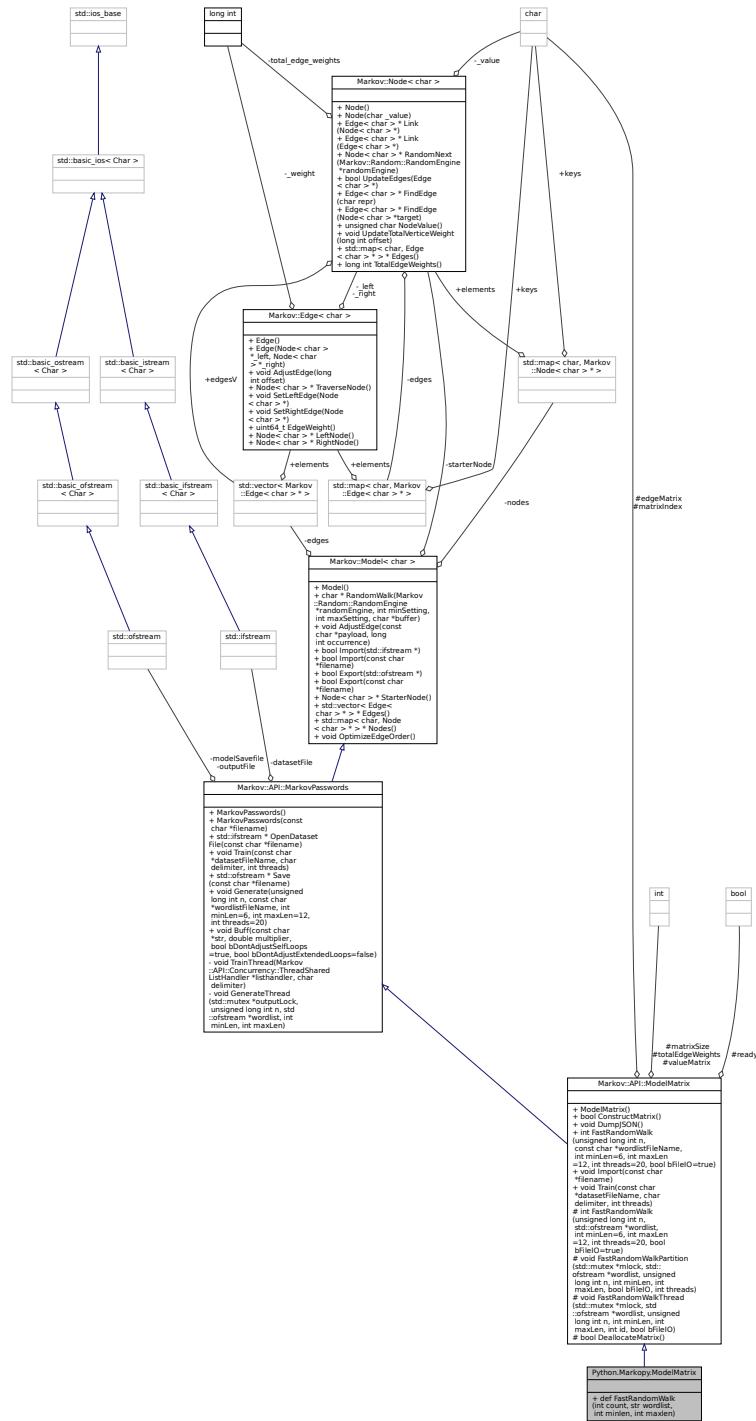
9.26 Python.Markopy.ModelMatrix Class Reference

Abstract representation of a matrix based model.

Inheritance diagram for Python.Markopy.ModelMatrix:



Collaboration diagram for Python.Markopy.ModelMatrix:



Public Member Functions

- def **FastRandomWalk** (int count, str wordlist, int minlen, int maxlen)
 - bool **ConstructMatrix** ()

Construct the related Matrix data for the model

- void DumpJSON ()

Debug function to dump the model to a .JSON file

- int **FastRandomWalk** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced `Markov::Model`.
- void **Import** (const char *filename)

Open a file to import with filename, and call bool `Model::Import` with std::ifstream.
- bool **Import** (std::ifstream *)

Import a file to construct the model.
- void **Train** (const char *datasetFileName, char delimiter, int threads)

Train the model with the dataset file.
- std::ifstream * **OpenDatasetFile** (const char *filename)

Open dataset file and return the ifstream pointer.
- std::ofstream * **Save** (const char *filename)

Export model to file.
- void **Generate** (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

Call `Markov::Model::RandomWalk` n times, and collect output.
- void **Buff** (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)

Buff expression of some characters in the model.
- char * **RandomWalk** (`Markov::Random::RandomEngine` *randomEngine, int minSetting, int maxSetting, char *buffer)

Do a random walk on this model.
- void **AdjustEdge** (const char *payload, long int occurrence)

Adjust the model with a single string.
- bool **Export** (std::ofstream *)

Export a file of the model.
- bool **Export** (const char *filename)

Open a file to export with filename, and call bool `Model::Export` with std::ofstream.
- Node< char > * **StarterNode** ()

Return starter Node.
- std::vector< Edge< char > * > * **Edges** ()

Return a vector of all the edges in the model.
- std::map< char, Node< char > * > * **Nodes** ()

Return starter Node.
- void **OptimizeEdgeOrder** ()

Sort edges of all nodes in the model ordered by edge weights.

Protected Member Functions

- int **FastRandomWalk** (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)

Random walk on the Matrix-reduced `Markov::Model`.
- void **FastRandomWalkPartition** (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)

A single partition of `FastRandomWalk` event.
- void **FastRandomWalkThread** (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)

A single thread of a single partition of `FastRandomWalk`.
- bool **DeallocateMatrix** ()

Deallocate matrix and make it ready for re-construction.

Protected Attributes

- char ** `edgeMatrix`
2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** `valueMatrix`
2-d Integer array for the value Matrix (For the weights of Edges)
- int `matrixSize`
to hold Matrix size
- char * `matrixIndex`
to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * `totalEdgeWeights`
Array of the Total Edge Weights.
- bool `ready`
True when matrix is constructed. False if not.

Private Member Functions

- void `TrainThread` (Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char delimiter)
A single thread invoked by the Train function.
- void `GenerateThread` (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)
A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * `datasetFile`
Dataset file input of our system
- std::ofstream * `modelSavefile`
File to save model of our system
- std::ofstream * `outputFile`
File to save model of our system
- std::map<char, Node<char>*> <`nodes`>
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node<char> * `starterNode`
Starter Node of this model.
- std::vector<Edge<char>*> <`edges`>
A list of all edges in this model.

9.26.1 Detailed Description

Abstract representation of a matrix based model.
 To help with the python-cpp gateway documentation.
 Definition at line 38 of file `mm.py`.

9.26.2 Member Function Documentation

9.26.2.1 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.26.2.2 Buff()

```
void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

```
00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
```

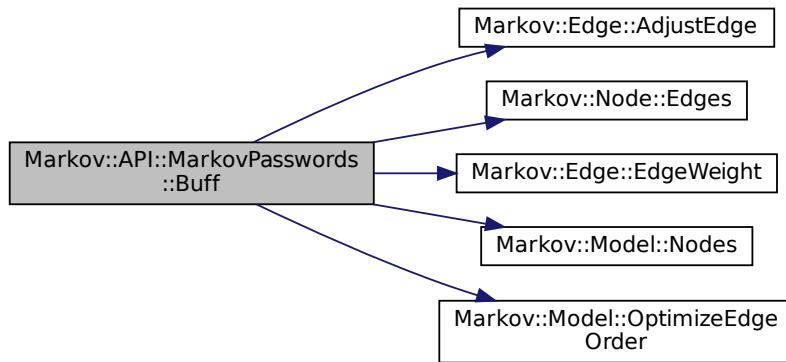
```

00160     edges = node->Edges();
00161     for (auto const& [targetrepr, edge] : *edges){
00162         if(buffstr.find(targetrepr) != std::string::npos) {
00163             if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164             if(bDontAdjustExtendedLoops) {
00165                 if(buffstr.find(repr) != std::string::npos) {
00166                     continue;
00167                 }
00168                 long int weight = edge->EdgeWeight();
00169                 weight = weight*multiplier;
00170                 edge->AdjustEdge(weight);
00171             }
00172         }
00173     }
00174     i++;
00175 }
00176 }
00177 this->OptimizeEdgeOrder();
00178 }
```

References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.26.2.3 ConstructMatrix()

`bool Markov::API::ModelMatrix::ConstructMatrix () [inherited]`

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content.

this will initialize: char** edgeMatrix -> a 2D array of mapping left and right connections of each edge. long int **valueMatrix -> a 2D array representing the edge weights. int matrixSize -> Size of the matrix, aka total number of nodes. char* matrixIndex -> order of nodes in the model long int *totalEdgeWeights -> total edge weights of each Node.

Returns

True if constructed. False if already construced.

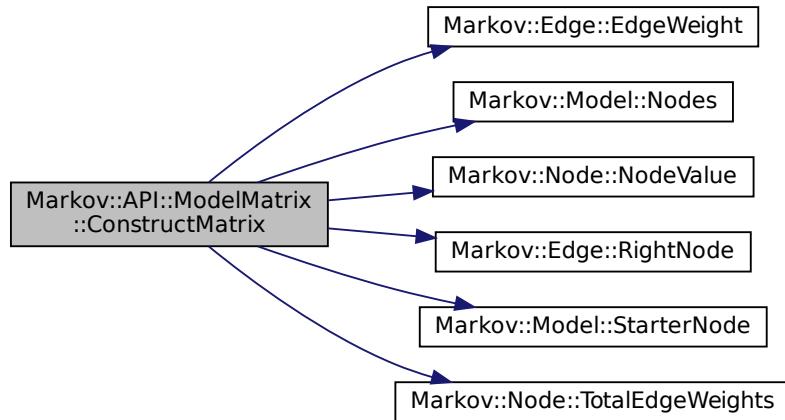
Definition at line 31 of file `modelMatrix.cpp`.

```

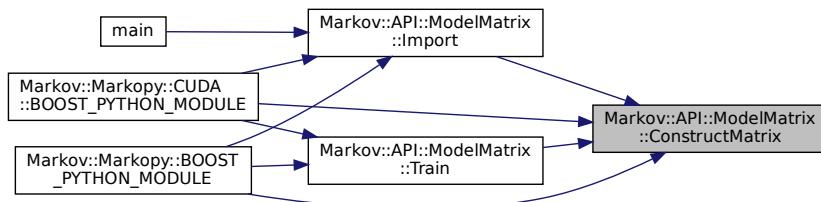
00031         {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight ();
00070                 this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();
00071             }
00072         }
00073     }
00074     i++;
00075 }
00076 this->ready = true;
00077 return true;
00078 //this->DumpJSON();
00079 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::API::ModelMatrix::Markov::API::ModelMatrix::matrixSize](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), [Markov::Node< storageType >::NodeValue\(\)](#), [Markov::API::ModelMatrix::ready](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::Starter\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), [Markov::Node< storageType >::TotalEdgeWeights\(\)](#), and [Markov::API::ModelMatrix::valueMatrix](#). Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.26.2.4 DeallocateMatrix()

```
bool Markov::API::ModelMatrix::DeallocateMatrix ( ) [protected], [inherited]
Deallocate matrix and make it ready for re-construction.
```

Returns

True if deallocated. False if matrix was not initialized

Definition at line 81 of file [modelMatrix.cpp](#).

```

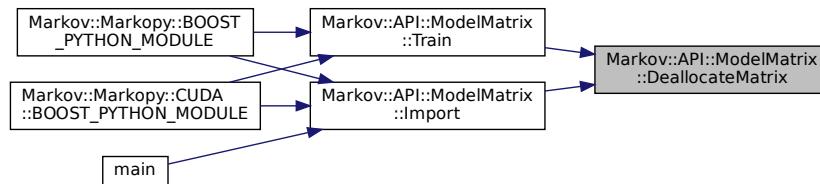
00081     if(!this->ready) return false;
00082     delete[] this->matrixIndex;
00083     delete[] this->totalEdgeWeights;
00084
00085     for(int i=0;i<this->matrixSize;i++) {
00086         delete[] this->edgeMatrix[i];
00087     }
00088     delete[] this->edgeMatrix;
00089
00090     for(int i=0;i<this->matrixSize;i++) {
00091         delete[] this->valueMatrix[i];
00092     }
00093     delete[] this->valueMatrix;
00094
00095     this->matrixSize = -1;
00096     this->ready = false;
00097
00098     return true;
  
```

```
00099 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::API::ModelMatrix::ready](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the caller graph for this function:



9.26.2.5 DumpJSON()

```
void Markov::API::ModelMatrix::DumpJSON() [inherited]
```

Debug function to dump the model to a JSON file.

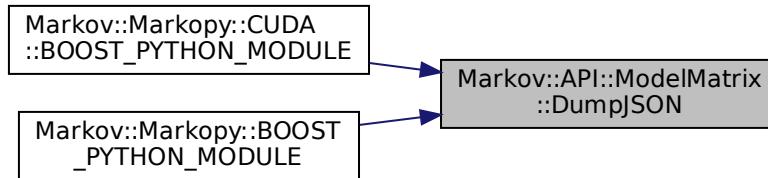
Might not work 100%. Not meant for production use.

Definition at line 101 of file [modelMatrix.cpp](#).

```
00101 {
00102
00103     std::cout << "{\n      \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout << "\\\\"\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout << "\\\\\\"";
00107         else if(this->matrixIndex[i]==0) std::cout << "\\\\x00";
00108         else if(i==0) std::cout << "\\\\xff";
00109         else if(this->matrixIndex[i]=='\\n') std::cout << "\\n";
00110         else std::cout << this->matrixIndex[i];
00111     }
00112     std::cout <
00113     "\\",\\n"
00114     "  \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++){
00117         if(this->matrixIndex[i]=='"') std::cout << "      \"\\\\\"\\\"": [";
00118         else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\\\\\\\"": [";
00119         else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\\\\\x00": [";
00120         else if(this->matrixIndex[i]<0) std::cout << "      \"\\\\\\\\xf\\": [";
00121         else std::cout << "      \" \" << this->matrixIndex[i] << ": [";
00122         for(int j=0;j<this->matrixSize;j++){
00123             if(this->edgeMatrix[i][j]=='"') std::cout << "        \"\\\\\"\\\"";
00124             else if(this->edgeMatrix[i][j]=='\\') std::cout << "        \"\\\\\\\\\\\"";
00125             else if(this->edgeMatrix[i][j]==0) std::cout << "        \"\\\\\\\\x00";
00126             else if(this->edgeMatrix[i][j]<0) std::cout << "        \"\\\\\\\\xf\\";
00127             else if(this->matrixIndex[i]=='\\n') std::cout << "        \"\\n";
00128             else std::cout << "        \" \" << this->edgeMatrix[i][j] << \"\"";
00129             if(j!=this->matrixSize-1) std::cout << ", ";
00130         }
00131         std::cout << "],\\n";
00132     }
00133     std::cout << "},\\n";
00134
00135     std::cout << "  \" weightmap\": {\n";
00136     for(int i=0;i<this->matrixSize;i++){
00137         if(this->matrixIndex[i]=='"') std::cout << "    \"\\\\\"\\\"": [";
00138         else if(this->matrixIndex[i]=='\\') std::cout << "    \"\\\\\\\\\\\"": [";
00139         else if(this->matrixIndex[i]==0) std::cout << "    \"\\\\\\\\x00": [";
00140         else if(this->matrixIndex[i]<0) std::cout << "    \"\\\\\\\\xf\\": [";
00141         else std::cout << "    \" \" << this->matrixIndex[i] << ": [";
00142
00143         for(int j=0;j<this->matrixSize;j++){
00144             std::cout << this->valueMatrix[i][j];
00145             if(j!=this->matrixSize-1) std::cout << ", ";
00146         }
00147         std::cout << "],\\n";
00148     }
00149     std::cout << "  }\\n}\\n";
00150 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the caller graph for this function:



9.26.2.6 Edges()

```
std::vector<Edge<char *>>* Markov::Model< char >::Edges ( ) [inherited]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.26.2.7 Export() [1/2]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.26.2.8 Export() [2/2]

```
bool Markov::Model< char >::Export (
    std::ofstream * f ) [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ostream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file model.h.

```
00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <
00294         "\n";
00295     }
00296     return true;
00297 }
```

9.26.2.9 FastRandomWalk() [1/3]

```
def Python.Markopy.ModelMatrix.FastRandomWalk (
    int count,
    str wordlist,
    int minlen,
    int maxlen )
```

Definition at line 48 of file mm.py.

```
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

Referenced by [Python.Markopy.ModelMatrixCLI._generate\(\)](#).

Here is the caller graph for this function:

**9.26.2.10 FastRandomWalk() [2/3]**

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to

Parameters

<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

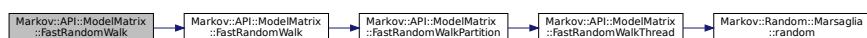
Definition at line 217 of file [modelMatrix.cpp](#).

```
00217
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

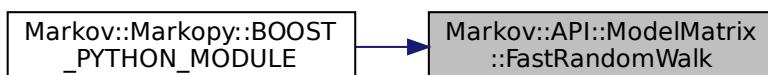
References [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.26.2.11 FastRandomWalk() [3/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [protected], [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

Definition at line 204 of file [modelMatrix.cpp](#).

```
00204
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
00209         threads);
00210     else{
00211         int numberOfPartitions = n/50000000ull;
00212         for(int i=0;i<numberOfPartitions;i++)
00213             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
00214             threads);
00215     }
00216     return 0;
00217 }
```

References [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.26.2.12 FastRandomWalkPartition()

```
void Markov::API::ModelMatrix::FastRandomWalkPartition (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    bool bFileIO,
    int threads ) [protected], [inherited]
```

A single partition of FastRandomWalk event.

Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation

- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 225 of file [modelMatrix.cpp](#).

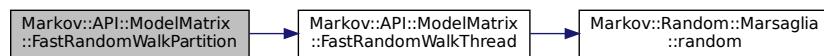
```

00225
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++) {
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00235                                         mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00236         id++;
00237     }
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00239                                         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00240     for(int i=0;i<threads;i++) {
00241         threadsV[i]->join();
00242     }
00243 }
```

References [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.26.2.13 FastRandomWalkThread()

```
void Markov::API::ModelMatrix::FastRandomWalkThread (
    std::mutex * mlock,
```

```

    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    int id,
    bool bFileIO ) [protected], [inherited]

```

A single thread of a single partition of FastRandomWalk.

A FastRandomWalkPartition will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>id</i>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 153 of file [modelMatrix.cpp](#).

```

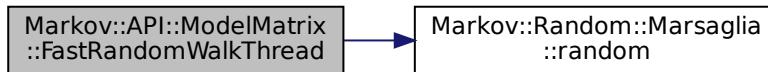
00153
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179             if (len >= maxLen) break;
00180             else if ((next < 0) && (len < minLen)) continue;
00181             else if (next < 0) break;
00182             cur = next;
00183             res[bufferctr + len++] = cur;
00184         }
00185     }
00186     res[bufferctr + len++] = '\n';
00187     bufferctr+=len;
00188
00189 }
00190 if(bFileIO){
00191     mlock->lock();
00192     *wordlist << res;
00193     mlock->unlock();
00194 }else{
00195     mlock->lock();
00196     std::cout << res;
00197     mlock->unlock();
00198 }
00199 delete res;
00200
00201 }

```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::Random::Marsaglia::random\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.26.2.14 Generate()

```

void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]

```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

```

00118
00119     char* res;
00120     char print[100];
00121     std::ofstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++) {
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129                                         &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130     }
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }

```

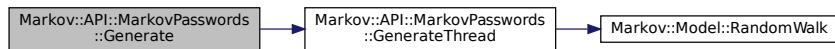
```

00135     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00136
00137
00138 }
```

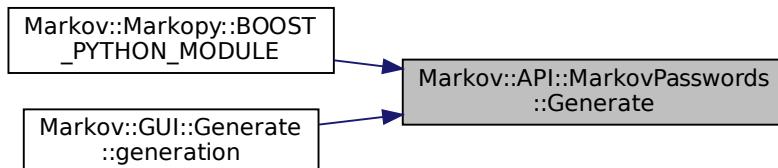
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.26.2.15 GenerateThread()

```

void Markov::API::MarkovPasswords::GenerateThread (
    std::mutex * outputLock,
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen,
    int maxLen ) [private], [inherited]
```

A single thread invoked by the Generate function.

DEPRECATED: See [Markov::API::MatrixModel::FastRandomWalkThread](#) for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Parameters

<i>outputLock</i>	- shared mutex lock to lock during output operation. Prevents race condition on write.
<i>n</i>	number of lines to be generated by this thread
<i>wordlist</i>	wordlistfile
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate

Definition at line 140 of file [markovPasswords.cpp](#).

```

00140
00141     {
00142         char* res = new char[maxLen+5];
00143         if(n==0) return;
00144         Markov::Random::Marsaglia MarsagliaRandomEngine;
00145         for (int i = 0; i < n; i++) {
00146             this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147             outputLock->lock();
```

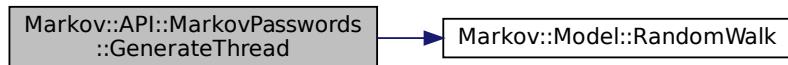
```

00148     *wordlist << res << "\n";
00149     outputLock->unlock();
00150 }
00151 }
```

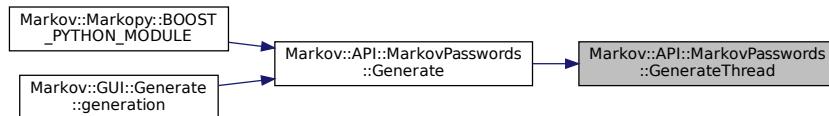
References [Markov::Model< NodeStorageType >::RandomWalk\(\)](#).

Referenced by [Markov::API::MarkovPasswords::Generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.26.2.16 Import() [1/2]

```
void Markov::API::ModelMatrix::Import (
    const char * filename ) [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.

Definition at line 19 of file [modelMatrix.cpp](#).

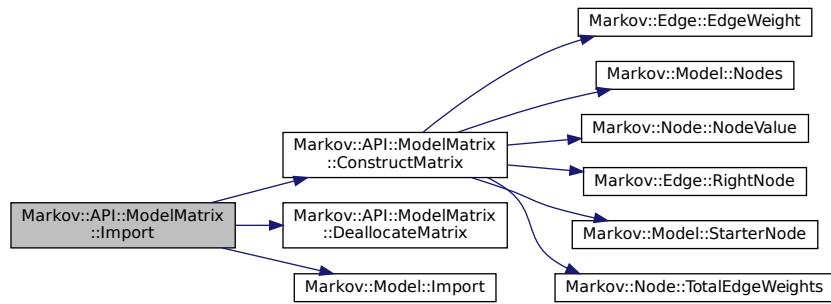
```

00019
00020     this->DeallocateMatrix();                                     {
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
```

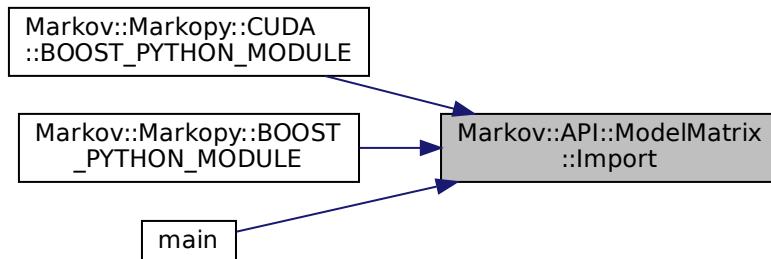
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.26.2.17 Import() [2/2]

```
bool Markov::Model< char >::Import (
    std::ifstream * f) [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file [model.h](#).

```
00216
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
```

```

00227     char* j;
00228     oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00229     //std::cout << oc << "\n";
00230     Markov::Node<NodeStorageType>* srcN;
00231     Markov::Node<NodeStorageType>* targetN;
00232     Markov::Edge<NodeStorageType>* e;
00233     if (this->nodes.find(src) == this->nodes.end()) {
00234         srcN = new Markov::Node<NodeStorageType>(src);
00235         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00236         //std::cout << "Creating new node at start.\n";
00237     }
00238     else {
00239         srcN = this->nodes.find(src)->second;
00240     }
00241
00242     if (this->nodes.find(target) == this->nodes.end()) {
00243         targetN = new Markov::Node<NodeStorageType>(target);
00244         this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00245         //std::cout << "Creating new node at end.\n";
00246     }
00247     else {
00248         targetN = this->nodes.find(target)->second;
00249     }
00250     e = srcN->Link(targetN);
00251     e->AdjustEdge(oc);
00252     this->edges.push_back(e);
00253
00254     //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00255     int(targetN->NodeValue()) << "\n";
00256
00257 }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
```

9.26.2.18 Nodes()

`std::map<char , Node<char >>* Markov::Model< char >::Nodes () [inline], [inherited]`
Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.26.2.19 OpenDatasetFile()

`std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (const char * filename) [inherited]`

Open dataset file and return the ifstream pointer.

Parameters

<code>filename</code>	- Filename to open
-----------------------	--------------------

Returns

`ifstream*` to the the dataset file

Definition at line 51 of file [markovPasswords.cpp](#).

```

00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058 }
```

```

00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }

```

References [Markov::Model< NodeStorageType >::Import\(\)](#).

Here is the call graph for this function:



9.26.2.20 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder [inherited]
```

Sort edges of all nodes in the model ordered by edge weights.

Definition at line 186 of file [model.h](#).

```

00265     for (std::pair<unsigned char, Markov::Node<NodeStorageType>>* const& x : this->nodes) {
00266         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00267         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
00268             Edge<NodeStorageType> *rhs)->bool{
00269             return lhs->EdgeWeight() > rhs->EdgeWeight();
00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00273         //std::cout << "\n";
00274     }
00275     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00276     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00277 }

```

9.26.2.21 RandomWalk()

```
char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]
```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-involve randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```

Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}

```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file [model.h](#).

```

00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext (randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL) {
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue ();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

9.26.2.22 Save()

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
```

Export model to file.

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

`std::ofstream*` of the exported file.

Definition at line 106 of file [markovPasswords.cpp](#).

```

00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export (exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.26.2.23 StarterNode()

`Node< char >* Markov::Model< char >::StarterNode () [inline], [inherited]`
Return starter Node.

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).
00171 { **return starterNode;** }

9.26.2.24 Train()

```
void Markov::API::ModelMatrix::Train (
    const char * datasetFileName,
    char delimiter,
    int threads) [inherited]
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

Construct the matrix when done.

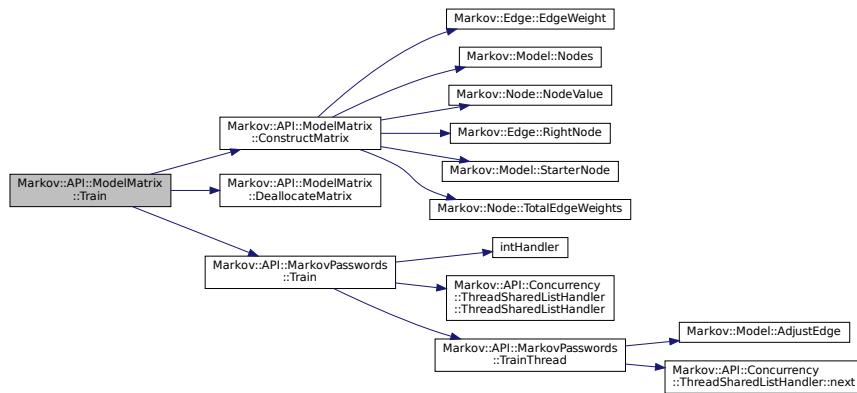
Definition at line 25 of file [modelMatrix.cpp](#).

```
00025
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

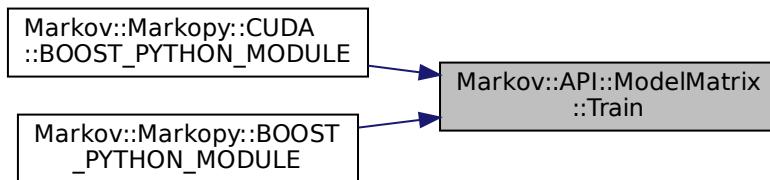
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.26.2.25 TrainThread()

```

void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private], [inherited]
  
```

A single thread invoked by the Train function.

Parameters

<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

Definition at line 85 of file [markovPasswords.cpp](#).

```

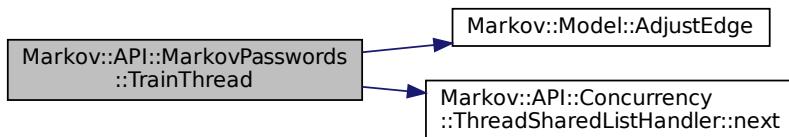
00085
00086     char format_str[] = "%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
00094         char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to-
00097         "%ld,%s"
00098     #else
  
```

```

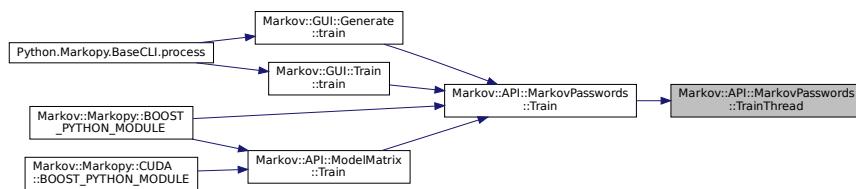
00098     sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100     this->AdjustEdge((const char*)linebuf, oc);
00101     delete linebuf;
00102 }
00103 }
```

References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), [keepRunning](#), and [Markov::API::Concurrency::ThreadSharedListHandler::next](#).
Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.26.3 Member Data Documentation

9.26.3.1 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]`
Definition at line [123](#) of file [markovPasswords.h](#).

9.26.3.2 edgeMatrix

`char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]`
2-D Character array for the edge Matrix (The characters of Nodes)
Definition at line [175](#) of file [modelMatrix.h](#).
Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.26.3.3 edges

`std::vector<Edge<char *>> Markov::Model< char >::edges [private], [inherited]`
A list of all edges in this model.
Definition at line [204](#) of file [model.h](#).

9.26.3.4 matrixIndex

char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]
to hold the Matrix index (To hold the orders of 2-D arrays)
Definition at line 190 of file [modelMatrix.h](#).
Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.26.3.5 matrixSize

int Markov::API::ModelMatrix::matrixSize [protected], [inherited]
to hold Matrix size
Definition at line 185 of file [modelMatrix.h](#).
Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#), and [Markov::API::CUDA::CUDAMoo](#).

9.26.3.6 modelSavefile

std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]
Dataset file input of our system

Definition at line 124 of file [markovPasswords.h](#).

9.26.3.7 nodes

std::map<char , Node<char >*> Markov::Model< char >::nodes [private], [inherited]
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
Definition at line 193 of file [model.h](#).

9.26.3.8 outputFile

std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]
File to save model of our system

Definition at line 125 of file [markovPasswords.h](#).

9.26.3.9 ready

bool Markov::API::ModelMatrix::ready [protected], [inherited]
True when matrix is constructed. False if not.
Definition at line 200 of file [modelMatrix.h](#).
Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::ModelMatrix\(\)](#).

9.26.3.10 starterNode

Node<char >* Markov::Model< char >::starterNode [private], [inherited]
Starter Node of this model.
Definition at line 198 of file [model.h](#).

9.26.3.11 totalEdgeWeights

long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]

Array of the Total Edge Weights.

Definition at line 195 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.26.3.12 valueMatrix

long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]

2-d Integer array for the value Matrix (For the weights of Edges)

Definition at line 180 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

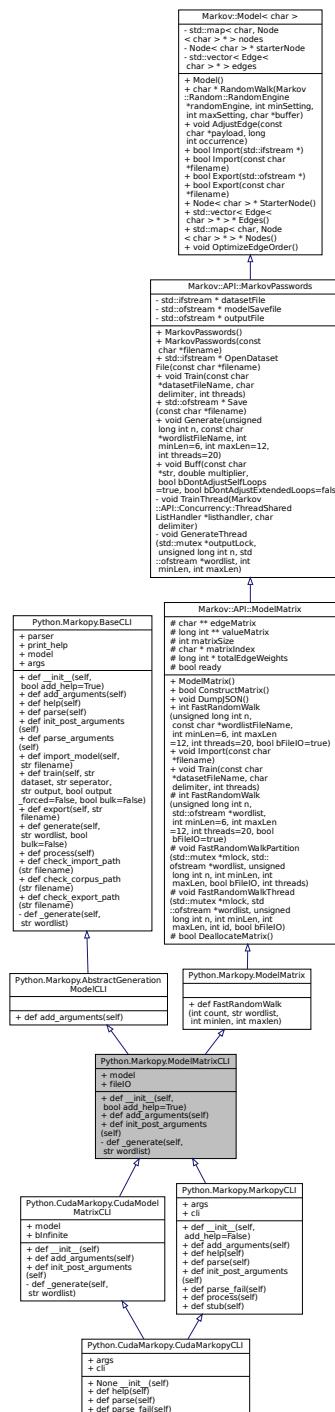
The documentation for this class was generated from the following file:

- [Markopy/Markopy/src/CLI/mm.py](#)

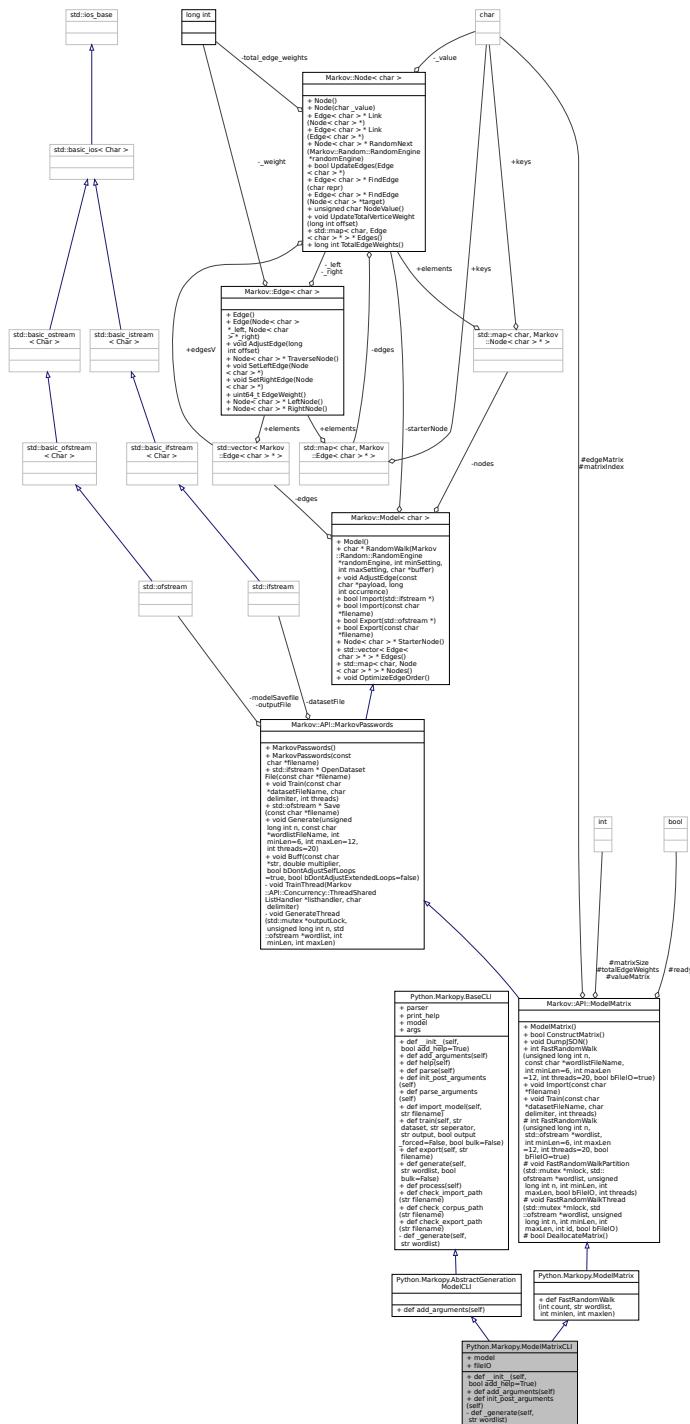
9.27 Python.Markopy.ModelMatrixCLI Class Reference

Extension of Python.Markopy.Base.BaseCLI for [Markov::API::ModelMatrix](#).

Inheritance diagram for Python.Markopy.ModelMatrixCLI:



Collaboration diagram for Python.Markopy.ModelMatrixCLI:



Public Member Functions

- def `__init__` (self, bool add_help=True)
initialize base CLI
 - def `add_arguments` (self)
 - def `init_post_arguments` (self)
 - def `help` (self)
 - def `parse` (self)

- def `parse_arguments` (self)
- def `import_model` (self, str filename)

Import a model file.
- def `train` (self, str dataset, str seperator, str output, bool output_forced=False, bool bulk=False)

Train a model via CLI parameters.
- def `export` (self, str filename)

Export model to a file.
- def `generate` (self, str wordlist, bool bulk=False)

Generate strings from the model.
- def `process` (self)

Process parameters for operation.
- def `FastRandomWalk` (int count, str wordlist, int minlen, int maxlen)

Random walk on the Matrix-reduced `Markov::Model`.
- bool `ConstructMatrix` ()

Construct the related Matrix data for the model.
- void `DumpJSON` ()

Debug function to dump the model to a JSON file.
- void `Import` (const char *filename)

Open a file to import with filename, and call `bool Model::Import` with `std::ifstream`.
- bool `Import` (`std::ifstream` *)

Import a file to construct the model.
- void `Train` (const char *datasetFileName, char delimiter, int threads)

Train the model with the dataset file.
- `std::ifstream` * `OpenDatasetFile` (const char *filename)

Open dataset file and return the `ifstream` pointer.
- `std::ofstream` * `Save` (const char *filename)

Export model to file.
- void `Generate` (unsigned long int n, const char *wordlistFileName, int minLen=6, int maxLen=12, int threads=20)

Call `Markov::Model::RandomWalk` n times, and collect output.
- void `Buff` (const char *str, double multiplier, bool bDontAdjustSelfLoops=true, bool bDontAdjustExtendedLoops=false)

Buff expression of some characters in the model.
- char * `RandomWalk` (`Markov::Random::RandomEngine` *randomEngine, int minSetting, int maxSetting, char *buffer)

Do a random walk on this model.
- void `AdjustEdge` (const char *payload, long int occurrence)

Adjust the model with a single string.
- bool `Export` (`std::ofstream` *)

Export a file of the model.
- bool `Export` (const char *filename)

Open a file to export with filename, and call `bool Model::Export` with `std::ofstream`.
- `Node<char>*` `StarterNode` ()

Return starter Node.
- `std::vector<Edge<char>>*>* Edges` ()

Return a vector of all the edges in the model.
- `std::map<char, Node<char>>*>* Nodes` ()

Return starter Node.
- void `OptimizeEdgeOrder` ()

Sort edges of all nodes in the model ordered by edge weights.

Static Public Member Functions

- def `check_import_path` (str filename)
check import path for validity
- def `check_corpus_path` (str filename)
check import path for validity
- def `check_export_path` (str filename)
check import path for validity

Public Attributes

- `model`
- `fileIO`
- `parser`
- `print_help`
- `args`

Protected Member Functions

- int `FastRandomWalk` (unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12, int threads=20, bool bFileIO=true)
Random walk on the Matrix-reduced `Markov::Model`.
- void `FastRandomWalkPartition` (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads)
A single partition of `FastRandomWalk` event.
- void `FastRandomWalkThread` (std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int minLen, int maxLen, int id, bool bFileIO)
A single thread of a single partition of `FastRandomWalk`.
- bool `DeallocateMatrix` ()
Deallocate matrix and make it ready for re-construction.

Protected Attributes

- char ** `edgeMatrix`
2-D Character array for the edge Matrix (The characters of Nodes)
- long int ** `valueMatrix`
2-d Integer array for the value Matrix (For the weights of Edges)
- int `matrixSize`
to hold Matrix size
- char * `matrixIndex`
to hold the Matrix index (To hold the orders of 2-D arrays')
- long int * `totalEdgeWeights`
Array of the Total Edge Weights.
- bool `ready`
True when matrix is constructed. False if not.

Private Member Functions

- def `_generate` (self, str wordlist)
wrapper for generate function.
- void `TrainThread` (`Markov::API::Concurrency::ThreadSharedListHandler` *listhandler, char delimiter)
A single thread invoked by the Train function.
- void `GenerateThread` (std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int minLen, int maxLen)
A single thread invoked by the Generate function.

Private Attributes

- std::ifstream * **datasetFile**
Dataset file input of our system
- std::ofstream * **modelSavefile**
File to save model of our system
- std::map< char, Node< char > * > **nodes**
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.
- Node< char > * **starterNode**
Starter Node of this model.
- std::vector< Edge< char > * > **edges**
A list of all edges in this model.

9.27.1 Detailed Description

Extension of Python.Markopy.Base.BaseCLI for [Markov::API::ModelMatrix](#).

adds -st/-stdout argument to the command line.

Definition at line 18 of file [mmx.py](#).

9.27.2 Constructor & Destructor Documentation

9.27.2.1 `__init__()`

```
def Python.Markopy.ModelMatrixCLI.__init__ (
    self,
    bool add_help = True )
initialize base CLI
```

Parameters

<code>add_help</code>	decide to overload the help function or not
-----------------------	---

Reimplemented from [Python.Markopy.BaseCLI](#).

Definition at line 27 of file [mmx.py](#).

```
00027     def __init__(self, add_help=True):
00028         """! @brief initialize model with Markov::API::ModelMatrix"""
00029         super().__init__(add_help)
00030         self.model = markopy.ModelMatrix()
00031
```

9.27.3 Member Function Documentation

9.27.3.1 `_generate()`

```
def Python.Markopy.ModelMatrixCLI._generate (
    self,
    str wordlist ) [private]
```

wrapper for generate function.

This can be overloaded by other models

Parameters

<code>wordlist</code>	filename to generate to
-----------------------	-------------------------

Reimplemented from [Python.Markopy.BaseCLI](#).

Reimplemented in [Python.CudaMarkopy.CudaModelMatrixCLI](#).

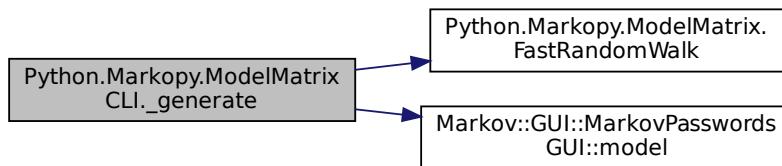
Definition at line 40 of file [mmx.py](#).

```
00040     def _generate(self, wordlist : str, ):
00041         self.model.FastRandomWalk(int(self.args.count), wordlist, int(self.args.min),
00042             int(self.args.max), int(self.args.threads), self.fileIO)
```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.ModelMatrix.FastRandomWalk\(\)](#), [Python.Markopy.ModelMatrixCLI.fileIO](#), [Python.CudaMarkopy.CudaModelMatrixCLI](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.2 add_arguments()

```
def Python.Markopy.ModelMatrixCLI.add_arguments (
    self )
```

Reimplemented from [Python.Markopy.AbstractGenerationModelCLI](#).

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

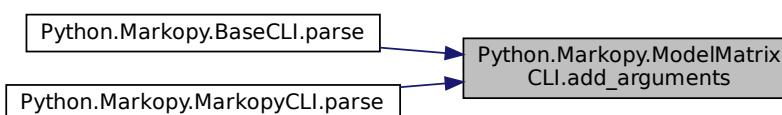
Definition at line 32 of file [mmx.py](#).

```
00032     def add_arguments(self):
00033         super().add_arguments()
00034         self.parser.add_argument("-st", "--stdout", action="store_true", help="Stdout mode")
00035
```

References [Python.Markopy.BaseCLI.parser](#).

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.27.3.3 AdjustEdge()

```
void Markov::Model< char >::AdjustEdge (
    const NodeStorageType * payload,
    long int occurrence ) [inherited]
```

Adjust the model with a single string.

Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from current node to the next, until NULL character is reached.

Then, update the edge EdgeWeight from current node, to the terminator node.

This function is used for training purposes, as it can be used for adjusting the model with each line of the corpus file.

Example Use: Create an empty model and train it with string: "testdata"

```
Markov::Model<char> model;
char test[] = "testdata";
model.AdjustEdge(test, 15);
```

Parameters

string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 109 of file [model.h](#).

```
00337
00338     NodeStorageType p = payload[0];
00339     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00340     Markov::Edge<NodeStorageType>* e;
00341     int i = 0;
00342
00343     if (p == 0) return;
00344     while (p != 0) {
00345         e = curnode->FindEdge(p);
00346         if (e == NULL) return;
00347         e->AdjustEdge(occurrence);
00348         curnode = e->RightNode();
00349         p = payload[++i];
00350     }
00351
00352     e = curnode->FindEdge('\xff');
00353     e->AdjustEdge(occurrence);
00354     return;
00355 }
```

9.27.3.4 Buff()

```
void Markov::API::MarkovPasswords::Buff (
    const char * str,
    double multiplier,
    bool bDontAdjustSelfLoops = true,
    bool bDontAdjustExtendedLoops = false ) [inherited]
```

Buff expression of some characters in the model.

Parameters

str	A string containing all the characters to be buffed
multiplier	A constant value to buff the nodes with.
bDontAdjustSelfEdges	Do not adjust weights if target node is same as source node
bDontAdjustExtendedLoops	Do not adjust if both source and target nodes are in first parameter

Definition at line 153 of file [markovPasswords.cpp](#).

```
00153
00154     std::string buffstr(str);
00155     std::map< char, Node< char > * > *nodes;
00156     std::map< char, Edge< char > * > *edges;
00157     nodes = this->Nodes();
00158     int i=0;
00159     for (auto const& [repr, node] : *nodes) {
```

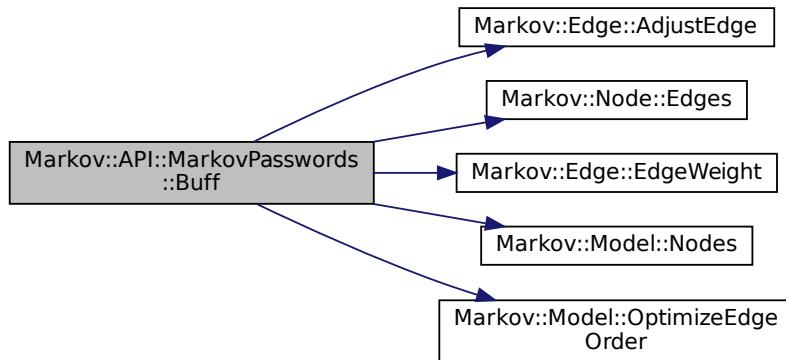
```

00160     edges = node->Edges();
00161     for (auto const& [targetrepr, edge] : *edges){
00162         if(buffstr.find(targetrepr) != std::string::npos) {
00163             if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00164             if(buffstr.find(repr) != std::string::npos) {
00165                 continue;
00166             }
00167         }
00168         long int weight = edge->EdgeWeight();
00169         weight = weight*multiplier;
00170         edge->AdjustEdge(weight);
00171     }
00172 }
00173 }
00174 i++;
00175 }
00176 }
00177 this->OptimizeEdgeOrder();
00178 }
00179 }
```

References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.5 check_corpus_path()

```

def Python.Markopy.BaseCLI.check_corpus_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 181 of file [base.py](#).

```
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185         """
00186
00187         if(not os.path.isfile(filename)):
00188             return False
00189         return True
00190
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.27.3.6 check_export_path()

```
def Python.Markopy.BaseCLI.check_export_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

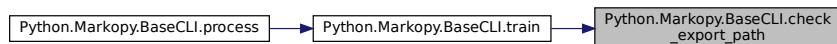
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 192 of file [base.py](#).

```
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196         """
00197
00198         if(filename and os.path.isfile(filename)):
00199             return True
00200         return True
00201
```

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the caller graph for this function:



9.27.3.7 check_import_path()

```
def Python.Markopy.BaseCLI.check_import_path (
    str filename ) [static], [inherited]
check import path for validity
```

Parameters

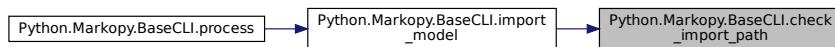
<i>filename</i>	filename to check
-----------------	-------------------

Definition at line 169 of file [base.py](#).

```
00169     def check_import_path(filename : str):
00170         """
00171             @brief check import path for validity
00172             @param filename filename to check
00173         """
00174
00175         if(not os.path.isfile(filename)):
00176             return False
00177         else:
00178             return True
00179
```

Referenced by [Python.Markopy.BaseCLI.import_model\(\)](#).

Here is the caller graph for this function:

**9.27.3.8 ConstructMatrix()**

`bool Markov::API::ModelMatrix::ConstructMatrix () [inherited]`

Construct the related Matrix data for the model.

This operation can be used after importing/training to allocate and populate the matrix content.

this will initialize: `char** edgeMatrix` -> a 2D array of mapping left and right connections of each edge. `long int **valueMatrix` -> a 2D array representing the edge weights. `int matrixSize` -> Size of the matrix, aka total number of nodes. `char* matrixIndex` -> order of nodes in the model `long int *totalEdgeWeights` -> total edge weights of each Node.

Returns

True if constructed. False if already constructed.

Definition at line 31 of file [modelMatrix.cpp](#).

```
00031                                     {
00032     if(this->ready) return false;
00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 for(int k=0;k<this->matrixSize;k++){
00064                     this->valueMatrix[i][k] = 0;
00065                     this->edgeMatrix[i][k] = 255;
00066                 }
00067             }
00068         }
00069     }
00070 }
```

```

00063     this->valueMatrix[i][j] = 0;
00064     this->edgeMatrix[i][j] = 255;
00065     }else if(j==(this->matrixSize-1)) {
00066         this->valueMatrix[i][j] = 0;
00067         this->edgeMatrix[i][j] = 255;
00068     }else{
00069         this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight();
00070         this->edgeMatrix[i][j] = node->edgesV[j]->RightNode()->NodeValue();
00071     }
00072 }
00073     i++;
00074 }
00075     this->ready = true;
00076     return true;
00077 //this->DumpJSON();
00078 }
```

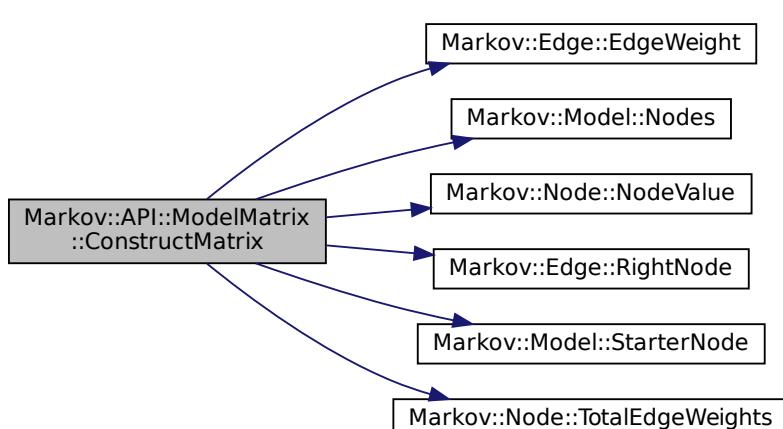
References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::API::ModelMatrix::Markov::API::ModelMatrix::matrixSize](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), [Markov::Node< storageType >::NodeValue\(\)](#)

[Markov::API::ModelMatrix::ready](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::StarterNode\(\)](#)

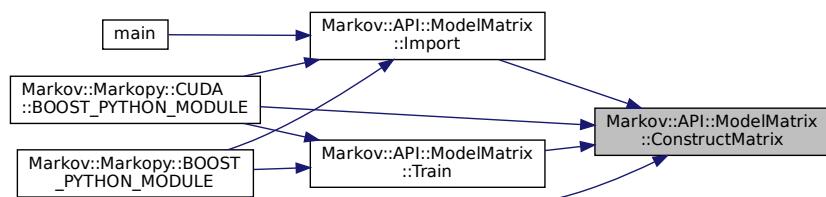
[Markov::API::ModelMatrix::totalEdgeWeights](#), [Markov::Node< storageType >::TotalEdgeWeights\(\)](#), and [Markov::API::ModelMatrix::v](#)

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.9 DeallocateMatrix()

```
bool Markov::API::ModelMatrix::DeallocateMatrix ( ) [protected], [inherited]
```

Deallocate matrix and make it ready for re-construction.

Returns

True if deallocated. False if matrix was not initialized

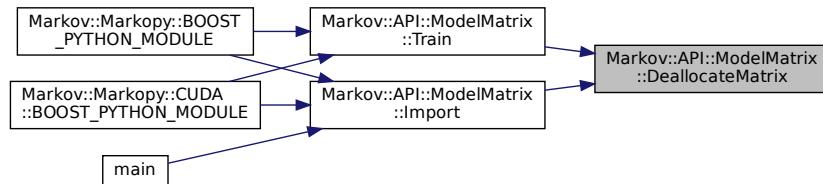
Definition at line 81 of file [modelMatrix.cpp](#).

```
00081     if(!this->ready) return false;
00082     delete[] this->matrixIndex;
00083     delete[] this->totalEdgeWeights;
00084
00085     for(int i=0;i<this->matrixSize;i++) {
00086         delete[] this->edgeMatrix[i];
00087     }
00088     delete[] this->edgeMatrix;
00089
00090     for(int i=0;i<this->matrixSize;i++) {
00091         delete[] this->valueMatrix[i];
00092     }
00093     delete[] this->valueMatrix;
00094
00095     this->matrixSize = -1;
00096     this->ready = false;
00097     return true;
00098 }
00099 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::API::ModelMatrix::ready](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::API::ModelMatrix::Import\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the caller graph for this function:

**9.27.3.10 DumpJSON()**

`void Markov::API::ModelMatrix::DumpJSON () [inherited]`

Debug function to dump the model to a JSON file.

Might not work 100%. Not meant for production use.

Definition at line 101 of file [modelMatrix.cpp](#).

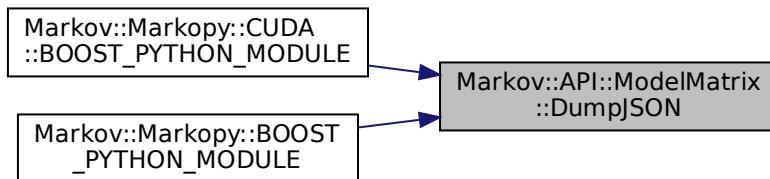
```
00101     {
00102
00103     std::cout << "{\n      \"index\": \"";
00104     for(int i=0;i<this->matrixSize;i++) {
00105         if(this->matrixIndex[i]=='\r') std::cout << "\\\\";";
00106         else if(this->matrixIndex[i]=='\n') std::cout << "\\\\\\\";";
00107         else if(this->matrixIndex[i]==0) std::cout << "\\\\x00";
00108         else if(i==0) std::cout << "\\\\xff";
00109         else if(this->matrixIndex[i]=='\r\n') std::cout << "\\n";
00110         else std::cout << this->matrixIndex[i];
00111     }
00112     std::cout <<
00113     "\\",\\n"
00114     "  \"edgemap\": {\n";
00115
00116     for(int i=0;i<this->matrixSize;i++) {
00117         if(this->matrixIndex[i]=='\r') std::cout << "      \\\\\\\\"": "[";
00118         else if(this->matrixIndex[i]=='\n') std::cout << "      \\\\\\\\"": "[";
00119         else if(this->matrixIndex[i]==0) std::cout << "      \\\\\\\\"x00": "[";
00120         else if(this->matrixIndex[i]<0) std::cout << "      \\\\\\\\"xf": "[";
00121         else std::cout << "      \" " << this->matrixIndex[i] << "": "[";
00122         for(int j=0;j<this->matrixSize;j++) {
00123             if(this->edgeMatrix[i][j]=='\r') std::cout << "        \\\\\\\\"=\"";
00124             else if(this->edgeMatrix[i][j]=='\n') std::cout << "        \\\\\\\\"": "[";
00125             else if(this->edgeMatrix[i][j]==0) std::cout << "        \\\\\\\\"x00": "[";
00126             else if(this->edgeMatrix[i][j]<0) std::cout << "        \\\\\\\\"xf": "[";
00127             else if(this->matrixIndex[i]=='\r\n') std::cout << "        \\\\\\\\"n": "[";
00128         }
00129     }
00130     std::cout << "    }";
00131 }
```

```

00128         else std::cout << "\\" << this->edgeMatrix[i][j] << "\\";
00129         if(j!=this->matrixSize-1) std::cout << ", ";
00130     }
00131     std::cout << "],\n";
00132 }
00133 std::cout << "},\n";
00134
00135 std::cout << "\" weightmap\": {\n";
00136 for(int i=0;i<this->matrixSize;i++){
00137     if(this->matrixIndex[i]=='"') std::cout << "      \"\\\\\"\\": [";
00138     else if(this->matrixIndex[i]=='\\') std::cout << "      \"\\\\\\\\\\": [";
00139     else if(this->matrixIndex[i]==0) std::cout << "      \"\\\\\\\\\\x00\\": [";
00140     else if(this->matrixIndex[i]<0) std::cout << "      \"\\\\\\\\\\xff\\": [";
00141     else std::cout << "      \" " << this->matrixIndex[i] << "\": [";
00142
00143     for(int j=0;j<this->matrixSize;j++){
00144         std::cout << this->valueMatrix[i][j];
00145         if(j!=this->matrixSize-1) std::cout << ", ";
00146     }
00147     std::cout << "],\n";
00148 }
00149 std::cout << " }\n}\n";
00150 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the caller graph for this function:



9.27.3.11 Edges()

```
std::vector<Edge<char *>>* Markov::Model< char >::Edges ( ) [inline], [inherited]
```

Return a vector of all the edges in the model.

Returns

vector of edges

Definition at line 176 of file [model.h](#).

```
00176 { return &edges; }
```

9.27.3.12 Export() [1/2]

```
bool Markov::Model< char >::Export (
    const char * filename ) [inherited]
```

Open a file to export with filename, and call bool Model::Export with std::ofstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Export file to filename

```
Markov::Model<char> model;
```

```
model.Export("test.mdl");
```

Definition at line 166 of file [model.h](#).

```

00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

9.27.3.13 export()

```
def Python.Markopy.BaseCLI.export (
    self,
    str filename ) [inherited]
```

Export model to a file.

Parameters

<i>filename</i>	filename to export to
-----------------	-----------------------

Definition at line 138 of file [base.py](#).

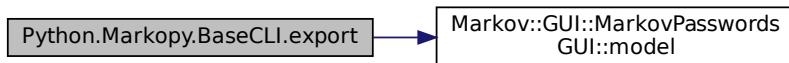
```

00138     def export(self, filename : str):
00139         """
00140             @brief Export model to a file
00141             @param filename filename to export to
00142         """
00143         self.model.Export(filename)
00144 
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.14 Export() [2/2]

```
bool Markov::Model< char >::Export (
    std::ofstream * f ) [inherited]
```

Export a file of the model.

File contains a list of edges. Format is: Left_repr;EdgeWeight;right_repr. For more information on the format, check out the project wiki or github readme.

Iterate over this vertices, and their edges, and write them to file.

Returns

True if successful, False for incomplete models.

Example Use: Export file to ofstream

```
Markov::Model<char> model;
std::ofstream file("test.mdl");
model.Export(&file);
```

Definition at line 155 of file model.h.

```
00288
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <
00293         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <
00294         "\n";
00295     }
00296     return true;
00297 }
```

9.27.3.15 FastRandomWalk() [1/3]

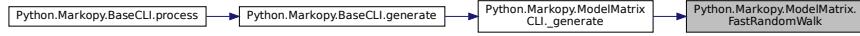
```
def Python.Markopy.ModelMatrix.FastRandomWalk (
    int count,
    str wordlist,
    int minlen,
    int maxlen ) [inherited]
```

Definition at line 48 of file mm.py.

```
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass
```

Referenced by [Python.Markopy.ModelMatrixCLI._generate\(\)](#).

Here is the caller graph for this function:

**9.27.3.16 FastRandomWalk() [2/3]**

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If n>50M, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to

Parameters

<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

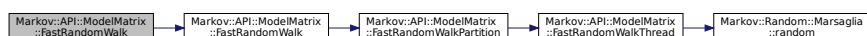
Definition at line 217 of file [modelMatrix.cpp](#).

```
00217
00218     std::ofstream wordlist;
00219     if(bFileIO)
00220         wordlist.open(wordlistFileName);
00221     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00222     return 0;
00223 }
```

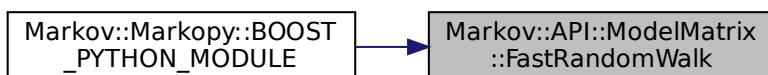
References [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.17 FastRandomWalk() [3/3]

```
int Markov::API::ModelMatrix::FastRandomWalk (
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20,
    bool bFileIO = true ) [protected], [inherited]
```

Random walk on the Matrix-reduced [Markov::Model](#).

This has an O(N) Memory complexity. To limit the maximum usage, requests with $n > 50M$ are partitioned using [Markov::API::ModelMatrix::FastRandomWalkPartition](#).

If $n > 50M$, threads are going to be synced, files are going to be flushed, and buffers will be reallocated every 50M generations. This comes at a minor performance penalty.

While it has the same functionality, this operation reduces [Markov::API::MarkovPasswords::Generate](#) runtime by %96.5

This function has deprecated [Markov::API::MarkovPasswords::Generate](#), and will eventually replace it.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

```
Markov::API::ModelMatrix mp;
mp.Import("models/finished.mdl");
mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
```

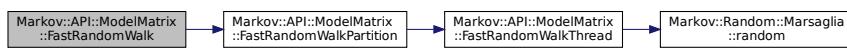
Definition at line 204 of file [modelMatrix.cpp](#).

```
00204
00205
00206
00207     std::mutex mlock;
00208     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
00209         threads);
00210     else{
00211         int numberOfPartitions = n/50000000ull;
00212         for(int i=0;i<numberOfPartitions;i++)
00213             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
00214             threads);
00215     }
00216     return 0;
00217 }
```

References [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.18 FastRandomWalkPartition()

```
void Markov::API::ModelMatrix::FastRandomWalkPartition (
    std::mutex * mlock,
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    bool bFileIO,
    int threads ) [protected], [inherited]
```

A single partition of FastRandomWalk event.

Since FastRandomWalk has to allocate its output buffer before operation starts and writes data in chunks, large n parameters would lead to huge memory allocations. **Without Partitioning:**

- 50M results 12 characters max -> 550 Mb Memory allocation

- 5B results 12 characters max -> 55 Gb Memory allocation
- 50B results 12 characters max -> 550GB Memory allocation

Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.

Parameters

<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 225 of file [modelMatrix.cpp](#).

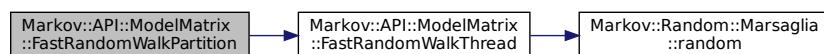
```

00225
00226
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++) {
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00235                                         mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00236         id++;
00237     }
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00239                                         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00240     for(int i=0;i<threads;i++) {
00241         threadsV[i]->join();
00242     }
00243 }
```

References [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.19 FastRandomWalkThread()

```
void Markov::API::ModelMatrix::FastRandomWalkThread (
    std::mutex * mlock,
```

```
    std::ofstream * wordlist,
    unsigned long int n,
    int minLen,
    int maxLen,
    int id,
    bool bFileIO ) [protected], [inherited]
```

A single thread of a single partition of FastRandomWalk.

A FastRandomWalkPartition will initiate as many of this function as requested.

This function contains the bulk of the generation algorithm.

Parameters

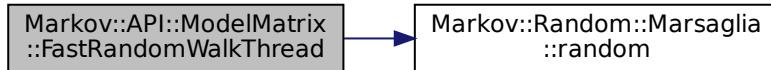
<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>id</i>	- DEPRECATED Thread id - No longer used
<i>bFileIO</i>	- If false, filename will be ignored and will output to stdout.

Definition at line 153 of file [modelMatrix.cpp](#).

```
00153
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++){
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0){
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179             if (len >= maxLen) break;
00180             else if ((next < 0) && (len < minLen)) continue;
00181             else if (next < 0) break;
00182             cur = next;
00183             res[bufferctr + len++] = cur;
00184         }
00185     }
00186     res[bufferctr + len++] = '\n';
00187     bufferctr+=len;
00188
00189 }
00190 if(bFileIO){
00191     mlock->lock();
00192     *wordlist << res;
00193     mlock->unlock();
00194 }else{
00195     mlock->lock();
00196     std::cout << res;
00197     mlock->unlock();
00198 }
00199 delete res;
00200 }
```

References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::API::ModelMatrix::matrixIndex](#), [Markov::API::ModelMatrix::matrixSize](#), [Markov::Random::Marsaglia::random\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::valueMatrix](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalkPartition\(\)](#).
 Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.20 generate()

```
def Python.Markopy.BaseCLI.generate (
    self,
    str wordlist,
    bool bulk = False ) [inherited]
```

Generate strings from the model.

Parameters

<i>model</i>	model instance
<i>wordlist</i>	wordlist filename
<i>bulk</i>	marks bulk operation with directories

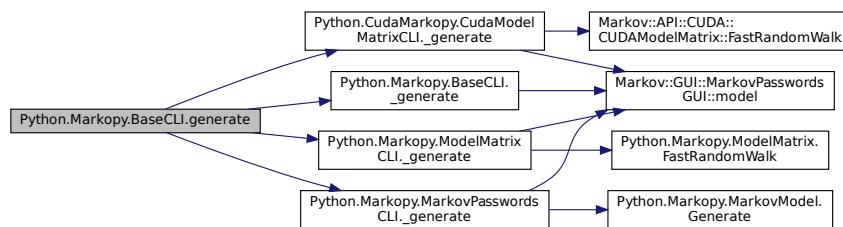
Definition at line 145 of file [base.py](#).

```
00145     def generate(self, wordlist : str, bulk : bool=False):
00146         """
00147             @brief Generate strings from the model
00148             @param model: model instance
00149             @param wordlist wordlist filename
00150             @param bulk marks bulk operation with directories
00151         """
00152         if not (wordlist or self.args.count):
00153             logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156         if(bulk and os.path.isfile(wordlist)):
00157             logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158             self._generate(wordlist)
```

References [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.CudaMarkopy.CudaMarkopy](#), [Python.Markopy.BaseCLI.args](#), and [Python.Markopy.MarkopyCLI.args](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.21 Generate()

```

void Markov::API::MarkovPasswords::Generate (
    unsigned long int n,
    const char * wordlistFileName,
    int minLen = 6,
    int maxLen = 12,
    int threads = 20 ) [inherited]
  
```

Call [Markov::Model::RandomWalk](#) n times, and collect output.

Generate from model and write results to a file. a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Deprecated See [Markov::API::MatrixModel::FastRandomWalk](#) for more information.

Parameters

<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>threads</i>	- number of OS threads to spawn

Definition at line 118 of file [markovPasswords.cpp](#).

```

0018
0019     {
0020         char* res;
0021         char print[100];
0022         std::ofstream wordlist;
0023         wordlist.open(wordlistFileName);
0024         std::mutex mlock;
0025         int iterationsPerThread = n/threads;
0026         int iterationsCarryOver = n%threads;
0027         std::vector<std::thread*> threadsV;
0028         for(int i=0;i<threads;i++) {
0029             threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
0030             &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
  
```

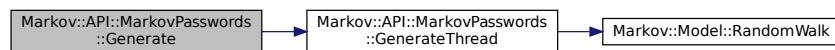
```

00129     }
00130
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
```

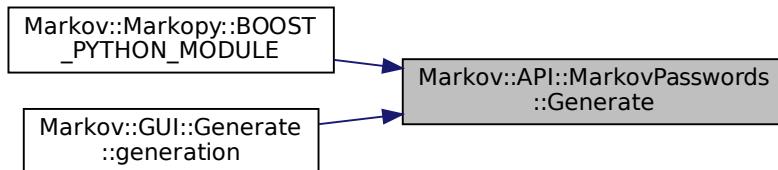
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Referenced by [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [Markov::GUI::Generate::generation\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.22 GenerateThread()

```

void Markov::API::MarkovPasswords::GenerateThread (
    std::mutex * outputLock,
    unsigned long int n,
    std::ofstream * wordlist,
    int minLen,
    int maxLen ) [private], [inherited]
```

A single thread invoked by the Generate function.

DEPRECATED: See [Markov::API::MatrixModel::FastRandomWalkThread](#) for more information. This has been replaced with a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5 on average.

Parameters

<i>outputLock</i>	- shared mutex lock to lock during output operation. Prevents race condition on write.
<i>n</i>	number of lines to be generated by this thread
<i>wordlist</i>	wordlistfile
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate

Definition at line 140 of file [markovPasswords.cpp](#).

```

00140
00141     char* res = new char[maxLen+5];
```

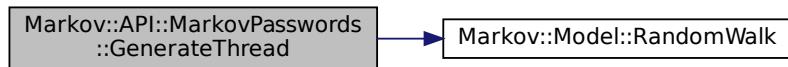
```

00142     if(n==0)  return;
00143
00144     Markov::Random::Marsaglia MarsagliaRandomEngine;
00145     for (int i = 0; i < n; i++) {
00146         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00147         outputLock->lock();
00148         *wordlist << res << "\n";
00149         outputLock->unlock();
00150     }
00151 }
```

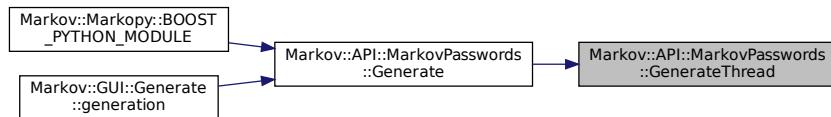
References [Markov::Model< NodeStorageType >::RandomWalk\(\)](#).

Referenced by [Markov::API::MarkovPasswords::Generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.23 help()

```
def Python.Markopy.BaseCLI.help (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

Definition at line 51 of file [base.py](#).

```
00051     def help(self):
00052         """@brief Handle help strings. Defaults to argparse's help"""
00053         self.print_help()
00054
```

References [Python.Markopy.BaseCLI.print_help](#).

Referenced by [Python.Markopy.MarkopyCLI.add_arguments\(\)](#).

Here is the caller graph for this function:



9.27.3.24 Import() [1/2]

```
void Markov::API::ModelMatrix::Import (
    const char * filename ) [inherited]
```

Open a file to import with filename, and call bool Model::Import with std::ifstream.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file with filename

```
Markov::Model<char> model;
model.Import("test.mdl");
```

Construct the matrix when done.

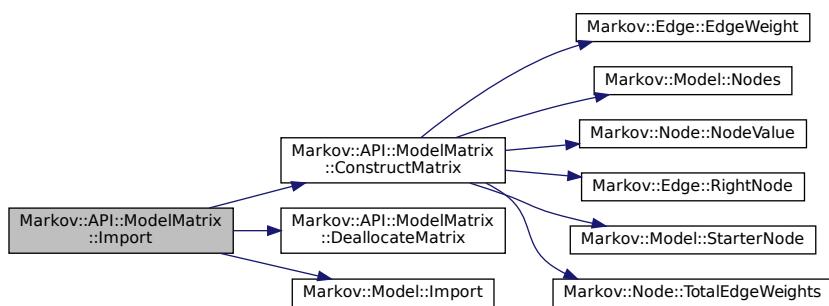
Definition at line 19 of file [modelMatrix.cpp](#).

```
00019
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

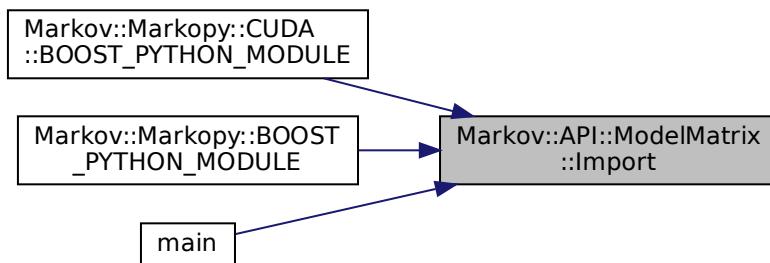
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**9.27.3.25 Import() [2/2]**

```
bool Markov::Model< char >::Import (
    std::ifstream * f )  [inherited]
```

Import a file to construct the model.

File contains a list of edges. For more info on the file format, check out the wiki and github readme pages. Format is: Left_repr;EdgeWeight;right_repr

Iterate over this list, and construct nodes and edges accordingly.

Returns

True if successful, False for incomplete models or corrupt file formats

Example Use: Import a file from ifstream

```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

Definition at line 126 of file model.h.

```
00216     std::string cell;                                {
00217     std::string cell;
00218     char src;
00219     char target;
00220     long int oc;
00221
00222     while (std::getline(*f, cell)) {
00223         //std::cout << "cell: " << cell << std::endl;
00224         src = cell[0];
00225         target = cell[cell.length() - 1];
00226         char* j;
00227         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00228         //std::cout << "oc << "\n";
00229         Markov::Node<NodeStorageType>* srcN;
00230         Markov::Node<NodeStorageType>* targetN;
00231         Markov::Edge<NodeStorageType>* e;
00232         if (this->nodes.find(src) == this->nodes.end()) {
00233             srcN = new Markov::Node<NodeStorageType>(src);
00234             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00235             //std::cout << "Creating new node at start.\n";
00236         }
00237         else {
00238             srcN = this->nodes.find(src)->second;
00239         }
00240
00241         if (this->nodes.find(target) == this->nodes.end()) {
00242             targetN = new Markov::Node<NodeStorageType>(target);
00243             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00244             //std::cout << "Creating new node at end.\n";
00245         }
00246         else {
00247             targetN = this->nodes.find(target)->second;
00248         }
00249         e = srcN->Link(targetN);
00250         e->AdjustEdge(oc);
00251         this->edges.push_back(e);
00252
00253         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00254         int(targetN->NodeValue()) << "\n";
00255
00256     }
00257
00258     this->OptimizeEdgeOrder();
00259
00260     return true;
00261 }
```

9.27.3.26 import_model()

```
def Python.Markopy.BaseCLI.import_model (
    self,
    str filename ) [inherited]
```

Import a model file.

Parameters

<i>filename</i>	filename to import
-----------------	--------------------

Definition at line 77 of file base.py.

```
00077     def import_model(self, filename : str):
00078         """
00079             @brief Import a model file
00080             @param filename filename to import
00081         """
00082         logging pprint("Importing model file.", 1)
00083
00084         if not self.check_import_path(filename):
```

```

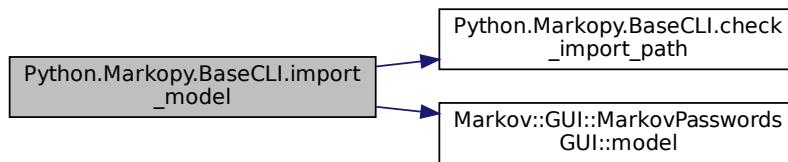
00085         logging pprint(f"Model file at {filename} not found. Check the file path, or working
00086         directory")
00087         return False
00088     self.model.Import(filename)
00089     logging pprint("Model imported successfully.", 2)
00090     return True
00091
00092
00093

```

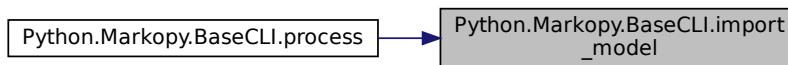
References [Python.Markopy.BaseCLI.check_import_path\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.27 init_post_arguments()

```

def Python.Markopy.ModelMatrixCLI.init_post_arguments (
    self )

```

Reimplemented from [Python.Markopy.BaseCLI](#).

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaModelMatrixCLI](#).

Definition at line 36 of file [mmx.py](#).

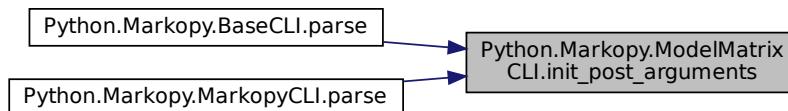
```

00036     def init_post_arguments(self):
00037         super().init_post_arguments()
00038         self.fileIO = not self.args.stdout
00039

```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.27.3.28 Nodes()

```
std::map<char , Node<char *>>* Markov::Model< char >::Nodes ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 181 of file [model.h](#).

```
00181 { return &nodes; }
```

9.27.3.29 OpenDatasetFile()

```
std::ifstream * Markov::API::MarkovPasswords::OpenDatasetFile (
    const char * filename ) [inherited]
```

Open dataset file and return the ifstream pointer.

Parameters

<i>filename</i>	- Filename to open
-----------------	--------------------

Returns

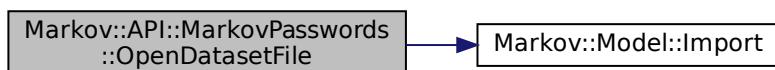
*ifstream** to the the dataset file

Definition at line 51 of file [markovPasswords.cpp](#).

```
00051
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
```

References [Markov::Model< NodeStorageType >::Import\(\)](#).

Here is the call graph for this function:



9.27.3.30 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder [inherited]
Sort edges of all nodes in the model ordered by edge weights.
```

Definition at line 186 of file [model.h](#).

```
00265
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType*>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort (x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
00269             Edge<NodeStorageType> *rhs)->bool{
00270                 return lhs->EdgeWeight() > rhs->EdgeWeight();
```

```

00270         });
00271         //for(int i=0;i<x.second->edgesV.size();i++)
00272         // std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00273         //std::cout << "\n";
00274     }
00275     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00276     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00277 }
```

9.27.3.31 parse()

```
def Python.Markopy.BaseCLI.parse (
    self ) [inherited]
```

Reimplemented in [Python.Markopy.MarkopyCLI](#), and [Python.CudaMarkopy.CudaMarkopyCLI](#).

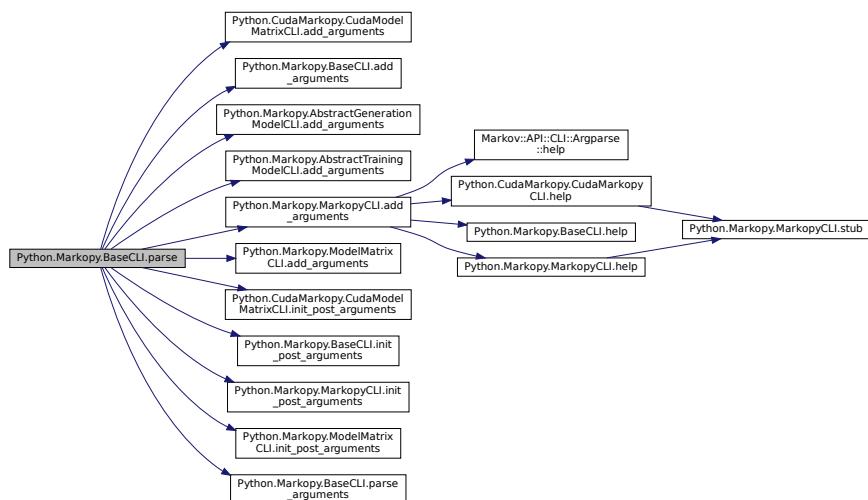
Definition at line 55 of file [base.py](#).

```

00055     def parse(self):
00056         """! @brief add, parse and hook arguments"""
00057         self.add_arguments()
00058         self.parse_arguments()
00059         self.init_post_arguments()
00060 
```

References [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaModelCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.init_post_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.parse_arguments\(\)](#) and [Python.Markopy.BaseCLI.parse_arguments\(\)](#).

Here is the call graph for this function:



9.27.3.32 parse_arguments()

```
def Python.Markopy.BaseCLI.parse_arguments (
    self ) [inherited]
```

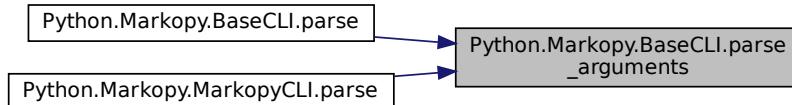
Definition at line 73 of file [base.py](#).

```

00073     def parse_arguments(self):
00074         """! @brief trigger parser"""
00075         self.args = self.parser.parse_known_args() [0]
00076 
```

Referenced by [Python.Markopy.BaseCLI.parse\(\)](#), and [Python.Markopy.MarkopyCLI.parse\(\)](#).

Here is the caller graph for this function:



9.27.3.33 process()

```
def Python.Markopy.BaseCLI.process (
    self ) [inherited]
```

Process parameters for operation.

Reimplemented in [Python.Markopy.MarkopyCLI](#).

Definition at line 202 of file [base.py](#).

```
00202     def process(self):
00203         """
00204             @brief Process parameters for operation
00205         """
00206         if(self.args.bulk):
00207             logging pprint(f"\"Bulk mode operation chosen.\", 4)
00208             if (self.args.mode.lower() == "train"):
00209                 if (os.path.isdir(self.args.output) and not os.path.isfile(self.args.output)) and
00210                     (os.path.isdir(self.args.dataset) and not os.path.isfile(self.args.dataset)):
00211                     corpus_list = os.listdir(self.args.dataset)
00212                     for corpus in corpus_list:
00213                         self.import_model(self.args.input)
00214                         logging pprint(f"\"Training {self.args.input} with {corpus}\", 2)
00215                         output_file_name = corpus
00216                         model_extension = ""
00217                         if "." in self.args.input:
00218                             model_extension = self.args.input.split(".")[-1]
00219                             self.train(f"{self.args.dataset}/{corpus}", self.args.seperator,
00220                                     f"{self.args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00221                         else:
00222                             logging pprint("In bulk training, output and dataset should be a directory.")
00223                             exit(1)
00224
00225             elif (self.args.mode.lower() == "generate"):
00226                 if (os.path.isdir(self.args.wordlist) and not os.path.isfile(self.args.wordlist)) and
00227                     (os.path.isdir(self.args.input) and not os.path.isfile(self.args.input)):
00228                     model_list = os.listdir(self.args.input)
00229                     print(model_list)
00230                     for input in model_list:
00231                         logging pprint(f"\"Generating from {self.args.input}/{input} to
00232                             {self.args.wordlist}/{input}.txt\", 2)
00233                         self.import_model(f"{self.args.input}/{input}")
00234                         model_base = input
00235                         if "." in self.args.input:
00236                             model_base = input.split(".")[-1]
00237                             self.generate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00238                         else:
00239                             logging pprint("In bulk generation, input and wordlist should be directory.")
00240
00241             else:
00242                 self.import_model(self.args.input)
00243                 if (self.args.mode.lower() == "generate"):
00244                     self.generate(self.args.wordlist)
00245
00246             elif(self.args.mode.lower() == "train"):
00247                 self.train(self.args.dataset, self.args.seperator, self.args.output,
00248                         output_forced=True)
00249
00250             elif(self.args.mode.lower() == "combine"):
00251                 self.train(self.args.dataset, self.args.seperator, self.args.output)
00252                 self.generate(self.args.wordlist)
00253
00254             else:
00255                 logging pprint("Invalid mode arguement given.")
```

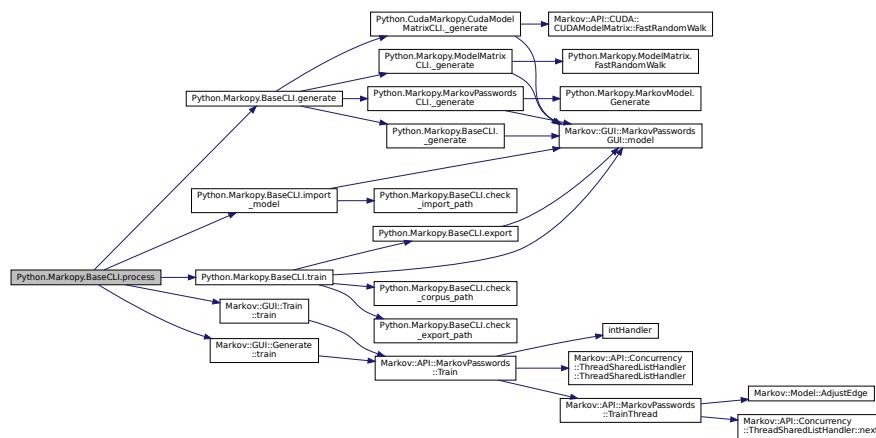
```

00254     logging pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255     exit(5)
00256

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), [Markov::GUI::Generate.train\(\)](#), [Markov::GUI::Train.train\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

Here is the call graph for this function:



9.27.3.34 RandomWalk()

```

char * Markov::Model< char >::RandomWalk (
    Markov::Random::RandomEngine * randomEngine,
    int minSetting,
    int maxSetting,
    NodeStorageType * buffer ) [inherited]

```

Do a random walk on this model.

Start from the starter node, on each node, invoke RandomNext using the random engine on current node, until terminator node is reached. If terminator node is reached before minimum length criteria is reached, ignore the last selection and re-invoke randomNext

If maximum length criteria is reached but final node is not, cut off the generation and proceed to the final node. This function takes [Markov::Random::RandomEngine](#) as a parameter to generate pseudo random numbers from This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne output is higher in entropy, most use cases don't really need super high entropy output, so [Markov::Random::Marsaglia](#) is preferable for better performance.

This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening via maximum length criteria.

Example Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use Marsaglia

```

Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}

```

Parameters

<i>randomEngine</i>	Random Engine to use for the random walks. For examples, see Markov::Random::Mersenne and Markov::Random::Marsaglia
<i>minSetting</i>	Minimum number of characters to generate
<i>maxSetting</i>	Maximum number of character to generate
<i>buffer</i>	buffer to write the result to

Returns

Null terminated string that was generated.

Definition at line 86 of file model.h.

```

00307
00308     Markov::Node<NodeStorageType>* n = this->starterNode;
00309     int len = 0;
00310     Markov::Node<NodeStorageType>* temp_node;
00311     while (true) {
00312         temp_node = n->RandomNext(randomEngine);
00313         if (len >= maxSetting) {
00314             break;
00315         }
00316         else if ((temp_node == NULL) && (len < minSetting)) {
00317             continue;
00318         }
00319
00320         else if (temp_node == NULL) {
00321             break;
00322         }
00323
00324         n = temp_node;
00325
00326         buffer[len++] = n->NodeValue();
00327     }
00328
00329     //null terminate the string
00330     buffer[len] = 0x00;
00331
00332     //do something with the generated string
00333     return buffer; //for now
00334 }
```

9.27.3.35 Save()

```
std::ofstream * Markov::API::MarkovPasswords::Save (
    const char * filename ) [inherited]
```

Export model to file.

Parameters

<i>filename</i>	- Export filename.
-----------------	--------------------

Returns

std::ofstream* of the exported file.

Definition at line 106 of file markovPasswords.cpp.

```

00106
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
```

References [Markov::Model< NodeStorageType >::Export\(\)](#).

Here is the call graph for this function:



9.27.3.36 StarterNode()

```
Node< char *>* Markov::Model< char >::StarterNode ( ) [inline], [inherited]
Return starter Node.
```

Returns

starter node with 00 NodeValue

Definition at line 171 of file [model.h](#).

```
00171 { return starterNode; }
```

9.27.3.37 Train()

```
void Markov::API::ModelMatrix::Train (
    const char * datasetFileName,
    char delimiter,
    int threads ) [inherited]
```

Train the model with the dataset file.

Parameters

<i>datasetFileName</i>	- ifstream* to the dataset. If null, use class member
<i>delimiter</i>	- a character, same as the delimiter in dataset content
<i>threads</i>	- number of OS threads to spawn

```
Markov::API::MarkovPasswords mp;
mp.Import("models/2gram.mdl");
mp.Train("password.corpus");
```

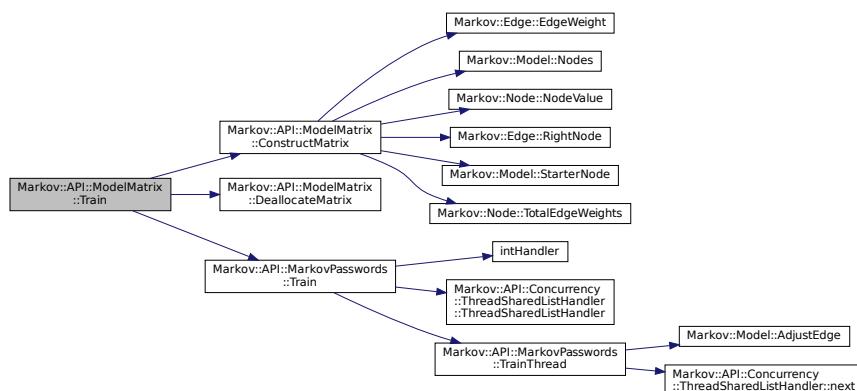
Construct the matrix when done.

Definition at line 25 of file [modelMatrix.cpp](#).

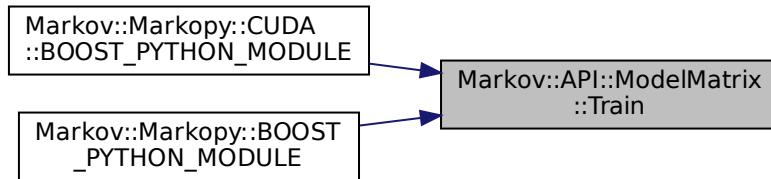
```
00025
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE\(\)](#), and [Markov::Markopy::BOOST_PYTHON_MODULE\(\)](#).
Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.38 train()

```
def Python.Markopy.BaseCLI.train (
        self,
        str dataset,
        str seperator,
        str output,
        bool output_forced = False,
        bool bulk = False ) [inherited]
```

Train a model via CLI parameters.

Parameters

<i>model</i>	Model instance
<i>dataset</i>	filename for the dataset
<i>seperator</i>	seperator used with the dataset
<i>output</i>	output filename
<i>output_forced</i>	force overwrite
<i>bulk</i>	marks bulk operation with directories

Definition at line 94 of file [base.py](#).

```
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """
00097             @brief Train a model via CLI parameters
00098             @param model Model instance
00099             @param dataset filename for the dataset
00100             @param seperator seperator used with the dataset
00101             @param output output filename
00102             @param output_forced force overwrite
00103             @param bulk marks bulk operation with directories
00104         """
00105         logging pprint("Training.")
00106
00107         if not (dataset and seperator and (output or not output_forced)):
00108             logging pprint(f"Training mode requires -d/--dataset{' , -o/--output' if output_forced
00109             else'} and -s/--seperator parameters. Exiting.")
00110             return False
00111
00112         if not bulk and not self.check_corpus_path(dataset):
00113             logging pprint(f"\{dataset\} doesn't exists. Check the file path, or working directory")
00114             return False
00115
00116         if not self.check_export_path(output):
00117             logging pprint(f"Cannot create output at {output}")
00118             return False
00119
00120         if(seperator == '\\t'):
00121             logging pprint("Escaping seperator.", 3)
00122             seperator = '\t'
```

```

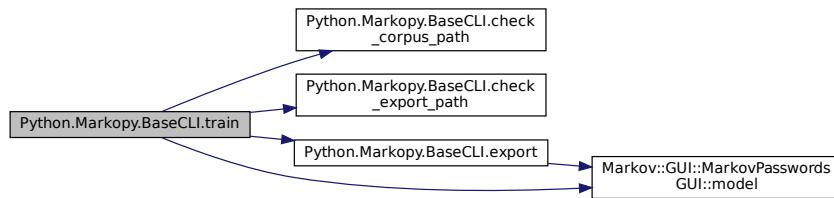
00122         if(len(seperator)!=1):
00123             logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00124             accepted.')
00125             exit(4)
00126
00127     logging pprint(f'Starting training.', 3)
00128     self.model.Train(dataset,seperator, int(self.args.threads))
00129     logging pprint(f'Training completed.', 2)
00130
00131     if(output):
00132         logging pprint(f'Exporting model to {output}', 2)
00133         self.export(output)
00134     else:
00135         logging pprint(f'Model will not be exported.', 1)
00136
00137     return True

```

References [Python.CudaMarkopy.CudaMarkopyCLI.args](#), [Python.Markopy.BaseCLI.args](#), [Python.Markopy.MarkopyCLI.args](#), [Python.Markopy.BaseCLI.check_corpus_path\(\)](#), [Python.Markopy.BaseCLI.check_export_path\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.model](#), [Python.Markopy.BaseCLI.model](#), [Python.Markopy.ModelMatrixCLI.model](#), [Python.Markopy.MarkovPasswordsCLI.model](#), and [Markov::GUI::MarkovPasswordsGUI.model\(\)](#).

Referenced by [Python.Markopy.BaseCLI.process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.3.39 TrainThread()

```

void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private], [inherited]

```

A single thread invoked by the Train function.

Parameters

<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

Definition at line 85 of file [markovPasswords.cpp](#).

```

00085
00086     {
00087         char format_str[] = "%ld,%s";
00088         format_str[3]=delimiter;
00089         std::string line;
00090         while (listhandler->next(&line) && keepRunning) {

```

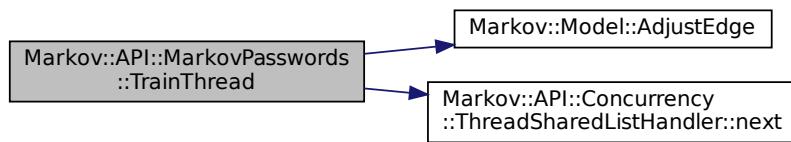
```

00090     long int oc;
00091     if (line.size() > 100) {
00092         line = line.substr(0, 100);
00093     }
00094     char* linebuf = new char[line.length()+5];
00095 #ifdef _WIN32
00096     sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to->
00097     "%ld,%s"
00098 #else
00099     sscanf(line.c_str(), format_str, &oc, linebuf);
00100 #endif
00101     this->AdjustEdge((const char*)linebuf, oc);
00102     delete linebuf;
00103 }
```

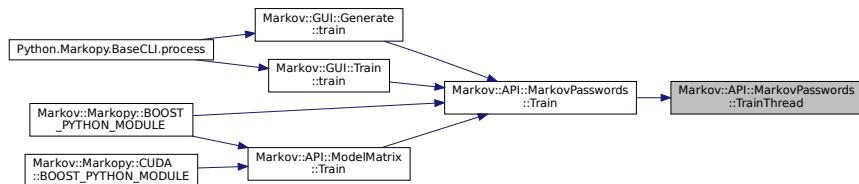
References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), [keepRunning](#), and [Markov::API::Concurrency::ThreadSharedListHandler::next](#).

Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.27.4 Member Data Documentation

9.27.4.1 args

`Python.Markopy.BaseCLI.args [inherited]`

Definition at line 75 of file [base.py](#).

Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.generate\(\)](#), [Python.Markopy.MarkopyCLI.help\(\)](#), [Python.Markopy.BaseCLI.init_post_arguments\(\)](#), [Python.Markopy.MarkopyCLI.parse\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.parse_fail\(\)](#), [Python.Markopy.BaseCLI.process\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.27.4.2 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]`

Definition at line 123 of file [markovPasswords.h](#).

9.27.4.3 edgeMatrix

`char** Markov::API::ModelMatrix::edgeMatrix [protected], [inherited]`
 2-D Character array for the edge Matrix (The characters of Nodes)
 Definition at line 175 of file [modelMatrix.h](#).
 Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.27.4.4 edges

`std::vector<Edge<char *>> Markov::Model< char >::edges [private], [inherited]`
 A list of all edges in this model.
 Definition at line 204 of file [model.h](#).

9.27.4.5 fileIO

`Python.Markopy.ModelMatrixCLI.fileIO`
 Definition at line 38 of file [mmx.py](#).
 Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), and [Python.Markopy.ModelMatrixCLI._generate\(\)](#).

9.27.4.6 matrixIndex

`char* Markov::API::ModelMatrix::matrixIndex [protected], [inherited]`
 to hold the Matrix index (To hold the orders of 2-D arrays")
 Definition at line 190 of file [modelMatrix.h](#).
 Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

9.27.4.7 matrixSize

`int Markov::API::ModelMatrix::matrixSize [protected], [inherited]`
 to hold Matrix size
 Definition at line 185 of file [modelMatrix.h](#).
 Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#), and [Markov::API::CUDA::CUDAMoo](#)

9.27.4.8 model

`Python.Markopy.ModelMatrixCLI.model`
 Definition at line 30 of file [mmx.py](#).
 Referenced by [Python.CudaMarkopy.CudaModelMatrixCLI._generate\(\)](#), [Python.Markopy.BaseCLI._generate\(\)](#), [Python.Markopy.ModelMatrixCLI._generate\(\)](#), [Python.Markopy.MarkovPasswordsCLI._generate\(\)](#), [Python.Markopy.BaseCLI.export\(\)](#), [Python.Markopy.BaseCLI.import_model\(\)](#), and [Python.Markopy.BaseCLI.train\(\)](#).

9.27.4.9 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`
 Dataset file input of our system

Definition at line 124 of file [markovPasswords.h](#).

9.27.4.10 nodes

```
std::map<char , Node<char *>> Markov::Model< char >::nodes [private], [inherited]  
Map LeftNode is the Nodes NodeValue Map RightNode is the node pointer.  
Definition at line 193 of file model.h.
```

9.27.4.11 outputFile

```
std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]  
File to save model of our system
```

Definition at line 125 of file [markovPasswords.h](#).

9.27.4.12 parser

```
Python.Markopy.BaseCLI.parser [inherited]  
Definition at line 25 of file base.py.
```

Referenced by [Python.CudaMarkopy.CudaMarkopyCLI.__init__\(\)](#), [Python.CudaMarkopy.CudaModelMatrixCLI.add_arguments\(\)](#), [Python.Markopy.BaseCLI.add_arguments\(\)](#), [Python.Markopy.AbstractGenerationModelCLI.add_arguments\(\)](#), [Python.Markopy.AbstractTrainingModelCLI.add_arguments\(\)](#), [Python.Markopy.MarkopyCLI.add_arguments\(\)](#), [Python.Markopy.ModelMatrixCLI.add_arguments\(\)](#), [Python.CudaMarkopy.CudaMarkopyCLI.help\(\)](#), and [Python.Markopy.MarkopyCLI.help\(\)](#).

9.27.4.13 print_help

```
Python.Markopy.BaseCLI.print\_help [inherited]  
Definition at line 39 of file base.py.  
Referenced by Python.Markopy.BaseCLI.help\(\), and Python.Markopy.MarkopyCLI.help\(\).
```

9.27.4.14 ready

```
bool Markov::API::ModelMatrix::ready [protected], [inherited]  
True when matrix is constructed. False if not.  
Definition at line 200 of file modelMatrix.h.  
Referenced by Markov::API::ModelMatrix::ConstructMatrix\(\), Markov::API::ModelMatrix::DeallocateMatrix\(\), and Markov::API::ModelMatrix::ModelMatrix\(\).
```

9.27.4.15 starterNode

```
Node<char *>* Markov::Model< char >::starterNode [private], [inherited]  
Starter Node of this model.  
Definition at line 198 of file model.h.
```

9.27.4.16 totalEdgeWeights

```
long int* Markov::API::ModelMatrix::totalEdgeWeights [protected], [inherited]  
Array of the Total Edge Weights.  
Definition at line 195 of file modelMatrix.h.  
Referenced by Markov::API::ModelMatrix::ConstructMatrix\(\), Markov::API::ModelMatrix::DeallocateMatrix\(\), and Markov::API::ModelMatrix::FastRandomWalkThread\(\).
```

9.27.4.17 valueMatrix

```
long int** Markov::API::ModelMatrix::valueMatrix [protected], [inherited]
2-d Integer array for the value Matrix (For the weights of Edges)
```

Definition at line 180 of file [modelMatrix.h](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), [Markov::API::ModelMatrix::DumpJSON\(\)](#), and [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

The documentation for this class was generated from the following file:

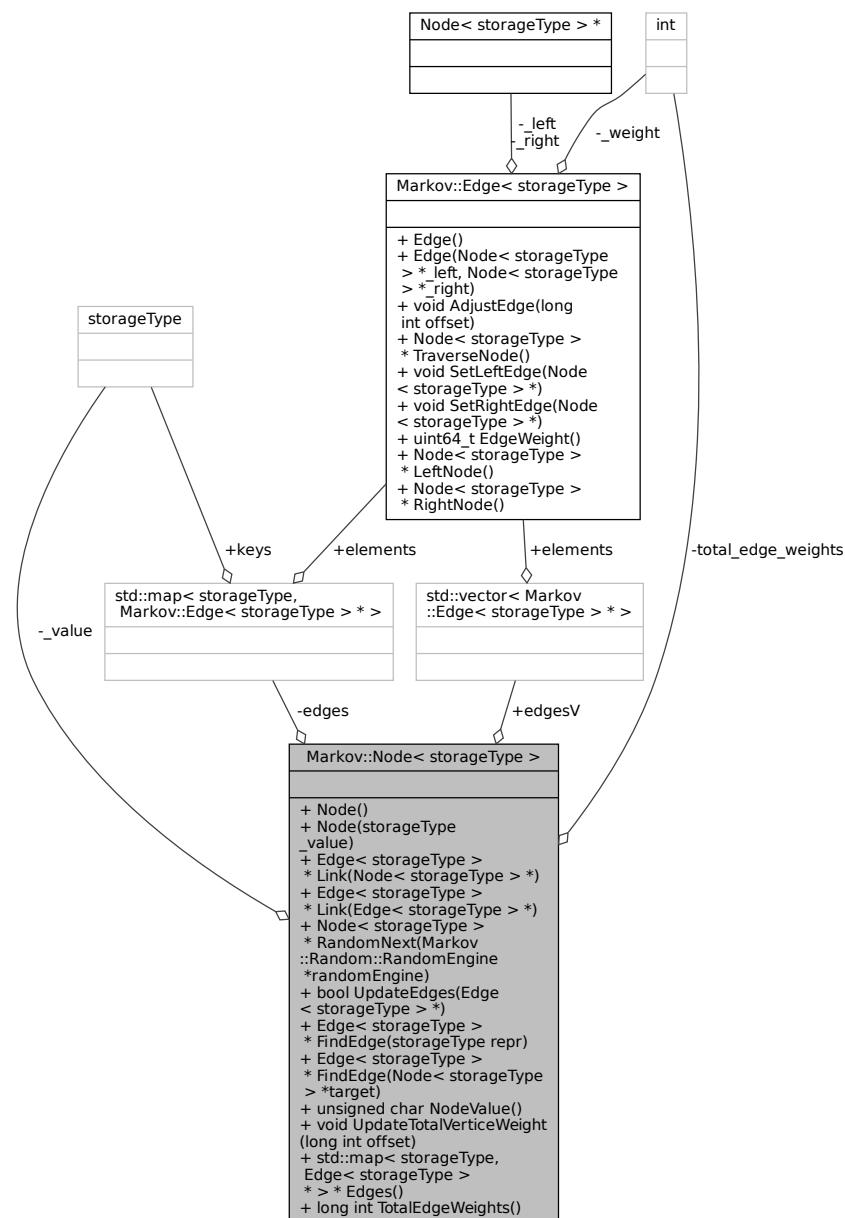
- [Markopy/Markopy/src/CLI/mmx.py](#)

9.28 Markov::Node< storageType > Class Template Reference

A node class that for the vertices of model. Connected with eachother using [Edge](#).

```
#include <node.h>
```

Collaboration diagram for [Markov::Node< storageType >](#):



Public Member Functions

- `Node ()`
Default constructor. Creates an empty `Node`.
- `Node (storageType _value)`
Constructor. Creates a `Node` with no edges and with given `NodeValue`.
- `Edge< storageType > * Link (Node< storageType > *)`
Link this node with another, with this node as its source.
- `Edge< storageType > * Link (Edge< storageType > *)`
Link this node with another, with this node as its source.
- `Node< storageType > * RandomNext (Markov::Random::RandomEngine *randomEngine)`
Choose a random node from the list of edges, with regards to its `EdgeWeight`, and `TraverseNode` to that.
- `bool UpdateEdges (Edge< storageType > *)`
Insert a new edge to the `this.edges`.
- `Edge< storageType > * FindEdge (storageType repr)`
Find an edge with its character representation.
- `Edge< storageType > * FindEdge (Node< storageType > *target)`
Find an edge with its pointer. Avoid unless necessary because computational cost of find by character is cheaper (because of std::map)
- `unsigned char NodeValue ()`
Return character representation of this node.
- `void UpdateTotalVertexWeight (long int offset)`
Change total weights with offset.
- `std::map< storageType, Edge< storageType > * > * Edges ()`
return edges
- `long int TotalEdgeWeights ()`
return total edge weights

Public Attributes

- `std::vector< Edge< storageType > * > edgesV`

Private Attributes

- `storageType _value`
Character representation of this node. 0 for starter, 0xff for terminator.
- `long int total_edge_weights`
Total weights of the vertices, required by `RandomNext`.
- `std::map< storageType, Edge< storageType > * > edges`
A map of all edges connected to this node, where this node is at the `LeftNode`. Map is indexed by `unsigned char`, which is the character representation of the node.

9.28.1 Detailed Description

```
template<typename storageType>
class Markov::Node< storageType >
```

A node class that for the vertices of model. Connected with eachother using `Edge`. This class will later be templated to accept other data types than `char*`. Definition at line 24 of file `node.h`.

9.28.2 Constructor & Destructor Documentation

9.28.2.1 Node() [1/2]

```
template<typename storageType >
Markov::Node< storageType >::Node
Default constructor. Creates an empty Node.
Definition at line 209 of file node.h.
00209
00210     this->value = 0;
00211     this->total_edge_weights = 0L;
00212 };
```

9.28.2.2 Node() [2/2]

```
template<typename storageType >
Markov::Node< storageType >::Node (
    storageType _value )
Constructor. Creates a Node with no edges and with given NodeValue.
```

Parameters

<u>_value</u>	- Nodes character representation.
---------------	-----------------------------------

Example Use: Construct nodes

```
Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
```

```
Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
```

Definition at line 203 of file node.h.

```
00203
00204     this->value = _value;
00205     this->total_edge_weights = 0L;
00206 };
```

9.28.3 Member Function Documentation

9.28.3.1 Edges()

```
template<typename storageType >
std::map< storageType, Markov::Edge< storageType > * > * Markov::Node< storageType >::Edges
[inline]
return edges
Definition at line 272 of file node.h.
00272
00273     return &(this->edges);
00274 }
```

Referenced by [Markov::API::MarkovPasswords::Buff\(\)](#).

Here is the caller graph for this function:



9.28.3.2 FindEdge() [1/2]

```
template<typename storageType >
Edge<storageType>* Markov::Node< storageType >::FindEdge (
    Node< storageType > * target )
```

Find an edge with its pointer. Avoid unless necessary because computational cost of find by character is cheaper (because of std::map)

Parameters

<i>target</i>	- target node.
---------------	----------------

Returns

[Edge](#) that is connected between this node, and the target node.

9.28.3.3 FindEdge() [2/2]

```
template<typename storageType >
Markov::Edge< storageType > * Markov::Node< storageType >::FindEdge (
    storageType repr )
```

Find an edge with its character representation.

Parameters

<i>repr</i>	- character NodeValue of the target node.
-------------	---

Returns

[Edge](#) that is connected between this node, and the target node.

Example Use: Construct and update edges

```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
Markov::Edge<unsigned char>* res = NULL;
src->Link(target1);
src->Link(target2);
res = src->FindEdge('b');
```

Definition at line 260 of file [node.h](#).

```
00260
00261     auto e = this->edges.find(repr); {
00262         if (e == this->edges.end()) return NULL;
00263         return e->second;
00264     };
```

9.28.3.4 Link() [1/2]

```
template<typename storageType >
Markov::Edge< storageType > * Markov::Node< storageType >::Link (
    Markov::Edge< storageType > * v )
```

Link this node with another, with this node as its source.

DOES NOT create a new [Edge](#).

Parameters

Edge	- Edge that will accept this node as its LeftNode.
----------------------	--

Returns

the same edge as parameter target.

Example Use: Construct and link nodes

```
Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
```

```
Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
LeftNode->Link(e);
```

Definition at line 227 of file [node.h](#).

```
00227
00228     v->SetLeftEdge(this);
00229     this->UpdateEdges(v);
00230     return v;
00231 }
```

9.28.3.5 Link() [2/2]

```
template<typename storageType >
Markov::Edge<storageType>* Markov::Node<storageType>::Link (
    Markov::Node<storageType>* n)
```

Link this node with another, with this node as its source.

Creates a new [Edge](#).

Parameters

<i>target</i>	- Target node which will be the RightNode() of new edge.
---------------	--

Returns

A new node with LeftNode as this, and RightNode as parameter target.

Example Use: Construct nodes

```
Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
```

Definition at line 220 of file [node.h](#).

```
00220
00221     Markov::Edge<storageType>* v = new Markov::Edge<storageType>(this, n);
00222     this->UpdateEdges(v);
00223     return v;
00224 }
```

9.28.3.6 NodeValue()

```
template<typename storageType >
unsigned char Markov::Node<storageType>::NodeValue [inline]
Return character representation of this node.
```

Returns

character representation at `_value`.

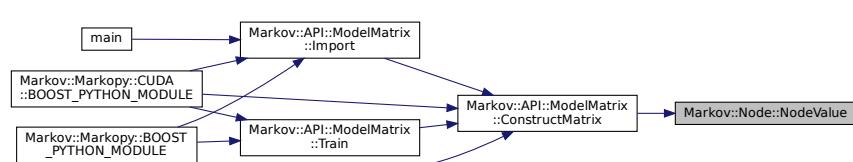
Definition at line 215 of file [node.h](#).

```
00215
00216     return _value;
00217 }
```

References [Markov::Node<storageType>::_value](#).

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#).

Here is the caller graph for this function:



9.28.3.7 RandomNext()

```
template<typename storageType >
Markov::Node< storageType > * Markov::Node< storageType >::RandomNext (
    Markov::Random::RandomEngine * randomEngine )
```

Chose a random node from the list of edges, with regards to its EdgeWeight, and TraverseNode to that. This operation is done by generating a random number in range of 0-this.total_edge_weights, and then iterating over the list of edges. At each step, EdgeWeight of the edge is subtracted from the random number, and once it is 0, next node is selected.

Returns

[Node](#) that was chosen at EdgeWeight biased random.

Example Use: Use randomNext to do a random walk on the model

```
char* buffer[64];
Markov::Model<char> model;
model.Import("model.mdl");
Markov::Node<char>* n = model.starterNode;
int len = 0;
Markov::Node<char>* temp_node;
while (true) {
    temp_node = n->RandomNext(randomEngine);
    if (len >= maxSetting) {
        break;
    }
    else if ((temp_node == NULL) && (len < minSetting)) {
        continue;
    }
    else if (temp_node == NULL){
        break;
    }

    n = temp_node;
    buffer[len++] = n->NodeValue();
}
```

Definition at line 234 of file node.h.

```
00234
00235
00236     //get a random NodeValue in range of total_vertex_weight
00237     long int selection = randomEngine->random() %
00238         this->total_edge_weights;/distribution()(generator());// distribution(generator);
00239     //make absolute, no negative modulus values wanted
00240     //selection = (selection >= 0) ? selection : (selection + this->total_edge_weights);
00241     for(int i=0;i<this->edgesV.size();i++){
00242         selection -= this->edgesV[i]->EdgeWeight();
00243         if (selection < 0) return this->edgesV[i]->TraverseNode();
00244     }
00245     //if this assertion is reached, it means there is an implementation error above
00246     std::cout << "This should never be reached (node failed to walk to next)\n"; //cant assert from
00247     child thread
00248     assert(true && "This should never be reached (node failed to walk to next)");
00249 }
```

References [Markov::Random::RandomEngine::random\(\)](#).

Here is the call graph for this function:



9.28.3.8 TotalEdgeWeights()

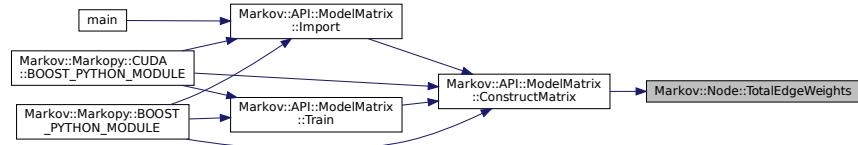
```
template<typename storageType >
long int Markov::Node< storageType >::TotalEdgeWeights [inline]
return total edge weights
```

Definition at line 277 of file [node.h](#).

```
00277     return this->total_edge_weights;
00278 }
00279 }
```

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#).

Here is the caller graph for this function:



9.28.3.9 UpdateEdges()

```
template<typename storageType >
bool Markov::Node<storageType >::UpdateEdges (
    Markov::Edge<storageType > * v )
```

Insert a new edge to the this.edges.

Parameters

<code>edge</code>	- New edge that will be inserted.
-------------------	-----------------------------------

Returns

true if insertion was successful, false if it fails.

Example Use: Construct and update edges

```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
e1->AdjustEdge(25);
src->UpdateEdges(e1);
e2->AdjustEdge(30);
src->UpdateEdges(e2);
```

Definition at line 252 of file [node.h](#).

```
00252
00253     this->edges.insert({ v->RightNode()->NodeValue(), v });
00254     this->edgesV.push_back(v);
00255     //this->total_edge_weights += v->EdgeWeight();
00256     return v->TraverseNode();
00257 }
```

9.28.3.10 UpdateTotalVerticeWeight()

```
template<typename storageType >
void Markov::Node<storageType >::UpdateTotalVerticeWeight (
    long int offset )
```

Change total weights with offset.

Parameters

<code>offset</code>	to adjust the vertice weight with
---------------------	-----------------------------------

Definition at line 267 of file [node.h](#).

```
00267
00268     this->total_edge_weights += offset;
00269 }
```

9.28.4 Member Data Documentation

9.28.4.1 _value

```
template<typename storageType >
storageType Markov::Node< storageType >::_value [private]
```

Character representation of this node. 0 for starter, 0xff for terminator.

Definition at line 179 of file [node.h](#).

Referenced by [Markov::Node< storageType >::NodeValue\(\)](#).

9.28.4.2 edges

```
template<typename storageType >
std::map<storageType, Edge<storageType>*>*> Markov::Node< storageType >::edges [private]
```

A map of all edges connected to this node, where this node is at the LeftNode. Map is indexed by unsigned char, which is the character representation of the node.

Definition at line 190 of file [node.h](#).

9.28.4.3 edgesV

```
template<typename storageType >
std::vector<Edge<storageType>*>*> Markov::Node< storageType >::edgesV
```

Definition at line 173 of file [node.h](#).

9.28.4.4 total_edge_weights

```
template<typename storageType >
long int Markov::Node< storageType >::total_edge_weights [private]
```

Total weights of the vertices, required by RandomNext.

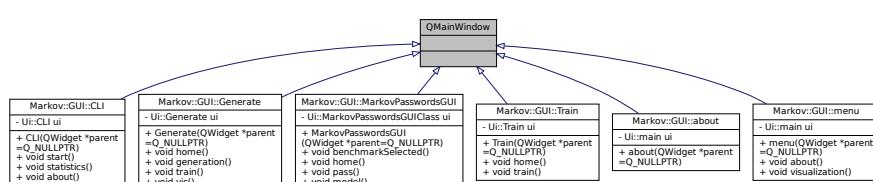
Definition at line 184 of file [node.h](#).

The documentation for this class was generated from the following files:

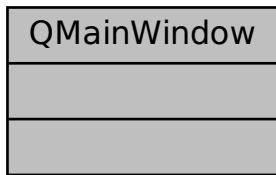
- [Markopy/MarkovModel/src/model.h](#)
- [Markopy/MarkovModel/src/node.h](#)

9.29 QMainWindow Class Reference

Inheritance diagram for QMainWindow:



Collaboration diagram for QMainWindow:



The documentation for this class was generated from the following file:

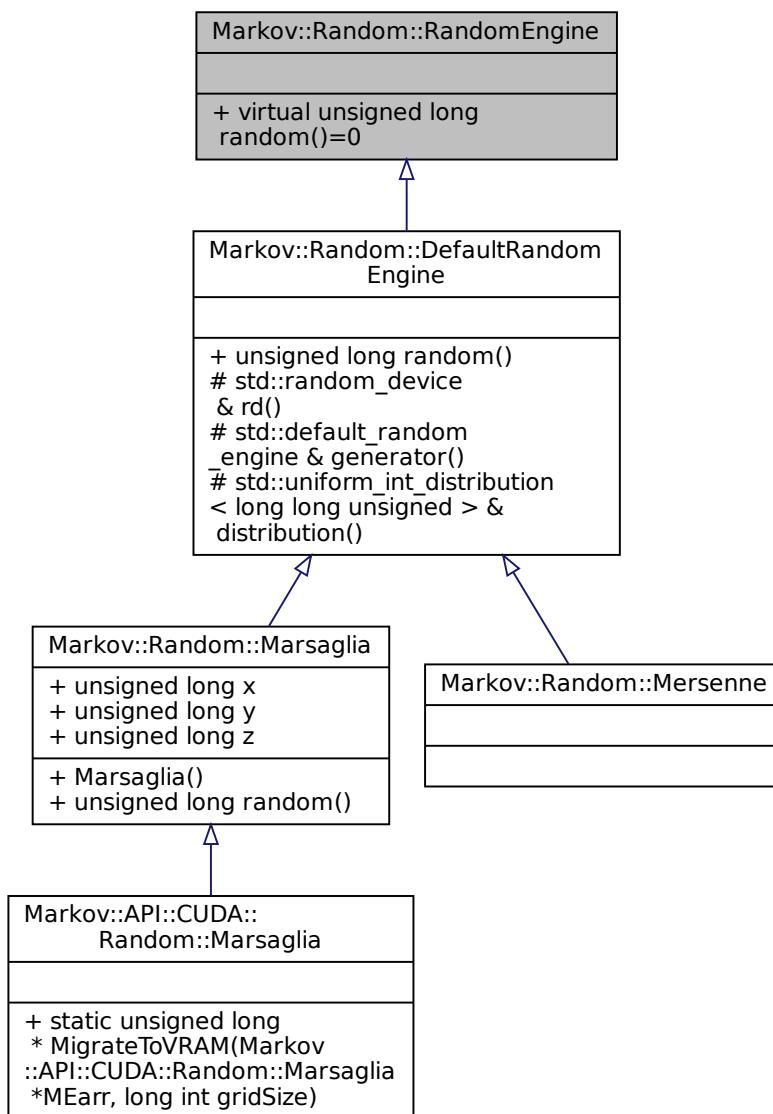
- [Markopy/MarkovPasswordsGUI/src/CLI.h](#)

9.30 **Markov::Random::RandomEngine Class Reference**

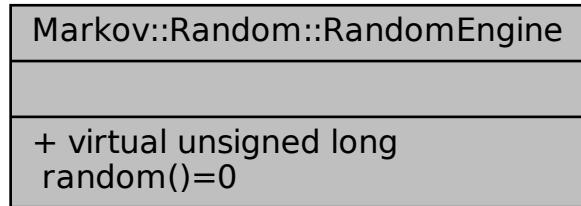
An abstract class for [Random](#) Engine.

```
#include <random.h>
```

Inheritance diagram for Markov::Random::RandomEngine:



Collaboration diagram for Markov::Random::RandomEngine:



Public Member Functions

- virtual unsigned long `random ()=0`

9.30.1 Detailed Description

An abstract class for [Random](#) Engine.

This class is used for generating random numbers, which are used for random walking on the graph.

Main reason behind allowing different random engines is that some use cases may favor performance, while some favor good random.

[Mersenne](#) can be used for truer random, while [Marsaglia](#) can be used for deterministic but fast random.

Definition at line 30 of file [random.h](#).

9.30.2 Member Function Documentation

9.30.2.1 random()

`virtual unsigned long Markov::Random::RandomEngine::random () [inline], [pure virtual]`

Implemented in [Markov::Random::Marsaglia](#), and [Markov::Random::DefaultRandomEngine](#).

Referenced by [Markov::Node< storageType >::RandomNext\(\)](#).

Here is the caller graph for this function:



The documentation for this class was generated from the following file:

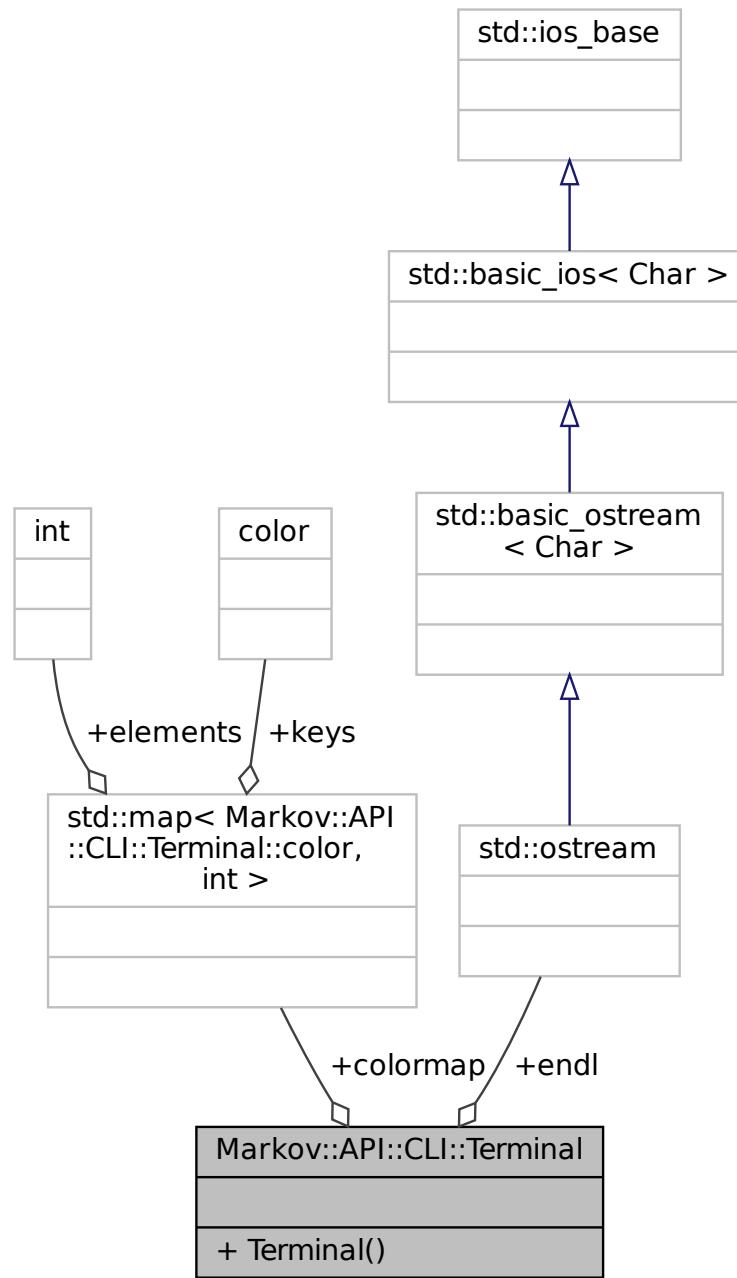
- [Markopy/MarkovModel/src/random.h](#)

9.31 Markov::API::CLI::Terminal Class Reference

pretty colors for [Terminal](#). Windows Only.

```
#include <term.h>
```

Collaboration diagram for Markov::API::CLI::Terminal:



Public Types

- enum `color` {

RESET , BLACK , RED , GREEN ,

YELLOW , BLUE , MAGENTA , CYAN ,

WHITE , LIGHTGRAY , DARKGRAY , BROWN }

Public Member Functions

- [Terminal \(\)](#)

Static Public Attributes

- static std::map< [Markov::API::CLI::Terminal::color](#), int > [colormap](#)
- static std::ostream [endl](#)

9.31.1 Detailed Description

pretty colors for [Terminal](#). Windows Only.
Definition at line [25](#) of file [term.h](#).

9.31.2 Member Enumeration Documentation

9.31.2.1 color

enum [Markov::API::CLI::Terminal::color](#)

Enumerator

RESET	
BLACK	
RED	
GREEN	
YELLOW	
BLUE	
MAGENTA	
CYAN	
WHITE	
LIGHTGRAY	
DARKGRAY	
BROWN	

Definition at line [33](#) of file [term.h](#).

```
00033 { RESET, BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, LIGHTGRAY, DARKGRAY, BROWN };
```

9.31.3 Constructor & Destructor Documentation

9.31.3.1 Terminal()

[Terminal::Terminal \(\)](#)

Default constructor. Get references to stdout and stderr handles.

Definition at line [62](#) of file [term.cpp](#).

```
00062 {
00063     /*this->;*/
00064 }
```

9.31.4 Member Data Documentation

9.31.4.1 colormap

```
std::map< Terminal::color, int > Terminal::colormap [static]
```

Initial value:

```
= {  
    {Terminal::color::BLACK, 30},  
    {Terminal::color::BLUE, 34},  
    {Terminal::color::GREEN, 32},  
    {Terminal::color::CYAN, 36},  
    {Terminal::color::RED, 31},  
    {Terminal::color::MAGENTA, 35},  
    {Terminal::color::BROWN, 01},  
    {Terminal::color::LIGHTGRAY, 0},  
    {Terminal::color::DARKGRAY, 0},  
    {Terminal::color::YELLOW, 33},  
    {Terminal::color::WHITE, 37},  
    {Terminal::color::RESET, 0},  
}
```

Definition at line 39 of file [term.h](#).

Referenced by [operator<<\(\)](#).

9.31.4.2 endl

```
std::ostream Markov::API::CLI::Terminal::endl [static]
```

Definition at line 44 of file [term.h](#).

The documentation for this class was generated from the following files:

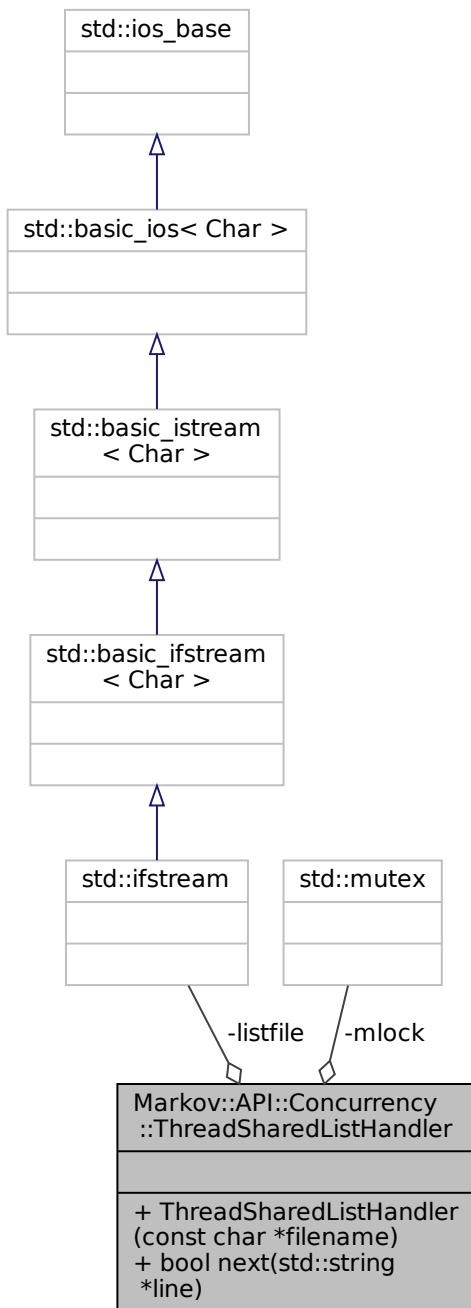
- Markopy/MarkovAPICLI/src/color/[term.h](#)
- Markopy/MarkovAPICLI/src/color/[term.cpp](#)

9.32 Markov::API::Concurrency::ThreadSharedListHandler Class Reference

Simple class for managing shared access to file.

```
#include <threadSharedListHandler.h>
```

Collaboration diagram for `Markov::API::Concurrency::ThreadSharedListHandler`:



Public Member Functions

- `ThreadSharedListHandler (const char *filename)`

Construct the Thread Handler with a filename.

- `bool next (std::string *line)`

Read the next line from the file.

Private Attributes

- std::ifstream [listfile](#)
- std::mutex [mlock](#)

9.32.1 Detailed Description

Simple class for managing shared access to file.

This class maintains the handover of each line from a file to multiple threads.

When two different threads try to read from the same file while reading a line isn't completed, it can have unexpected results. Line might be split, or might be read twice. This class locks the read action on the list until a line is completed, and then proceeds with the handover.

Definition at line 25 of file [threadSharedListHandler.h](#).

9.32.2 Constructor & Destructor Documentation

9.32.2.1 ThreadSharedListHandler()

```
Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler (
    const char * filename )
```

Construct the Thread Handler with a filename.

Simply open the file, and initialize the locks.

Example Use: Simple file read

```
ThreadSharedListHandler listhandler("test.txt");
std::string line;
std::cout << listhandler->next(&line) << "\n";
```

Example Use: Example use case from [MarkovPasswords](#) showing multithreaded access

```
void MarkovPasswords::Train(const char* datasetFileName, char delimiter, int threads) {
    ThreadSharedListHandler listhandler(datasetFileName);
    auto start = std::chrono::high_resolution_clock::now();
    std::vector<std::thread*> threadsV;
    for(int i=0;i<threads;i++) {
        threadsV.push_back(new std::thread(&MarkovPasswords::TrainThread, this, &listhandler,
                                         datasetFileName, delimiter));
    }
    for(int i=0;i<threads;i++){
        threadsV[i]->join();
        delete threadsV[i];
    }
    auto finish = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed = finish - start;
    std::cout << "Elapsed time: " << elapsed.count() << " s\n";
}
void MarkovPasswords::TrainThread(ThreadSharedListHandler *listhandler, const char* datasetFileName, char
    delimiter){
    char format_str[] ="%ld,%s";
    format_str[2]=delimiter;
    std::string line;
    while (listhandler->next(&line)) {
        long int oc;
        if (line.size() > 100) {
            line = line.substr(0, 100);
        }
        char* linebuf = new char[line.length()+5];
        sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
        this->AdjustEdge((const char*)linebuf, oc);
        delete linebuf;
    }
}
```

Parameters

filename	Filename for the file to manage.
--------------------------	----------------------------------

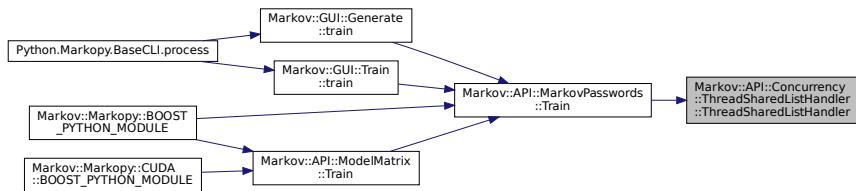
Definition at line 12 of file [threadSharedListHandler.cpp](#).

```
00012
00013     this->listfile;
00014     this->listfile.open(filename, std::ios_base::binary);
00015 }
```

References [listfile](#).

Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the caller graph for this function:



9.32.3 Member Function Documentation

9.32.3.1 next()

```
bool Markov::API::Concurrency::ThreadSharedListHandler::next (
    std::string * line )
```

Read the next line from the file.

This action will be blocked until another thread (if any) completes the read operation on the file.

Example Use: Simple file read

```
ThreadSharedListHandler listhandler("test.txt");
std::string line;
std::cout << listhandler->next(&line) << "\n";
```

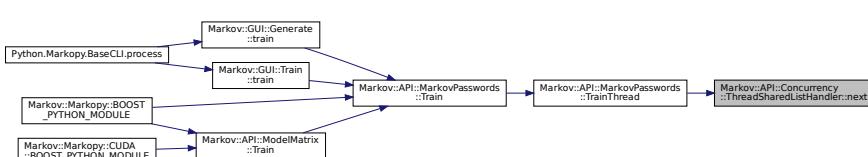
Definition at line 18 of file [threadSharedListHandler.cpp](#).

```
00018
00019     bool res = false;
00020     this->mlock.lock();
00021     res = (std::getline(this->listfile,*line,'\'\n'))? true : false;
00022     this->mlock.unlock();
00023
00024     return res;
00025 }
```

References [listfile](#), and [mlock](#).

Referenced by [Markov::API::MarkovPasswords::TrainThread\(\)](#).

Here is the caller graph for this function:



9.32.4 Member Data Documentation

9.32.4.1 listfile

```
std::ifstream Markov::API::Concurrency::ThreadSharedListHandler::listfile [private]
```

Definition at line 95 of file [threadSharedListHandler.h](#).

Referenced by [next\(\)](#), and [ThreadSharedListHandler\(\)](#).

9.32.4.2 mlock

```
std::mutex Markov::API::Concurrency::ThreadSharedListHandler::mlock [private]
```

Definition at line 96 of file [threadSharedListHandler.h](#).

Referenced by [next\(\)](#).

The documentation for this class was generated from the following files:

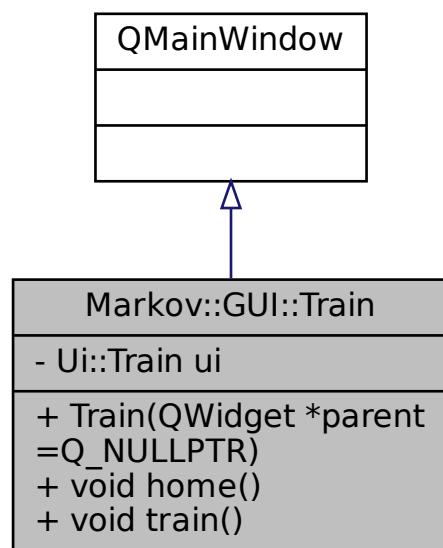
- [Markopy/MarkovAPI/src/threadSharedListHandler.h](#)
- [Markopy/MarkovAPI/src/threadSharedListHandler.cpp](#)

9.33 Markov::GUI::Train Class Reference

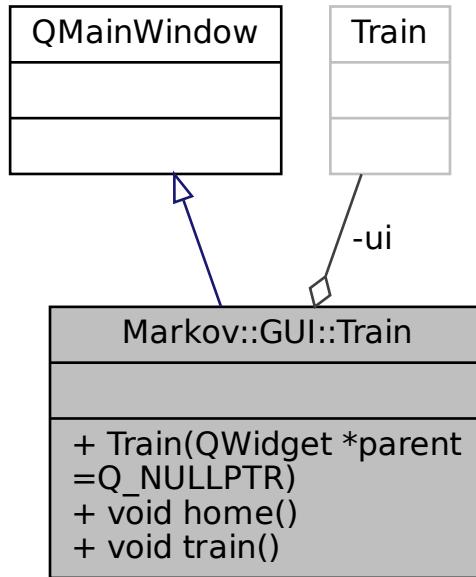
QT Training page class.

```
#include <Train.h>
```

Inheritance diagram for Markov::GUI::Train:



Collaboration diagram for Markov::GUI::Train:



Public Slots

- void [home \(\)](#)
- void [train \(\)](#)

Public Member Functions

- [Train \(QWidget *parent=Q_NULLPTR\)](#)

Private Attributes

- Ui::Train [ui](#)

9.33.1 Detailed Description

QT Training page class.

Definition at line [15](#) of file [Train.h](#).

9.33.2 Constructor & Destructor Documentation

9.33.2.1 Train()

```
Markov::GUI::Train::Train (
    QWidget * parent = Q_NULLPTR )
```

Definition at line [22](#) of file [Train.cpp](#).

```
00023     : QMainWidget(parent)
00024 {
00025     ui.setupUi(this);
00026 }
```

```

00027
00028
00029     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {train(); });
00030     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {home(); });
00031 //QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {goGenerate(); });
00032
00033 //ui.pushButton_3->setVisible(false);
00034
00035
00036 }
```

References [ui](#).

9.33.3 Member Function Documentation

9.33.3.1 home

```
void Markov::GUI::Train::home ( ) [slot]
```

Definition at line 73 of file [Train.cpp](#).

```

00073 {
00074     CLI* w = new CLI;
00075     w->show();
00076     this->close();
00077 }
```

9.33.3.2 train

```
void Markov::GUI::Train::train ( ) [slot]
```

Definition at line 38 of file [Train.cpp](#).

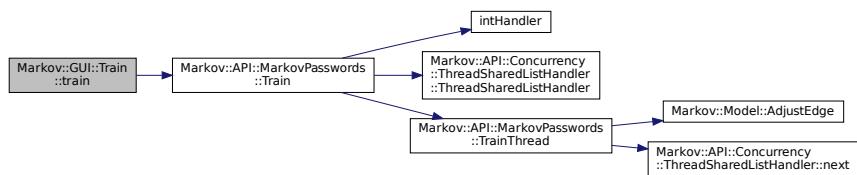
```

00038 {
00039
00040
00041
00042     QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00043     QFile file(file_name);
00044
00045     if (!file.open(QFile::ReadOnly | QFile::Text)) {
00046         QMessageBox::warning(this, "Error", "File Not Open!");
00047     }
00048     QTextStream in(&file);
00049     QString text = in.readAll();
00050     ui.plainTextEdit->setPlainText(text);
00051
00052
00053     char* cstr;
00054     std::string fname = file_name.toStdString();
00055     cstr = new char[fname.size() + 1];
00056     strcpy(cstr, fname.c_str());
00057
00058
00059
00060     char a=',';
00061     Markov::API::MarkovPasswords mp;
00062     mp.Import("models/2gram.mdl");
00063     mp.Train(cstr, a, 10); //please parameterize this hardcoded 10 threads
00064     mp.Export("models/finished.mdl");
00065
00066     ui.label_2->setText("Training DONE!");
00067 //ui.pushButton_3->setVisible(true);
00068
00069
00070     file.close();
00071 }
```

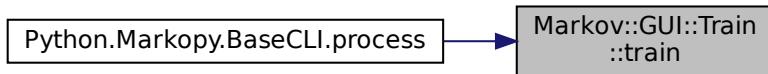
References [Markov::API::MarkovPasswords::Train\(\)](#), and [ui](#).

Referenced by [Python.Markopy.BaseCLI::process\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



9.33.4 Member Data Documentation

9.33.4.1 ui

Ui::Train Markov::GUI::Train::ui [private]

Definition at line 21 of file [Train.h](#).

Referenced by [train\(\)](#), and [Train\(\)](#).

The documentation for this class was generated from the following files:

- [Markopy/MarkovPasswordsGUI/src/Train.h](#)
- [Markopy/MarkovPasswordsGUI/src/Train.cpp](#)

CHAPTER10

File Documentation

10.1 Markopy/CudaMarkopy/src/CLI/cudamarkopy.py File Reference

base command line interface for python

Classes

- class [Python.CudaMarkopy.CudaMarkopyCLI](#)

CUDA extension to MarkopyCLI.

Namespaces

- [cudamarkopy](#)
- [Python.CudaMarkopy](#)

wrapper scripts for CudaMarkopy

Variables

- [cudamarkopy.spec = spec_from_loader\("markopy", SourceFileLoader\("markopy", os.path.abspath\("markopy.py"\)\)\)](#)
- [cudamarkopy.markopy = module_from_spec\(spec\)](#)
- [cudamarkopy.mp = CudaMarkopyCLI\(\)](#)

10.1.1 Detailed Description

base command line interface for python

Definition in file [cudamarkopy.py](#).

10.2 cudamarkopy.py

```
00001 #!/usr/bin/python3
00002
00003
00007
00008
00012
00013 import sys
00014 import os
00015 from importlib.util import spec_from_loader, module_from_spec
00016 from importlib.machinery import SourceFileLoader, ExtensionFileLoader
00017 import inspect
00018 try:
00019     spec = spec_from_loader("markopy", SourceFileLoader("markopy", os.path.abspath("markopy.py")))
00020     markopy = module_from_spec(spec)
00021     spec.loader.exec_module(markopy)
00022 except (ModuleNotFoundError, FileNotFoundError) as e:
00023     if(os.path.exists("../..../Markopy/src/CLI/markopy.py")):
00024         spec = spec_from_loader("markopy", SourceFileLoader("markopy",
00025             "../..../Markopy/src/CLI/markopy.py"))
00026         markopy = module_from_spec(spec)
00027         spec.loader.exec_module(markopy)
00028 try:
00029     from cudammx import CudaModelMatrixCLI
00030     from mmx import ModelMatrixCLI
00031     from mp import MarkovPasswordsCLI
00032
00033 except ModuleNotFoundError as e:
00034     print("Working in development mode. Trying to load markopy.py from ../..../Markopy/")
00035     if(os.path.exists("../..../Markopy/src/CLI/cudammx.py")):
```

```

00036     sys.path.insert(1, '../../../../../Markopy/src/CLI/')
00037     from cudammx import CudaModelMatrixCLI
00038     from mmx import ModelMatrixCLI
00039     from mp import MarkovPasswordsCLI
00040 else:
00041     raise e
00042
00043 from termcolor import colored
00044
00045
00046 class CudaMarkopyCLI(markopy.MarkopyCLI, CudaModelMatrixCLI):
00047     """
00048         @brief CUDA extension to MarkopyCLI. Adds CudaModelMatrixCLI to the interface.
00049         @belongsto Python::CudaMarkopy
00050         @extends Python::Markopy::MarkopyCLI
00051         @extends Python::CudaMarkopy::CudaModelMatrixCLI
00052     """
00053     def __init__(self) -> None:
00054         """
00055             ! @brief initialize CLI selector
00056             markopy.MarkopyCLI.__init__(self, add_help=False)
00057             self.parser.epilog+=f"""
00058             {__file__.split('/')[-1]} -mt CUDA generate trained.mdl -n 500 -w output.txt
00059             Import trained.mdl, and generate 500 lines to output.txt
00060         """
00061
00062     def help(self):
00063         """
00064             ! @brief overload help string
00065             self.parser.print_help = self.stub
00066             self.argsargsargsargs = self.parser.parse_known_args()[0]
00067             if(self.argsargsargsargs.model_type!="MMX"):
00068                 if(self.argsargsargsargs.model_type=="MP"):
00069                     mp = MarkovPasswordsCLI()
00070                     mp.add_arguments()
00071                     mp.parser.print_help()
00072                 elif(self.argsargsargsargs.model_type=="MMX"):
00073                     mp = ModelMatrixCLI()
00074                     mp.add_arguments()
00075                     mp.parser.print_help()
00076                 elif(self.argsargsargsargs.model_type == "CUDA"):
00077                     mp = CudaModelMatrixCLI()
00078                     mp.add_arguments()
00079                     mp.parser.print_help()
00080             else:
00081                 print(colored("Model Mode selection choices:", "green"))
00082                 self.print_helpprint_help()
00083                 print(colored("Following are applicable for -mt MP mode:", "green"))
00084                 mp = MarkovPasswordsCLI()
00085                 mp.add_arguments()
00086                 mp.parser.print_help()
00087                 print(colored("Following are applicable for -mt MMX mode:", "green"))
00088                 mp = ModelMatrixCLI()
00089                 mp.add_arguments()
00090                 mp.parser.print_help()
00091                 print(colored("Following are applicable for -mt CUDA mode:", "green"))
00092                 mp = CudaModelMatrixCLI()
00093                 mp.add_arguments()
00094                 mp.parser.print_help()
00095             exit()
00096
00097     def parse(self):
00098         markopy.MarkopyCLI.parse(self)
00099
00100     def parse_fail(self):
00101         """
00102             ! @brief Not a valid model type
00103             if(self.argsargsargsargs.model_type == "CUDA"):
00104                 self.cliclicli = CudaModelMatrixCLI()
00105             else:
00106                 markopy.MarkopyCLI.parse_fail(self)
00107
00108     if __name__ == "__main__":
00109         mp = CudaMarkopyCLI()
00110         mp.parse()
00110         mp.process()

```

10.3 Markopy/CudaMarkopy/src/CLI/cudammx.py File Reference

CUDAModelMatrix CLI wrapper.

Classes

- class [Python.CudaMarkopy.CudaModelMatrixCLI](#)

Python CLI wrapper for CudaModelMatrix.

Namespaces

- [cudammx](#)

Variables

- string [cudammx.ext](#) = "so"
- [cudammx.spec](#) = [spec_from_loader](#)("cudamarkopy", [ExtensionFileLoader](#)("cudamarkopy", [os.path.abspath](#)(f"cudamarkopy.{ext}")))
- [cudammx.cudamarkopy](#) = [module_from_spec](#)([spec](#))
- [cudammx.markopy](#) = [module_from_spec](#)([spec](#))
- [cudammx.mp](#) = [CudaModelMatrixCLI](#)()

10.3.1 Detailed Description

CUDAModelMatrix CLI wrapper.

Definition in file [cudammx.py](#).

10.4 cudammx.py

```

00001
00002
00006
00007
00008
00009 from importlib.util import spec_from_loader, module_from_spec
00010 from importlib.machinery import SourceFileLoader, ExtensionFileLoader
00011 import os
00012 import sys
00013 from mm import ModelMatrix
00014
00015 ext = "so"
00016 if os.name == 'nt':
00017     ext="pyd"
00018 try:
00019     spec = spec_from_loader("cudamarkopy", ExtensionFileLoader("cudamarkopy",
00020         os.path.abspath(f"cudamarkopy.{ext}")))
00021     cudamarkopy = module_from_spec(spec)
00022     spec.loader.exec_module(cudamarkopy)
00023 except ImportError as e:
00024     print(f"({_file_}) Working in development mode. Trying to load cudamarkopy.{ext} from
00025     ../../../../../out/")
00026     if(os.path.exists(f"../../../../out/lib/cudamarkopy.{ext}")):
00027         spec = spec_from_loader("cudamarkopy", ExtensionFileLoader("cudamarkopy",
00028             os.path.abspath(f"../../../../out/lib/cudamarkopy.{ext}")))
00029         cudamarkopy = module_from_spec(spec)
00030         spec.loader.exec_module(cudamarkopy)
00031     else:
00032         raise e
00033
00034 try:
00035     spec = spec_from_loader("markopy", SourceFileLoader("markopy", os.path.abspath("markopy.py")))
00036     markopy = module_from_spec(spec)
00037
00038     from mmx import ModelMatrixCLI
00039     from base import BaseCLI,AbstractGenerationModelCLI, AbstractTrainingModelCLI
00040 except ImportError as e:
00041     print("Working in development mode. Trying to load from ../../../../../out/")
00042     if(os.path.exists("../../../../Markopy/src/CLI/markopy.py")):
00043         spec = spec_from_loader("markopy", SourceFileLoader("markopy",
00044             os.path.abspath("../../../../Markopy/src/CLI/markopy.py")))
00045         markopy = module_from_spec(spec)
00046         sys.path.insert(1, '../../../../Markopy/src/CLI/')
00047
00048     from mmx import ModelMatrixCLI
00049     from base import BaseCLI,AbstractGenerationModelCLI, AbstractTrainingModelCLI

```

```

00049     else:
00050         raise e
00051
00052 import os
00053 import allegate as logging
00054
00055 class CudaModelMatrixCLI(ModelMatrixCLI,AbstractGenerationModelCLI):
00056     """
00057         @belongsto Python:::CudaMarkopy
00058         @brief Python CLI wrapper for CudaModelMatrix
00059         @extends Python:::Markopy:::ModelMatrixCLI
00060         @extends Python:::Markopy:::AbstractGenerationModelCLI
00061         @extends Markov:::API::CUDA::CUDAModelMatrix
00062     """
00063     def __init__(self):
00064         super().__init__()
00065         self.modelmodelmodelmodel = cudamarkopy.CUDAModelMatrix()
00066
00067     def add_arguments(self):
00068         super().add_arguments()
00069         self.parserparser.add_argument("-if", "--infinite", action="store_true", help="Infinite
generation mode")
00070
00071     def init_post_arguments(self):
00072         super().init_post_arguments()
00073         self.bInfinitebInfinite = self.argsargs.infinite
00074
00075     def _generate(self, wordlist : str ):
00076         self.modelmodelmodelmodel.FastRandomWalk(int(self.argsargs.count), wordlist,
int(self.argsargs.min), int(self.argsargs.max), self.fileIOfileIO, self.bInfinitebInfinite)
00077
00078 if __name__ == "__main__":
00079     mp = CudaModelMatrixCLI()
00080     mp.parse()
00081     mp.process()

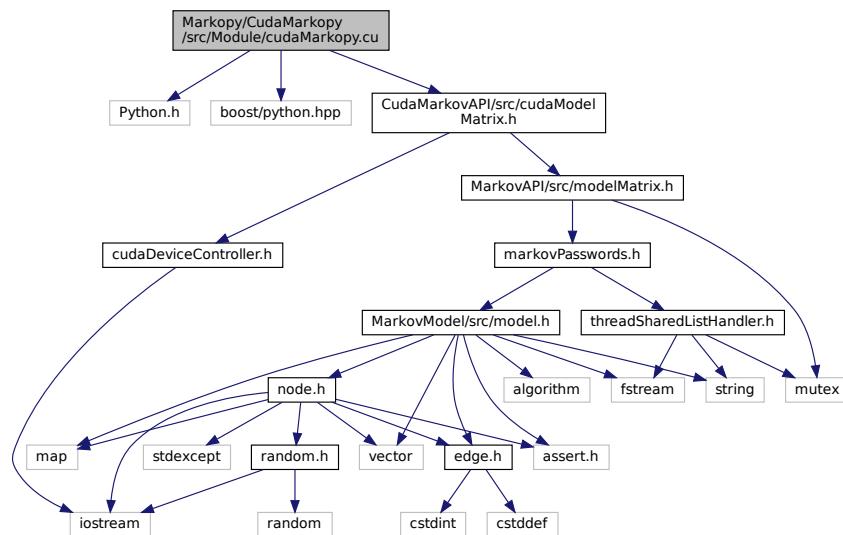
```

10.5 Markopy/CudaMarkopy/src/Module/cudaMarkopy.cu File Reference

```

#include <Python.h>
#include <boost/python.hpp>
#include "CudaMarkovAPI/src/cudaModelMatrix.h"
Include dependency graph for cudaMarkopy.cu:

```



Namespaces

- **Markov**

Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

- [Markov::Markopy](#)
*C*Python module for [Markov::API](#) objects.
- [Markov::Markopy::CUDA](#)
*C*Python module for [Markov::API::CUDA](#) objects.

Macros

- `#define BOOST_PYTHON_STATIC_LIB`

Functions

- [Markov::Markopy::CUDA::BOOST_PYTHON_MODULE](#) (`cudamarkopy`)

10.5.1 Macro Definition Documentation

10.5.1.1 BOOST_PYTHON_STATIC_LIB

```
#define BOOST_PYTHON_STATIC_LIB
Definition at line 9 of file cudaMarkopy.cu.
```

10.6 cudaMarkopy.cu

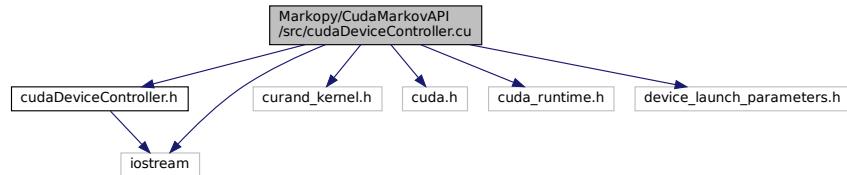
```
00001 /** @file cudaMarkopy.cpp
00002 * @brief CPython wrapper for libcudamarkov utils. GPU
00003 * @authors Ata Hakçıl
00004 *
00005 * @copydoc markopy.cpp
00006 * @copydoc cudaModelMatrix.cu
00007 */
00008
00009 #define BOOST_PYTHON_STATIC_LIB
00010 #include <Python.h>
00011 #include <boost/python.hpp>
00012 #include "CudaMarkovAPI/src/cudaModelMatrix.h"
00013
00014 using namespace boost::python;
00015
00016 /**
00017 * @brief CPython module for Markov::API::CUDA objects
00018 */
00019 namespace Markov::Markopy::CUDA{
00020     BOOST_PYTHON_MODULE(cudamarkopy)
00021     {
00022         bool (Markov::API::MarkovPasswords::*Export) (const char*) = &Markov::Model<char>::Export;
00023         void (Markov::API::CUDA::CUDAModelMatrix::*FastRandomWalk) (unsigned long int, const char*,
00024             int, int, bool, bool) = &Markov::API::CUDA::CUDAModelMatrix::FastRandomWalk;
00025         class_<Markov::API::CUDA::CUDAModelMatrix> ("CUDAModelMatrix", init<>())
00026
00027             .def(init<>())
00028             .def("Train", &Markov::API::ModelMatrix::Train)
00029             .def("Import", &Markov::API::ModelMatrix::Import, "Import a model file.")
00030             .def("Export", Export, "Export a model to file.")
00031             .def("ConstructMatrix", &Markov::API::ModelMatrix::ConstructMatrix)
00032             .def("DumpJSON", &Markov::API::ModelMatrix::DumpJSON)
00033             .def("FastRandomWalk", FastRandomWalk)
00034         ;
00035    };
00036 }
```

10.7 Markopy/CudaMarkovAPI/src/cudaDeviceController.cu File Reference

Simple static class for basic CUDA device controls.

```
#include "cudaDeviceController.h"
#include <iostream>
```

```
#include <curand_kernel.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <device_launch_parameters.h>
Include dependency graph for cudaDeviceController.cu:
```



Namespaces

- **Markov**
Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.
- **Markov::API**
Namespace for the [MarkovPasswords API](#).
- **Markov::API::CUDA**
Namespace for objects requiring [CUDA](#) libraries.

10.7.1 Detailed Description

Simple static class for basic CUDA device controls.

Authors

Ata Hakçıl

Controller class for CUDA device. This implementation only supports Nvidia devices.

Definition in file [cudaDeviceController.cu](#).

10.8 cudaDeviceController.cu

```

00001 /**
00002 * @brief Simple static class for basic CUDA device controls.
00003 * @authors Ata Hakçıl
00004 *
00005 * @copydoc Markov::API::CUDA::CUDADeviceController
00006 */
00007
00008 #include "cudaDeviceController.h"
00009 #include <iostream>
00010 #include <curand_kernel.h>
00011 #include <cuda.h>
00012 #include <cuda_runtime.h>
00013 #include <device_launch_parameters.h>
00014
00015 namespace Markov::API::CUDA{
00016     __host__ void Markov::API::CUDA::CUDADeviceController::ListCudaDevices() { //list cuda Capable
devices on host.
00017         int nDevices;
00018         cudaGetDeviceCount(&nDevices);
00019         for (int i = 0; i < nDevices; i++) {
00020             cudaDeviceProp prop;
00021             cudaGetDeviceProperties(&prop, i);
00022             std::cerr << "Device Number: " << i << "\n";
00023             std::cerr << "Device name: " << prop.name << "\n";
00024             std::cerr << "Memory Clock Rate (KHz): " << prop.memoryClockRate << "\n";
00025             std::cerr << "Memory Bus Width (bits): " << prop.memoryBusWidth << "\n";
00026             std::cerr << "Peak Memory Bandwidth (GB/s): " << 2.0 * prop.memoryClockRate *
(prop.memoryBusWidth / 8) / 1.0e6 << "\n";
00027             std::cerr << "Max Linear Threads: " << prop.maxThreadsDim[0] << "\n";
00028     }
00029 }
```

```

00028         }
00029     }
00030 }
00031
00032     __host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr(cudaError_t _status,
00033     const char* msg, bool bExit) {
00034     if (_status != cudaSuccess) {
00035         std::cerr << "\033[1;31m" << msg << " -> " << cudaGetErrorString(_status) << "(" << _status <<
00036         ")" << "\033[0m" << "\n";
00037     if(bExit) {
00038         cudaDeviceReset();
00039         exit(1);
00040     }
00041     return 0;
00042 }
00043
00044 */
00045     template <typename T>
00046     __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMalloc2DToFlat(T* dst, int row,
00047     int col){
00048         return cudaMalloc((T**) &dst, row*col*sizeof(T));
00049     }
00050
00051     template <typename T>
00052     __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat(T* dst, T** src,
00053     int row, int col){
00054         cudaError_t cudastatus;
00055         for(int i=0;i<row;i++){
00056             cudastatus = cudaMemcpy(dst + (i*col*sizeof(T)),
00057             src[i], col*sizeof(T), cudaMemcpyHostToDevice);
00058             if(cudastatus != cudaSuccess) return cudastatus;
00059         }
00060     }
00061 }
00062 };

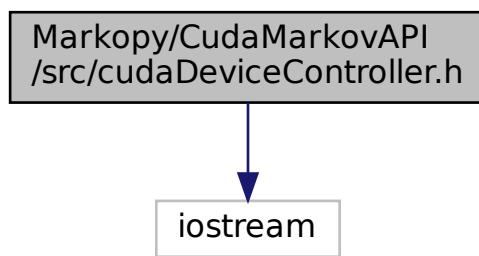
```

10.9 Markopy/CudaMarkovAPI/src/cudaDeviceController.h File Reference

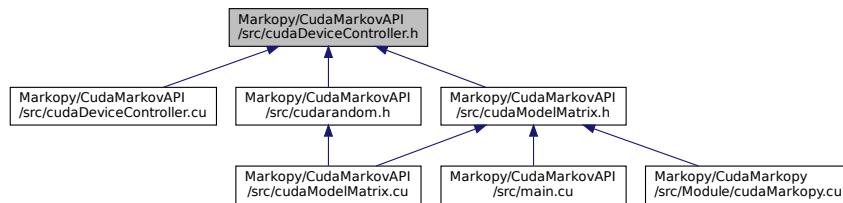
Simple static class for basic CUDA device controls.

```
#include <iostream>
```

Include dependency graph for cudaDeviceController.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::API::CUDA::CUDADeviceController](#)

Controller class for CUDA device.

Namespaces

- [Markov](#)

Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

- [Markov::API](#)

Namespace for the MarkovPasswords API.

- [Markov::API::CUDA](#)

Namespace for objects requiring CUDA libraries.

10.9.1 Detailed Description

Simple static class for basic CUDA device controls.

Authors

Ata Hakçıl

Controller class for CUDA device. This implementation only supports Nvidia devices.

Definition in file [cudaDeviceController.h](#).

10.10 cudaDeviceController.h

```

00001 /**
00002 * @file cudaDeviceController.h
00003 * @brief Simple static class for basic CUDA device controls.
00004 * @authors Ata Hakçıl
00005 * @copydoc Markov::API::CUDA::CUDADeviceController
00006 */
00007
00008 #pragma once
00009 #include <iostream>
00010
00011 /** @brief Namespace for objects requiring CUDA libraries.
00012 */
00013 namespace Markov::API::CUDA{
00014     /** @brief Controller class for CUDA device
00015      *
00016      * This implementation only supports Nvidia devices.
00017      */
00018     class CUDADeviceController{
00019     public:
00020         /** @brief List CUDA devices in the system.
00021         *
00022         * This function will print details of every CUDA capable device in the system.
00023         *
00024         * @b Example @b output:
00025         * @code{.txt}
00026         * Device Number: 0
00027         * Device name: GeForce RTX 2070
  
```

```

00028     * Memory Clock Rate (KHz): 7001000
00029     * Memory Bus Width (bits): 256
00030     * Peak Memory Bandwidth (GB/s): 448.064
00031     * Max Linear Threads: 1024
00032     * @endcode
00033   */
00034   __host__ static void ListCudaDevices();
00035
00036   protected:
00037     /** @brief Check results of the last operation on GPU.
00038      *
00039      * Check the status returned from cudaMalloc/cudaMemcpy to find failures.
00040      *
00041      * If a failure occurs, its assumed beyond redemption, and exited.
00042      * @param _status Cuda error status to check
00043      * @param msg Message to print in case of a failure
00044      * @return 0 if successful, 1 if failure.
00045      * @b Example @b output:
00046      * @code{.cpp}
00047      * char *da, a = "test";
00048      * cudastatus = cudaMalloc((char **)&da, 5*sizeof(char));
00049      * CudaCheckNotifyErr(cudastatus, "Failed to allocate VRAM for *da.\n");
00050      * @endcode
00051   */
00052   __host__ static int CudaCheckNotifyErr(cudaError_t _status, const char* msg, bool bExit=true);
00053
00054
00055   /** @brief Malloc a 2D array in device space
00056   *
00057   * This function will allocate enough space on VRAM for flattened 2D array.
00058   *
00059   * @param dst destination pointer
00060   * @param row row size of the 2d array
00061   * @param col column size of the 2d array
00062   * @return cudaError_t status of the cudaMalloc operation
00063   *
00064   * @b Example @b output:
00065   * @code{.cpp}
00066   *   cudaError_t cudastatus;
00067   *   char* dst;
00068   *   cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
00069   *   if(cudastatus!=cudaSuccess){
00070   *       CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
00071   *   }
00072   * @endcode
00073 */
00074   template <typename T>
00075   __host__ static cudaError_t CudaMalloc2DToFlat(T** dst, int row, int col){
00076     cudastatus = cudaMalloc((T **)dst, row*col*sizeof(T));
00077     CudaCheckNotifyErr(cudastatus, "cudaMalloc Failed.", false);
00078     return cudastatus;
00079   }
00080
00081
00082   /** @brief Memcpy a 2D array in device space after flattening
00083   *
00084   * Resulting buffer will not be true 2D array.
00085   *
00086   * @param dst destination pointer
00087   * @param rc source pointer
00088   * @param row row size of the 2d array
00089   * @param col column size of the 2d array
00090   * @return cudaError_t status of the cudaMalloc operation
00091   *
00092   * @b Example @b output:
00093   * @code{.cpp}
00094   *   cudaError_t cudastatus;
00095   *   char* dst;
00096   *   cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
00097   *   CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
00098   *   cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
00099   *   CudaCheckNotifyErr(cudastatus, " CudaMemcpy2DToFlat Failed.", false);
00100   * @endcode
00101 */
00102   template <typename T>
00103   __host__ static cudaError_t CudaMemcpy2DToFlat(T* dst, T** src, int row, int col){
00104     T* tempbuf = new T[row*col];
00105     for(int i=0;i<row;i++){
00106       memcpy(&(tempbuf[row*i]), src[i], col);
00107     }
00108     return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00109   }
00110
00111
00112   /** @brief Both malloc and memcpy a 2D array into device VRAM.
00113   *
00114   * Resulting buffer will not be true 2D array.

```

```

00115      *
00116      * @param dst destination pointer
00117      * @param rc source pointer
00118      * @param row row size of the 2d array
00119      * @param col column size of the 2d array
00120      * @return cudaError_t status of the cudaMalloc operation
00121      *
00122      * @b Example @b output:
00123      * @code{.cpp}
00124      *   cudaError_t cudastatus;
00125      *   char* dst;
00126      *   cudastatus = CudaMigrate2DFlat<long int>(
00127      *     &dst, this->valueMatrix, this->matrixSize, this->matrixSize);
00128      *   CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
00129      * @endcode
00130      */
00131  template <typename T>
00132  __host__ static cudaError_t CudaMigrate2DFlat(T** dst, T** src, int row, int col){
00133      cudaError_t cudastatus;
00134      cudastatus = CudaMalloc2DToFlat<T>(dst, row, col);
00135      if(cudastatus!=cudaSuccess){
00136          CudaCheckNotifyErr(cudastatus, "    CudaMalloc2DToFlat Failed.", false);
00137          return cudastatus;
00138      }
00139      cudastatus = CudaMemcpy2DToFlat<T>(*dst,src,row,col);
00140      CudaCheckNotifyErr(cudastatus, "    CudaMemcpy2DToFlat Failed.", false);
00141      return cudastatus;
00142  }
00143
00144  private:
00145  };
00146
00147 };

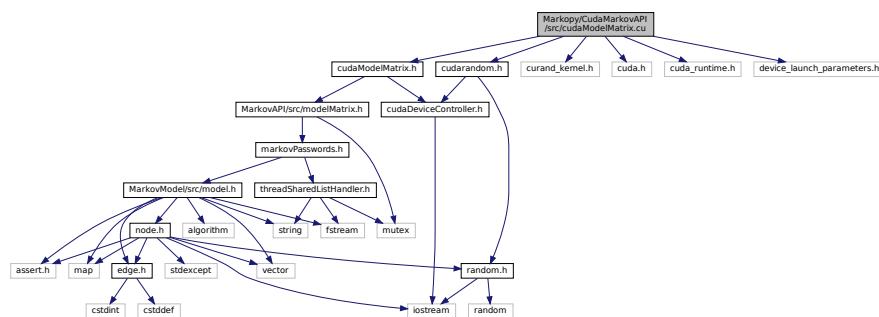
```

10.11 Markopy/CudaMarkovAPI/src/cudaModelMatrix.cu File Reference

CUDA accelerated extension of [Markov::API::ModelMatrix](#).

```
#include "cudaModelMatrix.h"
#include "cudarandom.h"
#include <curand_kernel.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <device_launch_parameters.h>
```

Include dependency graph for `cudaModelMatrix.cu`:



Namespaces

- [Markov](#)

Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.

- [Markov::API](#)

Namespace for the [MarkovPasswords API](#).

- [Markov::API::CUDA](#)

Namespace for objects requiring [CUDA](#) libraries.

Functions

- `__global__ void Markov::API::CUDA::FastRandomWalkCUDAKernel (unsigned long int n, int minLen, int maxLen, char *outputBuffer, char *matrixIndex, long int *totalEdgeWeights, long int *valueMatrix, char *edgeMatrix, int matrixSize, int memoryPerKernelGrid, unsigned long *seed)`
`CUDA kernel for the FastRandomWalk operation.`
- `__device__ char * Markov::API::CUDA::strchr (char *p, char c, int s_len)`
`strchr implementation on device space`

10.11.1 Detailed Description

CUDA accelerated extension of [Markov::API::ModelMatrix](#).

Authors

Ata Hakçıl

Extension of [Markov::API::ModelMatrix](#) which is modified to run on GPU devices. This implementation only supports Nvidia devices.

Class to flatten and reduce [Markov::Model](#) to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)

To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are synchronized and files are flushed every 50M operations.

Definition in file [cudaModelMatrix.cu](#).

10.12 cudaModelMatrix.cu

```

00001 /**
00002  * @file cudaModelMatrix.cu
00003  * @brief CUDA accelerated extension of Markov::API::ModelMatrix
00004  * @authors Ata Hakçıl
00005  * @copydoc Markov::API::CUDA::CUDAModelMatrix
00006 */
00007
00008 #include "cudaModelMatrix.h"
00009 #include "cudarandom.h"
00010
00011
00012 #include <curand_kernel.h>
00013 #include <cuda.h>
00014 #include <cuda_runtime.h>
00015 #include <device_launch_parameters.h>
00016
00017 using Markov::API::CUDA::CUDADeviceController;
00018
00019 namespace Markov::API::CUDA{
00020     __host__ void Markov::API::CUDA::CUDAModelMatrix::MigrateMatrix(){
00021         cudastatus;
00022
00023         cudastatus = cudaMalloc((char**)&(this->device_matrixIndex),
00024             this->matrixSize*sizeof(char));
00025         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_matrixIndex.");
00026
00027         cudastatus = cudaMalloc((long int **)&(this->device_totalEdgeWeights),
00028             this->matrixSize*sizeof(long int));
00029         CudaCheckNotifyErr(cudastatus, "Cuda failed to initialize device_totalEdgeWeights.");
00030
00031         cudastatus = cudaMemcpy(this->device_matrixIndex, this->matrixIndex,
00032             this->matrixSize*sizeof(char), cudaMemcpyHostToDevice);
00033         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Index)");
00034
00035         cudastatus = cudaMemcpy(this->device_totalEdgeWeights, this->totalEdgeWeights,
00036             this->matrixSize*sizeof(long int), cudaMemcpyHostToDevice);
00037         CudaCheckNotifyErr(cudastatus, "Cuda failed to copy to device memory. (Total Edge Values)");
00038
00039         cudastatus = CudaMigrate2DFlat<char>(
00040             &(this->device_edgeMatrix), this->edgeMatrix, this->matrixSize, this->matrixSize);
00041         CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize edge matrix.");
00042
00043         cudastatus = CudaMigrate2DFlat<long int>(
00044             &(this->device_valueMatrix), this->valueMatrix, this->matrixSize, this->matrixSize);
00045         CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
00046

```

```

00046      }
00047
00048     /*__host__ char* Markov::API::CUDAModelMatrix::AllocVRAMOutputBuffer(long int n, long int
00049     singleGenMaxLen, long int CUDAKernelGridSize, long int sizePerGrid){
00050         cudaError_t cudastatus;
00051         cudastatus = cudaMalloc((char**)&this->device_outputBuffer1, CUDAKernelGridSize*sizePerGrid);
00052         CudaCheckNotifyErr(cudastatus, "Failed to allocate VRAM buffer. (Possibly out of VRAM.)");
00053
00054     return this->device_outputBuffer1;
00055 }
00056
00057
00058 __host__ void Markov::API::CUDAModelMatrix::FastRandomWalk(unsigned long int n, const char*
00059 wordlistFileName, int minLen, int maxLen, bool bFileIO, bool bInfinite){
00060     cudaDeviceProp prop;
00061     int device=0;
00062     cudaGetDeviceProperties(&prop, device);
00063     cudaChooseDevice(&device, &prop);
00064     //std::cout << "Flattening matrix." << std::endl;
00065     this->FlattenMatrix();
00066     //std::cout << "Migrating matrix." << std::endl;
00067     this->MigrateMatrix();
00068     //std::cout << "Migrated matrix." << std::endl;
00069     std::ofstream wordlist;
00070     if(bFileIO)
00071         wordlist.open(wordlistFileName);
00072
00073     cudaBlocks = 1024;
00074     cudaThreads = 256;
00075     iterationsPerKernelThread = 100;
00076     alternatingKernels = 2;
00077     totalOutputPerKernel = (long int)cudaBlocks*(long int)cudaThreads*iterationsPerKernelThread;
00078     totalOutputPerSync= totalOutputPerKernel*alternatingKernels;
00079     numberofPartitions = n/totalOutputPerSync;
00080     cudaGridSize = cudaBlocks*cudaThreads;
00081     cudaMemPerGrid = (maxLen+2)*iterationsPerKernelThread;
00082     cudaPerKernelAllocationSize = cudaGridSize*cudamemPerGrid;
00083     this->prepKernelMemoryChannel(alternatingKernels);
00084
00085     unsigned long int leftover = n - (totalOutputPerSync*numberofPartitions);
00086
00087     if(bInfinite && !numberofPartitions) numberofPartitions=5;
00088     std::cerr << cudaPerKernelAllocationSize << "\n";
00089
00090     if(n%totalOutputPerSync) std::cerr << "For optimization, request outputs multiples of "<<
00091     totalOutputPerSync << ".\n";
00092
00093     //start kernelID 1
00094     this->LaunchAsyncKernel(1, minLen, maxLen);
00095
00096     for(int i=1;i<numberofPartitions;i++){
00097         if(bInfinite) i=0;
00098
00099         //wait kernelID1 to finish, and start kernelID 0
00100         cudaStreamSynchronize(this->cudastreams[1]);
00101         this->LaunchAsyncKernel(0, minLen, maxLen);
00102
00103         //start memcpy from kernel 1 (block until done)
00104         this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00105
00106         //wait kernelID 0 to finish, then start kernelID1
00107         cudaStreamSynchronize(this->cudastreams[0]);
00108         this->LaunchAsyncKernel(1, minLen, maxLen);
00109
00110         //start memcpy from kernel 0 (block until done)
00111         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00112     }
00113
00114     //wait kernelID1 to finish, and start kernelID 0
00115     cudaStreamSynchronize(this->cudastreams[1]);
00116     this->LaunchAsyncKernel(0, minLen, maxLen);
00117     this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00118     cudaStreamSynchronize(this->cudastreams[0]);
00119     this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00120
00121
00122     if(!leftover) return;
00123     alternatingKernels=1;
00124     std::cerr << "Remaining line count (" << leftover << ") is lower than partition. Adjusting CUDA
00125     workload..\n";
00126     this->iterationsPerKernelThread = leftover/cudaGridSize;
00127     this->LaunchAsyncKernel(0, minLen, maxLen);
00128     cudaStreamSynchronize(this->cudastreams[0]);
00129     this->GatherAsyncKernelOutput(0, bFileIO, wordlist);

```

```

00129     leftover -= this->iterationsPerKernelThread*cudaGridSize;
00130     if(!leftover) return;
00131
00132     std::cerr << "Remaining line count (" << leftover << ") is lower than minimum possible. Handing
00133     over to CPU generation.\n";
00134     this->iterationsPerKernelThread = leftover/cudaGridSize;
00135
00136     leftover -= this->iterationsPerKernelThread;
00137
00138     if(!leftover) return;
00139     std::cerr << "Remaining " << leftover << " lines are absolutely not worth printing.\n";
00140     Markov::API::ModelMatrix::ConstructMatrix();
00141     Markov::API::ModelMatrix::FastRandomWalk(leftover, &wordlist, minLen, maxLen, 1, bFileIO);
00142
00143 }
00144
00145 __host__ void Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel(int numberOfStreams){
00146
00147     this->cudastreams = new cudaStream_t[numberOfStreams];
00148     for(int i=0;i<numberOfStreams;i++)
00149         cudaStreamCreate(&this->cudastreams[i]);
00150
00151     this->outputBuffer = new char*[numberOfStreams];
00152     for(int i=0;i<numberOfStreams;i++)
00153         this->outputBuffer[i] = new char[cudaPerKernelAllocationSize];
00154
00155     cudaError_t cudastatus;
00156     this->device_outputBuffer = new char*[numberOfStreams];
00157     for(int i=0;i<numberOfStreams;i++){
00158         cudastatus = cudaMalloc((char**)&(device_outputBuffer[i]),
00159             cudaPerKernelAllocationSize);
00160         CudaCheckNotifyErr(cudastatus, "Failed to establish memory channel. Possibly out of
VRAM?");
00161     }
00162
00163     this->device_seeds = new unsigned long*[numberOfStreams];
00164     for(int i=0;i<numberOfStreams;i++){
00165         Markov::API::CUDA::Random::Marsaglia *MEarr = new
00166         Markov::API::CUDA::Random::Marsaglia[cudaGridSize];
00167         this->device_seeds[i] = Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(MEarr,
00168             cudaGridSize);
00169         delete[] MEarr;
00170     }
00171
00172     __host__ void Markov::API::CUDA::CUDAModelMatrix::LaunchAsyncKernel(int kernelID, int minLen, int
maxLen){
00173
00174     //if(kernelID == 0); // cudaStreamSynchronize(this->cudastreams[2]);
00175     //else cudaStreamSynchronize(this->cudastreams[kernelID-1]);
00176     FastRandomWalkCUDAKernel<<cudaBlocks,cudaThreads,0,
00177     this->cudastreams[kernelID]>>(iterationsPerKernelThread, minLen, maxLen,
00178     this->device_outputBuffer[kernelID], this->device_matrixIndex,
00179     this->device_totalEdgeWeights, this->device_valueMatrix, this->device_edgeMatrix,
00180     this->matrixSize, cudaMemPerGrid, this->device_seeds[kernelID]);
00181     //std::cerr << "Started kernel" << kernelID << "\n";
00182
00183     __host__ void Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput(int kernelID, bool
bFileIO, std::ofstream &wordlist){
00184
00185     cudaMemcpy(this->outputBuffer[kernelID],this->device_outputBuffer[kernelID],cudaPerKernelAllocationSize,
00186     cudaMemcpyDeviceToHost);
00187     //std::cerr << "Kernel" << kernelID << " output copied\n";
00188     if(bFileIO){
00189         for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00190             wordlist << &this->outputBuffer[kernelID][j];
00191         }
00192     }else{
00193         for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00194             std::cout << &this->outputBuffer[kernelID][j];
00195         }
00196     }
00197
00198     __global__ void FastRandomWalkCUDAKernel(unsigned long int n, int minLen, int maxLen, char*
00199     outputBuffer,
00200     char* matrixIndex, long int* totalEdgeWeights, long int* valueMatrix, char *edgeMatrix, int
matrixSize, int memoryPerKernelGrid, unsigned long *seed){
00201
00202         int kernelWorkerIndex = threadIdx.x + blockIdx.x * blockDim.x;
00203
00204         if(n==0) return;
00205         char* e;

```

```

00202     int index = 0;
00203     char next;
00204     int len=0;
00205     long int selection;
00206     char cur;
00207     long int bufferctr = 0;
00208     unsigned long int *x,*y,*z,t;
00209     char* res = &outputBuffer[kernelWorkerIndex*memoryPerKernelGrid];
00210     x=&seed[kernelWorkerIndex*3];
00211     y=&seed[kernelWorkerIndex*3+1];
00212     z=&seed[kernelWorkerIndex*3+2];
00213     for (int i = 0; i < n; i++) {
00214         cur=199;
00215         len=0;
00216         while (true) {
00217             e = strchr(matrixIndex, cur, matrixSize);
00218             index = e - matrixIndex;
00219             /*selection = Markov::API::CUDA::Random::devrandom(
00220                 seed[kernelWorkerIndex*3],
00221                 seed[kernelWorkerIndex*3+1],
00222                 seed[kernelWorkerIndex*3+2]) % totalEdgeWeights[index];*/
00223             *x ^= *x << 16;
00224             *x ^= *x >> 5;
00225             *x ^= *x << 1;
00226
00227             t = *x;
00228             *x = *y;
00229             *y = *z;
00230             *z = t ^ *x ^ *y;
00231             selection = *z % totalEdgeWeights[index];
00232             for(int j=0;j<matrixSize-1;j++){
00233                 selection -= valueMatrix[index*matrixSize + j];
00234                 if (selection < 0){
00235                     next = edgeMatrix[index*sizeof(char)*matrixSize + j];
00236                     break;
00237                 }
00238             }
00239             if (len >= maxLen) break;
00240             else if ((next < 0) && (len < minLen)) continue;
00241             else if (next < 0) break;
00242             cur = next;
00243             res[bufferctr + len++] = cur;
00244         }
00245         res[bufferctr + len++] = '\n';
00246         bufferctr+=len;
00247     }
00248     res[bufferctr] = '\0';
00249 }
00250
00251
00252 __device__ char* strchr(char* p, char c, int s_len){
00253     for (; ; ++p, s_len--) {
00254         if (*p == c)
00255             return((char *)p);
00256         if (!*p)
00257             return((char *)NULL);
00258     }
00259 }
00260
00261 __host__ void Markov::API::CUDAModelMatrix::FlattenMatrix(){
00262     this->flatEdgeMatrix = new char[this->matrixSize*this->matrixSize];
00263
00264     this->flatValueMatrix = new long int[this->matrixSize*this->matrixSize];
00265     for(int i=0;i<this->matrixSize;i++){
00266         memcpy(&this->flatEdgeMatrix[i*this->matrixSize], this->edgeMatrix[i], this->matrixSize );
00267         memcpy(&this->flatValueMatrix[i*this->matrixSize], this->valueMatrix[i],
00268                this->matrixSize*sizeof(long int ) );
00269     }
00270
00271 }
00272 };

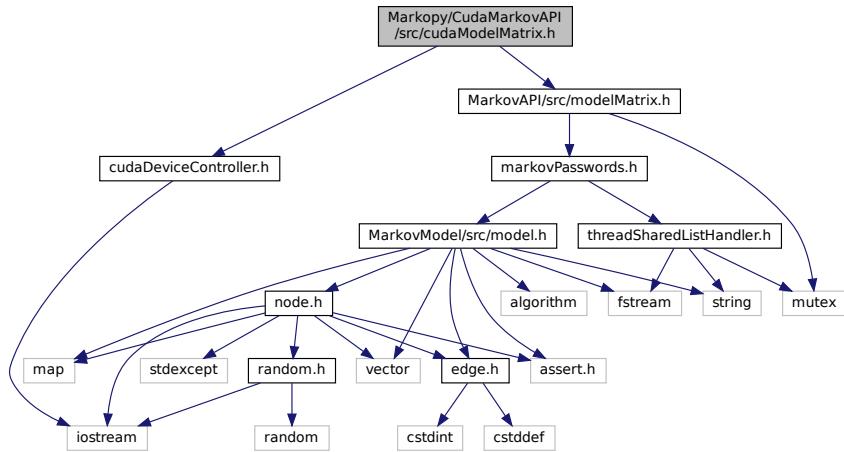
```

10.13 Markopy/CudaMarkovAPI/src/cudaModelMatrix.h File Reference

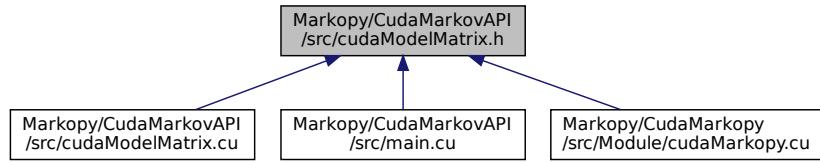
CUDA accelerated extension of [Markov::API::ModelMatrix](#).

```
#include "MarkovAPI/src/modelMatrix.h"
#include "cudaDeviceController.h"
```

Include dependency graph for cudaModelMatrix.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::API::CUDA::CUDAModelMatrix](#)

Extension of [Markov::API::ModelMatrix](#) which is modified to run on GPU devices.

Namespaces

- [Markov](#)
Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.
- [Markov::API](#)
Namespace for the [MarkovPasswords API](#).
- [Markov::API::CUDA](#)
Namespace for objects requiring [CUDA](#) libraries.

Functions

- `__global__ void Markov::API::CUDA::FastRandomWalkCUDAKernel (unsigned long int n, int minLen, int maxLen, char *outputBuffer, char *matrixIndex, long int *totalEdgeWeights, long int *valueMatrix, char *edgeMatrix, int matrixSize, int memoryPerKernelGrid, unsigned long *seed)`
CUDA kernel for the FastRandomWalk operation.
- `__device__ char * Markov::API::CUDA::strchr (char *p, char c, int s_len)`
*srtchr implementation on **device** space*

10.13.1 Detailed Description

CUDA accelerated extension of [Markov::API::ModelMatrix](#).

Authors

Ata Hakçıl

Extension of [Markov::API::ModelMatrix](#) which is modified to run on GPU devices. This implementation only supports Nvidia devices.

Class to flatten and reduce [Markov::Model](#) to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)

To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are synchronized and files are flushed every 50M operations.

Definition in file [cudaModelMatrix.h](#).

10.14 cudaModelMatrix.h

```

00001 /** @file cudaModelMatrix.h
00002 * @brief CUDA accelerated extension of Markov::API::ModelMatrix
00003 * @authors Ata Hakçıl
00004 *
00005 * @copydoc Markov::API::CUDA::CUDAModelMatrix
00006 */
00007
00008 #include "MarkovAPI/src/modelMatrix.h"
00009 #include "cudaDeviceController.h"
00010
00011 /** @brief Namespace for objects requiring CUDA libraries.
00012 */
00013 namespace Markov::API::CUDA{
00014     /** @brief Extension of Markov::API::ModelMatrix which is modified to run on GPU devices.
00015     *
00016     * This implementation only supports Nvidia devices.
00017     * @copydoc Markov::API::ModelMatrix
00018     */
00019     class CUDAModelMatrix : public Markov::API::ModelMatrix, public CUDADeviceController{
00020     public:
00021
00022         /** @brief Migrate the class members to the VRAM
00023         *
00024         * Cannot be used without calling Markov::API::ModelMatrix::ConstructMatrix at least once.
00025         * This function will manage the memory allocation and data transfer from CPU RAM to GPU VRAM.
00026         *
00027         * Newly allocated VRAM pointers are set in the class member variables.
00028         *
00029     */
00030     __host__ void MigrateMatrix();
00031
00032     /** @brief Flatten migrated matrix from 2d to 1d
00033     *
00034     *
00035     */
00036     __host__ void FlattenMatrix();
00037
00038     /** @brief Random walk on the Matrix-reduced Markov::Model
00039     *
00040     * TODO
00041     *
00042     *
00043     * @param n - Number of passwords to generate.
00044     * @param wordlistFileName - Filename to write to
00045     * @param minLen - Minimum password length to generate
00046     * @param maxLen - Maximum password length to generate
00047     * @param threads - number of OS threads to spawn
00048     * @param bFileIO - If false, filename will be ignored and will output to stdout.
00049     *
00050     *
00051     * @code{.cpp}
00052     * Markov::API::ModelMatrix mp;
00053     * mp.Import("models/finished.mdl");
00054     * mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
00055     * @endcode
00056     *
00057     */
00058     __host__ void FastRandomWalk(unsigned long int n, const char* wordlistFileName, int minLen,
00059     int maxLen, bool bFileIO, bool bInfinite);
00060     protected:

```

```

00061     /** @brief Allocate the output buffer for kernel operation
00062      *
00063      * TODO
00064      *
00065      *
00066      * @param n - Number of passwords to generate.
00067      * @param singleGenMaxLen - maximum string length for a single generation
00068      * @param CUDAKernelGridSize - Total number of grid members in CUDA kernel
00069      * @param sizePerGrid - Size to allocate per grid member
00070      * @return pointer to the allocation on VRAM
00071      *
00072      */
00073      *
00074      */
00075      __host__ char* AllocVRAMOutputBuffer(long int n, long int singleGenMaxLen, long int
00076      CUDAKernelGridSize, long int sizePerGrid);
00077      __host__ void LaunchAsyncKernel(int kernelID, int minLen, int maxLen);
00078      __host__ void prepKernelMemoryChannel(int numberOfStreams);
00079      __host__ void GatherAsyncKernelOutput(int kernelID, bool bFileIO, std::ofstream &wordlist);
00080
00081  private:
00082
00083      /**
00084       * @brief VRAM Address pointer of edge matrix (from modelMatrix.h)
00085      */
00086      char* device_edgeMatrix;
00087
00088      /**
00089       * @brief VRAM Address pointer of value matrix (from modelMatrix.h)
00090      */
00091      long int *device_valueMatrix;
00092
00093      /**
00094       * @brief VRAM Address pointer of matrixIndex (from modelMatrix.h)
00095      */
00096      char *device_matrixIndex;
00097
00098      /**
00099       * @brief VRAM Address pointer of total edge weights (from modelMatrix.h)
00100      */
00101      long int *device_totalEdgeWeights;
00102
00103      /**
00104       * @brief RandomWalk results in device
00105      */
00106      char** device_outputBuffer;
00107
00108      /**
00109       * @brief RandomWalk results in host
00110      */
00111      char** outputBuffer;
00112
00113      /**
00114       * @brief Adding Edge matrix end-to-end and resize to 1-D array for better performance on
00115       * traversing
00116      */
00117      char* flatEdgeMatrix;
00118
00119      /**
00120       * @brief Adding Value matrix end-to-end and resize to 1-D array for better performance on
00121       * traversing
00122      */
00123      long int* flatValueMatrix;
00124
00125      int cudaBlocks;
00126      int cudaThreads;
00127      int iterationsPerKernelThread;
00128      long int totalOutputPerSync;
00129      long int totalOutputPerKernel;
00130      int numberOfPartitions;
00131      int cudaGridsize;
00132      int cudaMemPerGrid;
00133      long int cudaPerKernelAllocationSize;
00134
00135      int alternatingKernels;
00136
00137      unsigned long** device_seeds;
00138
00139      cudaStream_t *cudastreams;
00140
00141  };
00142
00143  /** @brief CUDA kernel for the FastRandomWalk operation
00144  */

```

```

00145 * Will be initiated by CPU and continued by GPU (__global__ tag)
00146 *
00147 *
00148 * @param n - Number of passwords to generate.
00149 * @param minLen - minimum string length for a single generation
00150 * @param maxLen - maximum string length for a single generation
00151 * @param outputBuffer - VRAM ptr to the output buffer
00152 * @param matrixIndex - VRAM ptr to the matrix indices
00153 * @param totalEdgeWeights - VRAM ptr to the totalEdgeWeights array
00154 * @param valueMatrix - VRAM ptr to the edge weights array
00155 * @param edgeMatrix - VRAM ptr to the edge representations array
00156 * @param matrixSize - Size of the matrix dimensions
00157 * @param memoryPerKernelGrid - Maximum memory usage per kernel grid
00158 * @param seed - seed chunk to generate the random from (generated & used by Marsaglia)
00159 *
00160 *
00161 *
00162 */
00163 __global__ void FastRandomWalkCUDAKernel(unsigned long int n, int minLen, int maxLen, char*
00164 outputBuffer,
00165     char* matrixIndex, long int* totalEdgeWeights, long int* valueMatrix, char *edgeMatrix,
00166     int matrixSize, int memoryPerKernelGrid, unsigned long *seed); //, unsigned long mex, unsigned
00167     long mey, unsigned long mez);
00168
00169 /**
00170 * Find the first matching index of a string
00171 *
00172 *
00173 * @param p - string to check
00174 * @param c - character to match
00175 * @param s_len - maximum string length
00176 * @returns pointer to the match
00177 */
00178 __device__ char* strchr(char* p, char c, int s_len);
00179
00180 };

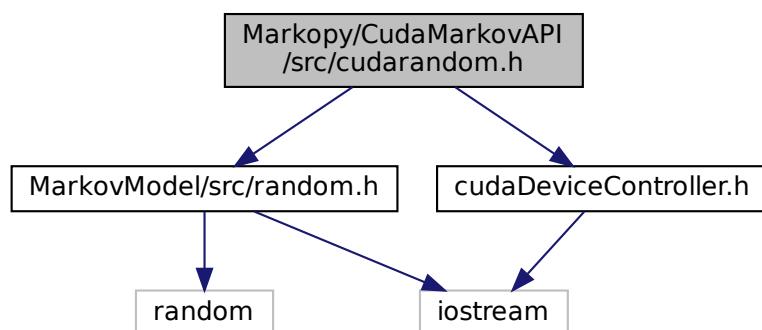
```

10.15 Markopy/CudaMarkovAPI/src/cudarandom.h File Reference

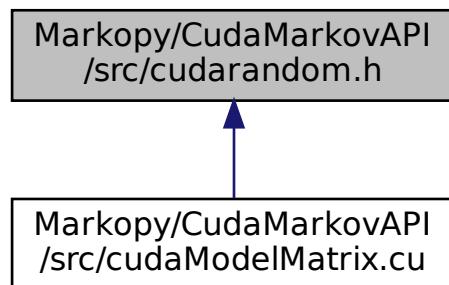
Extension of [Markov::Random::Marsaglia](#) for CUDA.

```
#include "MarkovModel/src/random.h"
#include "cudaDeviceController.h"
```

Include dependency graph for cudarandom.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::API::CUDA::Random::Marsaglia](#)

*Extension of [Markov::Random::Marsaglia](#) which is capable of working on **device** space.*

Namespaces

- [Markov](#)
Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.
- [Markov::API](#)
Namespace for the [MarkovPasswords API](#).
- [Markov::API::CUDA](#)
Namespace for objects requiring [CUDA](#) libraries.
- [Markov::API::CUDA::Random](#)
*Namespace for [Random](#) engines operable under **device** space.*

Functions

- `__device__ unsigned long Markov::API::CUDA::Random::devrandom (unsigned long &x, unsigned long &y, unsigned long &z)`

*Marsaglia Random Generation function operable in **device** space.*

10.15.1 Detailed Description

Extension of [Markov::Random::Marsaglia](#) for CUDA.

Authors

Ata Hakçıl

Implementation of Marsaglia Random Engine. This is an implementation of Marsaglia Random engine, which for most use cases is a better fit than other solutions. Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.

This implementation of the Marsaglia Engine is seeded by [random.h](#) default random engine. RandomEngine is only seeded once so its not a performance issue.

Example Use: Using Marsaglia Engine with RandomWalk

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
```

```

for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}

```

Example Use: Generating a random number with Marsaglia Engine

```

Markov::Random::Marsaglia me;
std::cout << me.random();

```

Definition in file [cudarandom.h](#).

10.16 cudarandom.h

```

00001 /** @file cudarandom.h
00002 * @brief Extension of Markov::Random::Marsaglia for CUDA
00003 * @authors Ata Hakçıl
00004 *
00005 * @copydoc Markov::Random::Marsaglia
00006 */
00007
00008 #pragma once
00009 #include "MarkovModel/src/random.h"
00010 #include "cudaDeviceController.h"
00011
00012 /** @brief Namespace for Random engines operable under __device__ space.
00013 */
00014 namespace Markov::API::CUDA::Random{
00015
00016     /** @brief Extension of Markov::Random::Marsaglia which is capable o working on __device__ space.
00017     *
00018     * @copydoc Markov::Random::Marsaglia
00019     */
00020     class Marsaglia : public Markov::Random::Marsaglia, public CUDADeviceController{
00021     public:
00022
00023         /** @brief Migrate a Marsaglia[] to VRAM as seedChunk
00024         * @param MEarr Array of Marsaglia Engines
00025         * @param gridSize GridSize of the CUDA Kernel, aka size of array
00026         * @returns pointer to the resulting seed chunk in device VRAM.
00027         */
00028         static unsigned long* MigrateToVRAM(Markov::API::CUDA::Random::Marsaglia *MEarr, long int
gridSize) {
00029             cudaError_t cudastatus;
00030             unsigned long* seedChunk;
00031             cudastatus = cudaMalloc((unsigned long**)&seedChunk, gridSize*3*sizeof(unsigned long));
00032             CudaCheckNotifyErr(cudastatus, "Failed to allocate seed buffer");
00033             unsigned long *temp = new unsigned long[gridSize*3];
00034             for(int i=0;i<gridSize;i++){
00035                 temp[i*3] = MEarr[i].x;
00036                 temp[i*3+1] = MEarr[i].y;
00037                 temp[i*3+2] = MEarr[i].z;
00038             }
00039             //for(int i=0;i<gridSize*3;i++) std::cout << temp[i] << "\n";
00040             cudaMemcpy(seedChunk, temp, gridSize*3*sizeof(unsigned long), cudaMemcpyHostToDevice);
00041             CudaCheckNotifyErr(cudastatus, "Failed to memcpy seed buffer.");
00042
00043             delete[] temp;
00044             return seedChunk;
00045         }
00046     };
00047
00048     /** @brief Marsaglia Random Generation function operable in __device__ space
00049     * @param x marsaglia internal x. Not constant, (ref)
00050     * @param y marsaglia internal y. Not constant, (ref)
00051     * @param z marsaglia internal z. Not constant, (ref)
00052     * @returns returns z
00053     */
00054     __device__ unsigned long devrandom(unsigned long &x, unsigned long &y, unsigned long &z){
00055         unsigned long t;
00056         x ^= x << 16;
00057         x ^= x >> 5;
00058         x ^= x << 1;
00059
00060         t = x;
00061         x = y;
00062         y = z;
00063         z = t ^ x ^ y;
00064
00065         return z;
00066     }
00067 };

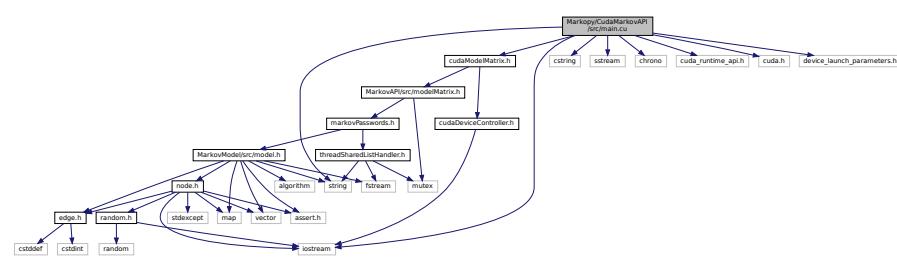
```

10.17 Markopy/CudaMarkovAPI/src/main.cu File Reference

Simple test file to check libcudamarkov.

```
#include <iostream>
#include <string>
#include <cstring>
#include <sstream>
#include <chrono>
#include "cudaModelMatrix.h"
#include <cuda_runtime_api.h>
#include <cuda.h>
#include <device_launch_parameters.h>
```

Include dependency graph for main.cu:



Functions

- int [main](#) (int argc, char **argv)

10.17.1 Detailed Description

Simple test file to check libcudamarkov.

Authors

Ata Hakçıl

Controller class for CUDA device. This implementation only supports Nvidia devices.

Definition in file [main.cu](#).

10.17.2 Function Documentation

10.17.2.1 [main\(\)](#)

```
int main (
    int argc,
    char ** argv )
Definition at line 21 of file main.cu.
00021
00022
00023
00024
00025     Markov::API::CUDA::CUDAModelMatrix markovPass;
00026     std::cerr << "Importing model.\n";
00027     markovPass.Import("models/finished.mdl");
00028     markovPass.Buff("!\\"#$%&'()*+,-./:;<=>?@{\\}^`{|}{|}~", 10, true, true);
00029     std::cerr << "Import done. \n";
00030     markovPass.ConstructMatrix();
00031 //markovPass.DumpJSON();
00032     CUDADeviceController::ListCudaDevices();
00033
00034     std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
00035     std::cerr << "Starting walk. \n";
```

```

00036     markovPass.FastRandomWalk(1000000000,"/media/ignis/Stuff/wordlist.txt",12,12, false, false);
00037     //markovPass.FastRandomWalk(500000000,"/media/ignis/Stuff/wordlist2.txt",6,12,25, true);
00038     std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
00039
00040     std::cerr << "Finished in:" << std::chrono::duration_cast<std::chrono::milliseconds> (end -
00041 begin).count() << " milliseconds" << std::endl;
00042
00043 }

```

10.18 main.cu

```

00001 /**
00002 * @file main.cu
00003 * @brief Simple test file to check libcudamarkov
00004 * @authors Ata Hakçıl
00005 * @copydoc Markov::API::CudaModelMatrix
00006 * @copydoc Markov::API::CUDADeviceController
00007 */
00008
00009 #include <iostream>
00010 #include <string>
00011 #include <cstring>
00012 #include <sstream>
00013 #include <chrono>
00014 #include "cudaModelMatrix.h"
00015 #include <cuda_runtime_api.h>
00016 #include <cuda.h>
00017 #include <device_launch_parameters.h>
00018
00019 using Markov::API::CUDA::CUDADeviceController;
00020
00021 int main(int argc, char** argv) {
00022
00023
00024
00025     Markov::API::CUDA::CUDAModelMatrix markovPass;
00026     std::cerr << "Importing model.\n";
00027     markovPass.Import("models/finished.mdl");
00028     markovPass.Buff("!\\"#$%&'()*+,-./:;<=>?@[\\"{}~", 10, true, true);
00029     std::cerr << "Import done.\n";
00030     markovPass.ConstructMatrix();
00031     //markovPass.DumpJSON();
00032     CUDADeviceController::ListCudaDevices();
00033
00034     std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
00035     std::cerr << "Starting walk. \n";
00036     markovPass.FastRandomWalk(1000000000,"/media/ignis/Stuff/wordlist.txt",12,12, false, false);
00037     //markovPass.FastRandomWalk(500000000,"/media/ignis/Stuff/wordlist2.txt",6,12,25, true);
00038     std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
00039
00040     std::cerr << "Finished in:" << std::chrono::duration_cast<std::chrono::milliseconds> (end -
00041 begin).count() << " milliseconds" << std::endl;
00042
00043 }

```

10.19 Markopy/documentation/dirs.docs File Reference

doxygen information about the directories

10.19.1 Detailed Description

doxygen information about the directories

Definition in file [dirs.docs](#).

10.20 dirs.docs

```

00001 /**
00002 *
00003 * @file dirs.docs
00004 * @brief doxygen information about the directories
00005 */
00006
00007 /**
00008 * @dir Markopy/Markopy

```

```

00009 * @brief CPython extension for MarkovAPI
00010 */
00011
00012 /**
00013 * @dir Markopy/Markopy/src/CLI
00014 * @brief Python CLI for Markopy
00015 */
00016
00017 /**
00018 * @dir Markopy/Markopy/src/Module
00019 * @brief CPP module for Markopy
00020 */
00021
00022 /**
00023 * @dir Markopy/CudaMarkopy
00024 * @brief CPython extension for CudaMarkovAPI
00025 */
00026
00027 /**
00028 * @dir Markopy/CudaMarkopy/src/CLI
00029 * @brief Python CLI for CudaMarkopy
00030 */
00031
00032 /**
00033 * @dir Markopy/CudaMarkopy/src/Module
00034 * @brief CPP module for CudaMarkopy
00035 */
00036
00037 /**
00038 * @dir Markopy/MarkovAPI
00039 * @brief Base project
00040 */
00041
00042 /**
00043 * @dir Markopy/CudaMarkovAPI
00044 * @brief Markov API with CUDA Acceleration
00045 */
00046
00047 /**
00048 * @dir Markopy/MarkovAPICLI
00049 * @brief Test utility for MarkovAPI
00050 */
00051
00052 /**
00053 * @dir Markopy/MarkovModel
00054 * @brief Header only markov library
00055 */
00056
00057 /**
00058 * @dir Markopy/MarkovPasswordsGUI
00059 * @brief Graphical Interface for MarkovAPI
00060 */
00061
00062 /**
00063 * @dir Markopy/UnitTests
00064 * @brief Unit tests for MarkovAPI
00065 */
00066
00067 /**
00068 * @dir Markopy/documentation
00069 * @brief Files related to documentation generation
00070 */

```

10.21 Markopy/Markopy/src/CLI/base.py File Reference

base command line interface for python

Classes

- class [Python.Markopy.BaseCLI](#)
Base CLI class to handle user interactions
- class [Python.Markopy.AbstractGenerationModelCLI](#)
abstract class for generation capable models
- class [Python.Markopy.AbstractTrainingModelCLI](#)
abstract class for training capable models

Namespaces

- base

10.21.1 Detailed Description

base command line interface for python

Definition in file [base.py](#).

10.22 base.py

```

00001 #!/usr/bin/python3
00002
00003
00007
00008 import argparse
00009 import allegate as logging
00010 import os
00011 from abc import abstractmethod
00012 from termcolor import colored
00013 from mm import MarkovModel
00014
00015
00016 class BaseCLI():
00017     """! @brief Base CLI class to handle user interactions
00018         @belongsto Python::Markopy
00019     """
00020     def __init__(self, add_help : bool=True):
00021         """
00022             @brief initialize base CLI
00023             @param add_help decide to overload the help function or not
00024         """
00025         self.parserparser = argparse.ArgumentParser(description="Python wrapper for MarkovPasswords.",
00026                                         epilog=f"""{colored("Sample runs:", "yellow")}
00027 {__file__.split('/')[-1]} train untrained.mdl -d dataset.dat -s "\\t" -o trained.mdl
00028             Import untrained.mdl, train it with dataset.dat which has tab delimited data, output
00029             resulting model to trained.mdl\n
00030 {__file__.split('/')[-1]} generate trained.mdl -n 500 -w output.txt
00031             Import trained.mdl, and generate 500 lines to output.txt
00032
00033 {__file__.split('/')[-1]} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt
00034             Train and immediately generate 500 lines to output.txt. Do not export trained model.
00035
00036 {__file__.split('/')[-1]} combine untrained.mdl -d dataset.dat -s "\\t" -n 500 -w output.txt
00037             -o trained.mdl
00038             Train and immediately generate 500 lines to output.txt. Export trained model.
00039             """", add_help=add_help, formatter_class=argparse.RawTextHelpFormatter)
00040             self.print_helpprint_help = self.parserparser.print_help
00041             self.modelmodel = MarkovModel()
00042
00043             @abstractmethod
00044             def add_arguments(self):
00045                 "! @brief Add command line arguments to the parser"
00046                 self.parserparser.add_argument("mode",                                     help="Process mode. Either
00047 'Train', 'Generate', or 'Combine'.")
00048                 self.parserparser.add_argument("-t", "--threads", default=10,           help="Number of lines to
00049 generate. Ignored in training mode.")
00050                 self.parserparser.add_argument("-v", "--verbosity", action="count",       help="Output verbosity.")
00051                 self.parserparser.add_argument("-b", "--bulk", action="store_true",    help="Bulk generate or bulk
00052 train every corpus/model in the folder.")
00053
00054             @abstractmethod
00055             def help(self):
00056                 "! @brief Handle help strings. Defaults to argparse's help"
00057                 self.print_helpprint_help()
00058
00059             def parse(self):
00060                 "! @brief add, parse and hook arguments"
00061                 self.add_argumentsadd_arguments()
00062                 self.parse_argumentsparse_arguments()
00063                 self.init_post_argumentsinit_post_arguments()
00064
00065             @abstractmethod
00066             def init_post_arguments(self):
00067                 "! @brief set up stuff that is collected from command line arguments"
00068                 logging.VERBOSITY = 0
00069                 try:
00070                     if self.argsargs.verbosity:
00071                         logging.VERBOSITY = self.argsargs.verbosity
00072                         logging pprint(f"Verbosity set to {self.args.verbosity}.", 2)
00073                 except:

```

```

00070         pass
00071
00072     @abstractmethod
00073     def parse_arguments(self):
00074         """!
00075         @brief trigger parser
00076         @param argsargs = self.parserparser.parse_known_args() [0]
00077
00078     def import_model(self, filename : str):
00079         """!
00080         @brief Import a model file
00081         @param filename filename to import
00082         """
00083         logging pprint("Importing model file.", 1)
00084
00085         if not self.check_import_pathcheck_import_path(filename):
00086             logging pprint(f"Model file at {filename} not found. Check the file path, or working
00087             directory")
00088             return False
00089
00090         self.modelmodel.Import(filename)
00091         logging pprint("Model imported successfully.", 2)
00092         return True
00093
00094     def train(self, dataset : str, seperator : str, output : str, output_forced : bool=False, bulk :
00095         bool=False):
00096         """!
00097         @brief Train a model via CLI parameters
00098         @param model Model instance
00099         @param dataset filename for the dataset
00100         @param seperator seperator used with the dataset
00101         @param output output filename
00102         @param output_forced force overwrite
00103         @param bulk marks bulk operation with directories
00104         """
00105         logging pprint("Training.")
00106
00107         if not (dataset and seperator and (output or not output_forced)):
00108             logging pprint(f"Training mode requires -d/--dataset('', -o/--output' if output_forced
00109             else') and -s/--seperator parameters. Exiting.")
00110             return False
00111
00112         if not bulk and not self.check_corpus_pathcheck_corpus_path(dataset):
00113             logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00114             return False
00115
00116         if not self.check_export_pathcheck_export_path(output):
00117             logging pprint(f"Cannot create output at {output}")
00118             return False
00119
00120         if(seperator == '\\t'):
00121             logging pprint("Escaping seperator.", 3)
00122             seperator = '\t'
00123
00124         if(len(seperator)!=1):
00125             logging pprint(f'Delimiter must be a single character, and "{seperator}" is not
00126             accepted.')
00127             exit(4)
00128
00129         logging pprint(f'Starting training.', 3)
00130         self.modelmodel.Train(dataset,seperator, int(self.argsargs.threads))
00131         logging pprint(f'Training completed.', 2)
00132
00133         if(output):
00134             logging pprint(f'Exporting model to {output}', 2)
00135             self.exportexport(output)
00136         else:
00137             logging pprint(f'Model will not be exported.', 1)
00138
00139     def export(self, filename : str):
00140         """!
00141         @brief Export model to a file
00142         @param filename filename to export to
00143         """
00144         self.modelmodel.Export(filename)
00145
00146     def generate(self, wordlist : str, bulk : bool=False):
00147         """!
00148         @brief Generate strings from the model
00149         @param model: model instance
00150         @param wordlist wordlist filename
00151         @param bulk marks bulk operation with directories
00152         """
00153         if not (wordlist or self.argsargs.count):

```

```

00153         logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters.
00154             Exiting.")
00155             return False
00156     if(bulk and os.path.isfile(wordlist)):
00157         logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00158         self._generate_generate(wordlist)
00159
00160     @abstractmethod
00161     def _generate(self, wordlist : str):
00162         """
00163             @brief wrapper for generate function. This can be overloaded by other models
00164             @param wordlist filename to generate to
00165             """
00166             self.modelmodel.Generate(int(self.argsargs.count), wordlist, int(self.argsargs.min),
00167             int(self.argsargs.max), int(self.argsargs.threads))
00168
00169     @staticmethod
00170     def check_import_path(filename : str):
00171         """
00172             @brief check import path for validity
00173             @param filename filename to check
00174             """
00175             if(not os.path.isfile(filename)):
00176                 return False
00177             else:
00178                 return True
00179
00180     @staticmethod
00181     def check_corpus_path(filename : str):
00182         """
00183             @brief check import path for validity
00184             @param filename filename to check
00185             """
00186
00187             if(not os.path.isfile(filename)):
00188                 return False
00189             return True
00190
00191     @staticmethod
00192     def check_export_path(filename : str):
00193         """
00194             @brief check import path for validity
00195             @param filename filename to check
00196             """
00197
00198             if(filename and os.path.isfile(filename)):
00199                 return True
00200             return True
00201
00202     def process(self):
00203         """
00204             @brief Process parameters for operation
00205             """
00206             if(self.argsargs.bulk):
00207                 logging pprint(f"\"Bulk mode operation chosen.\", 4)
00208                 if (self.argsargs.mode.lower() == "train"):
00209                     if (os.path.isdir(self.argsargs.output) and not os.path.isfile(self.argsargs.output))
00210                         and (os.path.isdir(self.argsargs.dataset) and not os.path.isfile(self.argsargs.dataset)):
00211                             corpus_list = os.listdir(self.argsargs.dataset)
00212                             for corpus in corpus_list:
00213                                 self.import_modelimport_model(self.argsargs.input)
00214                                 logging pprint(f"\"Training {self.args.input} with {corpus}\", 2)
00215                                 output_file_name = corpus
00216                                 model_extension = ""
00217                                 if "." in self.argsargs.input:
00218                                     model_extension = self.argsargs.input.split(".")[-1]
00219                                     self.traintrain(f"\"{self.args.dataset}/{corpus}\", self.argsargs.separator,
00220                                     f"\"{self.args.output}/{corpus}.\"{model_extension}\", output_forced=True, bulk=True)
00221                                     else:
00222                                         logging pprint("In bulk training, output and dataset should be a directory.")
00223                                         exit(1)
00224
00225                                     elif (self.argsargs.mode.lower() == "generate"):
00226                                         if (os.path.isdir(self.argsargs.wordlist) and not
00227                                         os.path.isfile(self.argsargs.wordlist)) and (os.path.isdir(self.argsargs.input) and not
00228                                         os.path.isfile(self.argsargs.input)):
00229                                             model_list = os.listdir(self.argsargs.input)
00230                                             print(model_list)
00231                                             for input in model_list:
00232                                                 logging pprint(f"\"Generating from {self.args.input}/{input} to
00233                                                 {self.args.wordlist}/{input}.txt\", 2)
00234                                                 self.import_modelimport_model(f"\"{self.args.input}/{input}\")
00235                                                 model_base = input
00236                                                 if "." in self.argsargs.input:
00237                                                     model_base = input.split(".")[-1]

```

```

00233             self.generategenerate(f"{self.args.wordlist}/{model_base}.txt", bulk=True)
00234         else:
00235             logging.pprint("In bulk generation, input and wordlist should be directory.")
00236
00237     else:
00238         self.import_modelimport_model(self.argsargs.input)
00239         if (self.argsargs.mode.lower() == "generate"):
00240             self.generategenerate(self.argsargs.wordlist)
00241
00242
00243         elif (self.argsargs.mode.lower() == "train"):
00244             self.traintrain(self.argsargs.dataset, self.argsargs.seperator, self.argsargs.output,
00245             output_forced=True)
00246
00247         elif(self.argsargs.mode.lower() == "combine"):
00248             self.traintrain(self.argsargs.dataset, self.argsargs.seperator, self.argsargs.output)
00249             self.generategenerate(self.argsargs.wordlist)
00250
00251
00252     else:
00253         logging.pprint("Invalid mode arguement given.")
00254         logging.pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00255         exit(5)
00256
00257 class AbstractGenerationModelCLI(BaseCLI):
00258 """
00259     @brief abstract class for generation capable models
00260     @belongsto Python::Markopy
00261     @extends Python::Markopy::BaseCLI
00262 """
00263     @abstractmethod
00264     def add_arguments(self):
00265         "Add command line arguements to the parser"
00266         super().add_arguments()
00267         self.parserparser.add_argument("input",
00268             This model will be imported before starting operation.)
00269             self.parserparser.add_argument("-w", "--wordlist",
00270                 export generation results to. Will be ignored for training mode")
00271             self.parserparser.add_argument("--min", default=6,
00272                 is allowed during generation")
00273             self.parserparser.add_argument("--max", default=12,
00274                 is allowed during generation")
00275             self.parserparser.add_argument("-n", "--count",
00276                 generate. Ignored in training mode.")
00277
00278
00279 class AbstractTrainingModelCLI(AbstractGenerationModelCLI, BaseCLI):
00280 """
00281     @brief abstract class for training capable models
00282     @belongsto Python::Markopy
00283     @extends Python::Markopy::BaseCLI
00284     @extends Python::Markopy::AbstractGenerationModelCLI
00285 """
00286     @abstractmethod
00287     def add_arguments(self):
00288         "Add command line arguements to the parser"
00289         self.parserparser.add_argument("-o", "--output",
00290             This model will be exported when done. Will be ignored for generation mode.)
00291             self.parserparser.add_argument("-d", "--dataset",
00292                 input from for training. Will be ignored for generation mode")
00293             self.parserparser.add_argument("-s", "--seperator",
00294                 to use with training data.(character between occurrence and value)")
00295             super().add_arguments()

```

10.23 Markopy/Markopy/src/CLI/importer.py File Reference

dynamic import wrapper for markopy model

Namespaces

- [importer](#)

Functions

- def [importer.import_markopy \(\)](#)

import and return markopy module

10.23.1 Detailed Description

dynamic import wrapper for markopy model
Definition in file [importer.py](#).

10.24 importer.py

```

00001
00002
00006
00007 from importlib.util import spec_from_loader, module_from_spec
00008 from importlib.machinery import SourceFileLoader, ExtensionFileLoader
00009 import os
00010
00011
00012 def import_markopy():
00013     """! @brief import and return markopy module
00014         @returns markopy module
00015     """
00016     ext = "so"
00017     if os.name == 'nt':
00018         ext="pyd"
00019     try:
00020         spec = spec_from_loader("markopy", ExtensionFileLoader("markopy",
00021             os.path.abspath(f"markopy.{ext}")))
00022         markopy = module_from_spec(spec)
00023         spec.loader.exec_module(markopy)
00024         return markopy
00025     except ImportError as e:
00026         print(f"({_file_}) Working in development mode. Trying to load markopy.{ext} from
00027             ../../../../out/")
00028         if(os.path.exists(f"../../../../out/lib/markopy.{ext}")):
00029             spec = spec_from_loader("markopy", ExtensionFileLoader("markopy",
00030                 os.path.abspath(f"../../../../out/lib/markopy.{ext}")))
00031             markopy = module_from_spec(spec)
00032             spec.loader.exec_module(markopy)
00033             return markopy
00034     else:
00035         raise e

```

10.25 Markopy/Markopy/src/CLI/markopy.py File Reference

Entry point for markopy scripts.

Classes

- class [Python.Markopy.MarkopyCLI](#)
Top level model selector for Markopy CLI.

Namespaces

- [markopy](#)
- [Python.Markopy](#)
wrapper scripts for Markopy

Variables

- string [markopy.ext](#) = "so"
- [markopy.spec](#) = spec_from_loader("markopy", ExtensionFileLoader("markopy", os.path.abspath(f"markopy.{ext}")))
- [markopy.markopy](#) = module_from_spec(spec)
- [markopy.mp](#) = MarkopyCLI()

10.25.1 Detailed Description

Entry point for markopy scripts.
Definition in file [markopy.py](#).

10.26 markopy.py

```

00001 #!/usr/bin/env python3
00002
00003
00004
00005
00006
00007
00008
00009
00010
00011
00012
00013
00014
00015
00016
00017
00018
00019 from importlib.util import spec_from_loader, module_from_spec
00020 from importlib.machinery import SourceFileLoader, ExtensionFileLoader
00021 import os
00022 import sys
00023
00024 ext = "so"
00025 if os.name == 'nt':
00026     ext="pyd"
00027 try:
00028     spec = spec_from_loader("markopy", ExtensionFileLoader("markopy",
00029         os.path.abspath(f"markopy.{ext}")))
00030     markopy = module_from_spec(spec)
00031     spec.loader.exec_module(markopy)
00032 except ImportError as e:
00033     print(f"({_file_}) Working in development mode. Trying to load markopy.{ext} from
00034     ../../../{out}/")
00035     if(os.path.exists("../../../{out}/lib/markopy.so")):
00036         spec = spec_from_loader("markopy", ExtensionFileLoader("markopy",
00037             os.path.abspath(f"../../../{out}/lib/markopy.{ext}")))
00038         markopy = module_from_spec(spec)
00039         spec.loader.exec_module(markopy)
00040     else:
00041         raise e
00042
00043 try:
00044     from base import BaseCLI
00045     from mp import MarkovPasswordsCLI
00046     from mmx import ModelMatrixCLI
00047
00048 except ModuleNotFoundError as e:
00049     print("Working in development mode. Trying to load markopy.py from ../../../Markopy/")
00050     if(os.path.exists("../../../Markopy/src/CLI/base.py")):
00051         sys.path.insert(1, '../../../Markopy/src/CLI/')
00052         from base import BaseCLI
00053         from mp import MarkovPasswordsCLI
00054         from mmx import ModelMatrixCLI
00055     else:
00056         raise e
00057
00058
00059 from termcolor import colored
00060 from abc import abstractmethod
00061
00062 class MarkopyCLI(BaseCLI):
00063     """
00064         @brief Top level model selector for Markopy CLI.
00065         This class is used for injecting the -mt parameter to the CLI, and determining the model type
00066         depending on that.
00067         @belongsto Python::Markopy
00068         @extends Python::Markopy::BaseCLI
00069         @extends Python::Markopy::ModelMatrixCLI
00070         @extends Python::Markopy::MarkovPasswordsCLI
00071     """
00072
00073     def __init__(self, add_help=False):
00074         """
00075             @brief default constructor
00076         """
00077         BaseCLI.__init__(self,add_help)
00078         self.argsargsargs = None
00079         self.parserparser.epilog = f"""
00080             {colored("Sample runs:", "yellow")}
00081             {_file_.split('/')[-1]} -mt MP generate trained.mdl -n 500 -w output.txt
00082                 Import trained.mdl, and generate 500 lines to output.txt
00083
00084             {_file_.split('/')[-1]} -mt MMX generate trained.mdl -n 500 -w output.txt
00085                 Import trained.mdl, and generate 500 lines to output.txt
00086
00087         @abstractmethod
00088         def add_arguments(self):
00089             """
00090                 @brief add -mt/--model_type constructor
00091             """
00092             self.parserparser.add_argument("-mt", "--model_type", default="_MMX", help="Model type to use.

```

```

Accepted values: MP, MMX")
00091     self.parserparser.add_argument("-h", "--help", action="store_true", help="Model type to use.
Accepted values: MP, MMX")
00092     self.parserparser.print_help = self.helphelp
00093
00094     @abstractmethod
00095     def help(self):
00096         """
00097             @brief overload help function to print submodel helps
00098         """
00099         self.parserparser.print_help = self.stub
00100         self.argsargsargs = self.parserparser.parse_known_args() [0]
00101         if(self.argsargsargs.model_type!="_MMX"):
00102             if(self.argsargsargs.model_type=="MP"):
00103                 mp = MarkovPasswordsCLI()
00104                 mp.add_arguments()
00105                 mp.parser.print_help()
00106             elif(self.argsargsargs.model_type=="MMX"):
00107                 mp = ModelMatrixCLI()
00108                 mp.add_arguments()
00109                 mp.parser.print_help()
00110         else:
00111             print(colored("Model Mode selection choices:", "green"))
00112             self.print_helpprint_help()
00113             print(colored("Following are applicable for -mt MP mode:", "green"))
00114             mp = MarkovPasswordsCLI()
00115             mp.add_arguments()
00116             mp.parser.print_help()
00117             print(colored("Following are applicable for -mt MMX mode:", "green"))
00118             mp = ModelMatrixCLI()
00119             mp.add_arguments()
00120             mp.parser.print_help()
00121
00122         exit()
00123
00124
00125     @abstractmethod
00126     def parse(self):
00127         """
00128             @brief overload parse function to parse for submodels"
00129             self.add_argumentsadd_argumentsadd_argumentsadd_argumentsadd_arguments()
00130             self.parse_argumentsparse_arguments()
00131             self.init_post_argumentsinit_post_argumentsinit_post_argumentsinit_post_arguments()
00132             if(self.argsargsargs.model_type == "MP"):
00133                 self.clicli = MarkovPasswordsCLI()
00134             elif(self.argsargsargs.model_type == "MMX" or self.argsargsargs.model_type == "_MMX"):
00135                 self.clicli = ModelMatrixCLI()
00136             else:
00137                 self.parse_failparse_fail()
00138
00139             if(self.argsargsargs.help): return self.helphelp
00140             self.clicli.parse()
00141
00142     @abstractmethod
00143     def init_post_arguments(self):
00144         pass
00145
00146     @abstractmethod
00147     def parse_fail(self):
00148         """
00149             @brief failed to parse model type"
00150             print("Unrecognized model type.")
00151             exit()
00152
00153     def process(self):
00154         """
00155             @brief pass the process request to selected submodel"
00156             return self.clicli.process()
00157
00158     def stub(self):
00159         """
00160             @brief stub function to hack help requests"
00161             return
00162
00163     if __name__ == "__main__":
00164         mp = MarkopyCLI()
00165         mp.parse()
00166         mp.process()

```

10.27 Markopy/Markopy/src/CLI/mm.py File Reference

Abstract representation of CPP/Python intermediate layer classes.

Classes

- class [Python.Markopy.MarkovModel](#)
Abstract representation of a markov model.
- class [Python.Markopy.ModelMatrix](#)
Abstract representation of a matrix based model.

Namespaces

- [mm](#)

Variables

- [mm.markopy](#) = import_markopy()

10.27.1 Detailed Description

Abstract representation of CPP/Python intermediate layer classes.

Definition in file [mm.py](#).

10.28 mm.py

```

00001
00002
00003
00007
00008 from abc import abstractmethod
00009
00010 from importer import import_markopy
00011 markopy = import_markopy()
00012
00013 class MarkovModel(markopy.MarkovPasswords):
00014     """
00015     @brief Abstract representation of a markov model
00016     @implements Markov::API::MarkovPasswords
00017     @belongsto Python::Markopy
00018
00019     To help with the python-cpp gateway documentation.
00020     """
00021     @abstractmethod
00022     def Import(filename : str):
00023         pass
00024
00025     @abstractmethod
00026     def Export(filename : str):
00027         pass
00028
00029     @abstractmethod
00030     def Train(dataset: str, seperator : str, threads : int):
00031         pass
00032
00033     @abstractmethod
00034     def Generate(count : int, wordlist : str, minlen : int, maxlen: int, threads : int):
00035         pass
00036
00037
00038 class ModelMatrix(markopy.ModelMatrix):
00039     """
00040     @brief Abstract representation of a matrix based model
00041     @implements Markov::API::ModelMatrix
00042     @belongsto Python::Markopy
00043
00044     To help with the python-cpp gateway documentation.
00045     """
00046
00047     @abstractmethod
00048     def FastRandomWalk(count : int, wordlist : str, minlen : int, maxlen : int):
00049         pass

```

10.29 Markopy/Markopy/src/CLI/mmx.py File Reference

ModelMatrix CLI wrapper.

Classes

- class [Python.Markopy.ModelMatrixCLI](#)

Extension of Python.Markopy.Base.BaseCLI for Markov::API::ModelMatrix.

Namespaces

- [mmx](#)

Variables

- [mmx.markopy](#) = import_markopy()
- [mmx.mp](#) = ModelMatrixCLI()

10.29.1 Detailed Description

ModelMatrix CLI wrapper.

Definition in file [mmx.py](#).

10.30 mmx.py

```
00001 #!/usr/bin/python3
00002
00003
00007
00008
00009 from mm import ModelMatrix
00010
00011 from importer import import_markopy
00012 markopy = import_markopy()
00013
00014 from base import BaseCLI, AbstractGenerationModelCLI
00015 import os
00016 import allogate as logging
00017
00018 class ModelMatrixCLI(AbstractGenerationModelCLI, ModelMatrix):
00019     """
00020         @brief Extension of Python.Markopy.Base.BaseCLI for Markov::API::ModelMatrix
00021         @belongsto Python::Markopy
00022         @extends Python::Markopy::AbstractGenerationModelCLI
00023         @extends Python::Markopy::ModelMatrix
00024
00025             adds -st/--stdout argument to the command line.
00026     """
00027     def __init__(self, add_help:bool=True):
00028         """
00029             @brief initialize model with Markov::API::ModelMatrix
00030         """
00031         super().__init__(add_help)
00032         self.modelmodelmodel = markopy.ModelMatrix()
00033
00034     def add_arguments(self):
00035         super().add_arguments()
00036         self.parserparser.add_argument("-st", "--stdout", action="store_true", help="Stdout mode")
00037
00038     def init_post_arguments(self):
00039         super().init_post_arguments()
00040         self.fileIOfileIO = not self.argsargs.stdout
00041
00042     def _generate(self, wordlist : str, ):
00043         self.modelmodelmodel.FastRandomWalk(int(self.argsargs.count), wordlist,
00044             int(self.argsargs.min), int(self.argsargs.max), int(self.argsargs.threads), self.fileIOfileIO)
00045
00046 if __name__ == "__main__":
00047     mp = ModelMatrixCLI()
00048     mp.parse()
00049     mp.process()
```

10.31 Markopy/Markopy/src/CLI/mp.py File Reference

CLI wrapper for MarkovPasswords.

Classes

- class [Python.Markopy.MarkovPasswordsCLI](#)

Extension of Python.Markopy.Base.BaseCLI for Markov::API::MarkovPasswords.

Namespaces

- [mp](#)

Variables

- [mp.markopy](#) = import_markopy()
- [mp.mp](#) = MarkovPasswordsCLI()

10.31.1 Detailed Description

CLI wrapper for MarkovPasswords.

Definition in file [mp.py](#).

10.32 mp.py

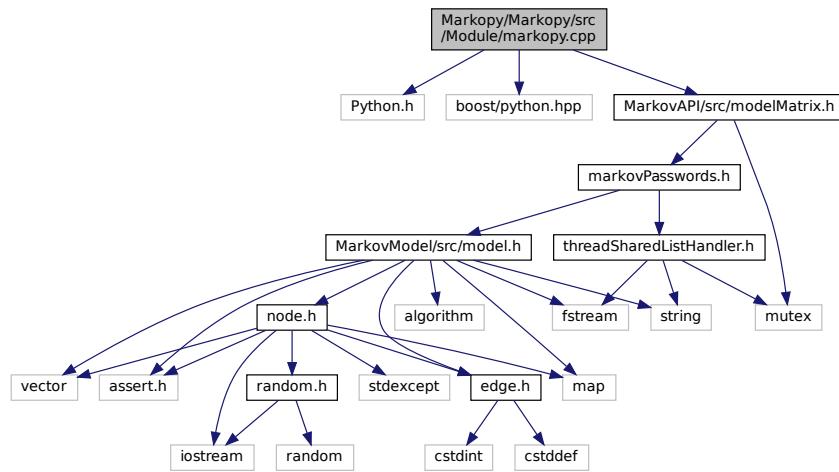
```
00001 #!/usr/bin/python3
00002
00003
00007
00008 from importlib.util import spec_from_loader, module_from_spec
00009 from importlib.machinery import SourceFileLoader, ExtensionFileLoader
00010 import os
00011 from mm import MarkovModel
00012 from importer import import_markopy
00013 markopy = import_markopy()
00014
00015 from base import BaseCLI, AbstractGenerationModelCLI, AbstractTrainingModelCLI
00016
00017 class MarkovPasswordsCLI(AbstractTrainingModelCLI, MarkovModel):
00018     """
00019         @brief Extension of Python.Markopy.Base.BaseCLI for Markov::API::MarkovPasswords
00020         @belongsto Python::Markopy
00021         @extends Python::Markopy::MarkovModel
00022         @extends Python::Markopy::AbstractTrainingModelCLI
00023
00024     adds -st/--stdout argument to the command line.
00025     """
00026     def __init__(self, add_help=True):
00027         """
00028             @brief initialize model with Markov::API::MarkovPasswords
00029         """
00030         super().__init__(add_help)
00031         self.modelmodelmodel = markopy.MarkovPasswords()
00032
00033     def _generate(self, wordlist):
00034         """
00035             @brief map generation function to Markov::API::MarkovPasswords::Generate"
00036             @param wordlist
00037             @param int(argsargs.count), wordlist, int(argsargs.min),
00038             int(argsargs.max), int(argsargs.threads)
00039
00040     if __name__ == "__main__":
00041         mp = MarkovPasswordsCLI()
00042         mp.parse()
00043         mp.process()
```

10.33 Markopy/Markopy/src/Module/markopy.cpp File Reference

CPython wrapper for libmarkov utils.

```
#include <Python.h>
#include <boost/python.hpp>
#include <MarkovAPI/src/modelMatrix.h>
```

Include dependency graph for markopy.cpp:



Namespaces

- [Markov](#)
Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.
- [Markov::Markopy](#)
CPython module for [Markov::API](#) objects.

Macros

- `#define BOOST_ALL_STATIC_LIB 1`
- `#define BOOST_PYTHON_STATIC_LIB 1`

Functions

- [Markov::Markopy::BOOST_PYTHON_MODULE](#) (markopy)

10.33.1 Detailed Description

CPython wrapper for libmarkov utils.

Authors

Ata Hakçıl, Celal Sahir Çetiner

This file is a wrapper for libmarkov utilities, exposing:

- MarkovPasswords
 - Import
 - Export
 - Train
 - Generate
- ModelMatrix
 - Import
 - Export

- Train
- ConstructMatrix
- DumpJSON
- FastRandomWalk

[Markov::Model](#) with char represented nodes. Includes wrappers for [Markov::Model](#) and additional helper functions to handle file I/O

This class is an extension of [Markov::Model<char>](#), with higher level abstractions such as train and generate. Class to flatten and reduce [Markov::Model](#) to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)

To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are synchronized and files are flushed every 50M operations.

Definition in file [markopy.cpp](#).

10.33.2 Macro Definition Documentation

10.33.2.1 BOOST_ALL_STATIC_LIB

```
#define BOOST_ALL_STATIC_LIB 1
```

Definition at line [24](#) of file [markopy.cpp](#).

10.33.2.2 BOOST_PYTHON_STATIC_LIB

```
#define BOOST_PYTHON_STATIC_LIB 1
```

Definition at line [25](#) of file [markopy.cpp](#).

10.34 markopy.cpp

```
00001 /** @file markopy.cpp
00002 * @brief CPython wrapper for libmarkov utils.
00003 * @authors Ata Hakçıl, Celal Sahir Çetiner
00004 *
00005 * This file is a wrapper for libmarkov utilities, exposing:
00006 * - MarkovPasswords
00007 * - Import
00008 * - Export
00009 * - Train
00010 * - Generate
00011 * - ModelMatrix
00012 * - Import
00013 * - Export
00014 * - Train
00015 * - ConstructMatrix
00016 * - DumpJSON
00017 * - FastRandomWalk
00018 *
00019 * @copydoc Markov::API::MarkovPasswords
00020 * @copydoc Markov::API::ModelMatrix
00021 *
00022 */
00023
00024 #define BOOST_ALL_STATIC_LIB 1
00025 #define BOOST_PYTHON_STATIC_LIB 1
00026 #include <Python.h>
00027 #include <boost/python.hpp>
00028 #include <MarkovAPI/src/modelMatrix.h>
00029
00030
00031 using namespace boost::python;
00032
00033 /**
00034 * @brief CPython module for Markov::API objects
00035 */
00036 namespace Markov::Markopy{
00037     BOOST_PYTHON_MODULE(markopy)
00038 }
```

```

00039     bool (Markov::API::MarkovPasswords::*Import) (const char*) = &Markov::Model<char>::Import;
00040     bool (Markov::API::MarkovPasswords::*Export) (const char*) = &Markov::Model<char>::Export;
00041     class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
00042         .def(init<>())
00043         .def("Train", &Markov::API::MarkovPasswords::Train)
00044         .def("Generate", &Markov::API::MarkovPasswords::Generate)
00045         .def("Import", Import, "Import a model file.")
00046         .def("Export", Export, "Export a model to file.")
00047     ;
00048
00049     int (Markov::API::ModelMatrix::*FastRandomWalk)(unsigned long int, const char*, int, int, int,
00050     bool) = &Markov::API::ModelMatrix::FastRandomWalk;
00051     class_<Markov::API::ModelMatrix>("ModelMatrix", init<>())
00052
00053         .def(init<>())
00054         .def("Train", &Markov::API::ModelMatrix::Train)
00055         .def("Import", &Markov::API::ModelMatrix::Import, "Import a model file.")
00056         .def("Export", Export, "Export a model to file.")
00057         .def("ConstructMatrix", &Markov::API::ModelMatrix::ConstructMatrix)
00058         .def("DumpJSON", &Markov::API::ModelMatrix::DumpJSON)
00059         .def("FastRandomWalk", FastRandomWalk)
00060     ;
00061 };
00062 };

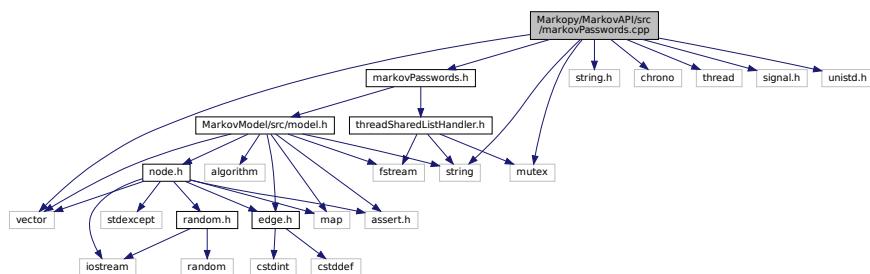
```

10.35 Markopy/MarkovAPI/src/markovPasswords.cpp File Reference

Wrapper for [Markov::Model](#) to use with char represented models.

```
#include "markovPasswords.h"
#include <string.h>
#include <chrono>
#include <thread>
#include <vector>
#include <mutex>
#include <string>
#include <signal.h>
#include <unistd.h>
```

Include dependency graph for markovPasswords.cpp:



Functions

- void [intHandler](#) (int dummy)

Variables

- static volatile int [keepRunning](#) = 1

10.35.1 Detailed Description

Wrapper for [Markov::Model](#) to use with char represented models.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay

This file contains the implementation for [Markov::API::MarkovPasswords](#) class.

[Markov::Model](#) with char represented nodes. Includes wrappers for [Markov::Model](#) and additional helper functions to handle file I/O

This class is an extension of [Markov::Model<char>](#), with higher level abstractions such as train and generate.

Definition in file [markovPasswords.cpp](#).

10.35.2 Function Documentation

10.35.2.1 intHandler()

```
void intHandler (
    int dummy )
```

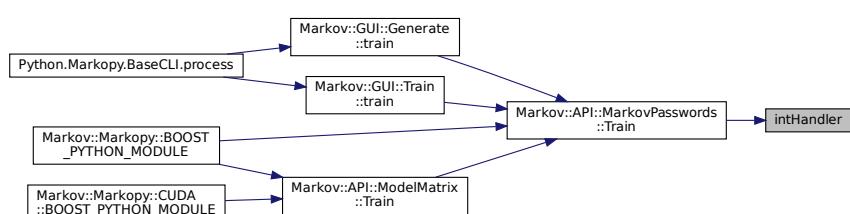
Definition at line 26 of file [markovPasswords.cpp](#).

```
00026     {
00027         std::cout << "Terminating.\n";
00028         //Sleep(5000);
00029         keepRunning = 0;
00030         exit(0);
00031 }
```

References [keepRunning](#).

Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the caller graph for this function:



10.35.3 Variable Documentation

10.35.3.1 keepRunning

```
volatile int keepRunning = 1 [static]
```

Definition at line 24 of file [markovPasswords.cpp](#).

Referenced by [intHandler\(\)](#), and [Markov::API::MarkovPasswords::TrainThread\(\)](#).

10.36 markovPasswords.cpp

```
00001 /** @file markovPasswords.cpp
00002 * @brief Wrapper for Markov::Model to use with char represented models.
00003 * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
00004 *
00005 * This file contains the implementation for Markov::API::MarkovPasswords class.
00006 *
00007 * @copydoc Markov::API::MarkovPasswords
00008 */
00009
00010 #include "markovPasswords.h"
00011 #include <string.h>
00012 #include <chrono>
00013 #include <thread>
```

```

00014 #include <vector>
00015 #include <mutex>
00016 #include <string>
00017 #include <signal.h>
00018 #ifdef _WIN32
00019 #include <Windows.h>
00020 #else
00021 #include <unistd.h>
00022 #endif
00023
00024 static volatile int keepRunning = 1;
00025
00026 void intHandler(int dummy) {
00027     std::cout << "Terminating.\n";
00028     //Sleep(5000);
00029     keepRunning = 0;
00030     exit(0);
00031 }
00032
00033
00034 Markov::API::MarkovPasswords::MarkovPasswords() : Markov::Model<char>() {
00035
00036
00037 }
00038
00039 Markov::API::MarkovPasswords::MarkovPasswords(const char* filename) {
00040
00041     std::ifstream* importFile;
00042
00043     this->Import(filename);
00044
00045     //std::ifstream* newFile(filename);
00046
00047     //importFile = newFile;
00048
00049 }
00050
00051 std::ifstream* Markov::API::MarkovPasswords::OpenDatasetFile(const char* filename) {
00052
00053     std::ifstream* datasetFile;
00054
00055     std::ifstream newFile(filename);
00056
00057     datasetFile = &newFile;
00058
00059     this->Import(datasetFile);
00060     return datasetFile;
00061 }
00062
00063
00064
00065 void Markov::API::MarkovPasswords::Train(const char* datasetFileName, char delimiter, int threads) {
00066     signal(SIGINT, intHandler);
00067     Markov::API::Concurrency::ThreadSharedListHandler listhandler(datasetFileName);
00068     auto start = std::chrono::high_resolution_clock::now();
00069
00070     std::vector<std::thread*> threadsV;
00071     for(int i=0;i<threads;i++) {
00072         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::TrainThread, this,
00073             &listhandler, delimiter));
00074     }
00075     for(int i=0;i<threads;i++) {
00076         threadsV[i]->join();
00077         delete threadsV[i];
00078     }
00079     auto finish = std::chrono::high_resolution_clock::now();
00080     std::chrono::duration<double> elapsed = finish - start;
00081     std::cout << "Elapsed time: " << elapsed.count() << " s\n";
00082
00083 }
00084
00085 void Markov::API::MarkovPasswords::TrainThread(Markov::API::Concurrency::ThreadSharedListHandler
00086     *listhandler, char delimiter) {
00087     char format_str[] = "%ld,%s";
00088     format_str[3]=delimiter;
00089     std::string line;
00090     while (listhandler->next(&line) && keepRunning) {
00091         long int oc;
00092         if (line.size() > 100) {
00093             line = line.substr(0, 100);
00094         }
00095         char* linebuf = new char[line.length()+5];
00096         sscanf_s(line.c_str(), "%ld,%s", &oc, linebuf, line.length()+5); //<== changed format_str to-
00097         "%ld,%s"
00098     }
00099 }
```

```

00098         sscanf(line.c_str(), format_str, &oc, linebuf);
00099 #endif
00100     this->AdjustEdge((const char*)linebuf, oc);
00101     delete linebuf;
00102 }
00103 }
00104
00105
00106 std::ofstream* Markov::API::MarkovPasswords::Save(const char* filename) {
00107     std::ofstream* exportFile;
00108
00109     std::ofstream newFile(filename);
00110
00111     exportFile = &newFile;
00112
00113     this->Export(exportFile);
00114     return exportFile;
00115 }
00116
00117
00118 void Markov::API::MarkovPasswords::Generate(unsigned long int n, const char* wordlistFileName, int
00119     minLen, int maxLen, int threads) {
00120     char res[100];
00121     std::fstream wordlist;
00122     wordlist.open(wordlistFileName);
00123     std::mutex mlock;
00124     int iterationsPerThread = n/threads;
00125     int iterationsCarryOver = n%threads;
00126     std::vector<std::thread*> threadsV;
00127     for(int i=0;i<threads;i++) {
00128         threadsV.push_back(new std::thread(&Markov::API::MarkovPasswords::GenerateThread, this,
00129             &mlock, iterationsPerThread, &wordlist, minLen, maxLen));
00130     }
00131     for(int i=0;i<threads;i++) {
00132         threadsV[i]->join();
00133         delete threadsV[i];
00134     }
00135
00136     this->GenerateThread(&mlock, iterationsCarryOver, &wordlist, minLen, maxLen);
00137
00138 }
00139
00140 void Markov::API::MarkovPasswords::GenerateThread(std::mutex *outputLock, unsigned long int n,
00141     std::fstream *wordlist, int minLen, int maxLen) {
00142     char* res = new char[maxLen+5];
00143     if(n==0) return;
00144
00145     Markov::Random::Marsaglia MarsagliaRandomEngine;
00146     for (int i = 0; i < n; i++) {
00147         this->RandomWalk(&MarsagliaRandomEngine, minLen, maxLen, res);
00148         outputLock->lock();
00149         *wordlist << res << "\n";
00150         outputLock->unlock();
00151     }
00152
00153 void Markov::API::MarkovPasswords::Buff(const char* str, double multiplier, bool bDontAdjustSelfLoops,
00154     bool bDontAdjustExtendedLoops) {
00155     std::string buffstr(str);
00156     std::map< char, Node< char > * > *nodes;
00157     std::map< char, Edge< char > * > *edges;
00158     nodes = this->Nodes();
00159     int i=0;
00160     for (auto const& [repr, node] : *nodes) {
00161         edges = node->Edges();
00162         for (auto const& [targetrepr, edge] : *edges) {
00163             if(buffstr.find(targetrepr)!= std::string::npos) {
00164                 if(bDontAdjustSelfLoops && repr==targetrepr) continue;
00165                 if(bDontAdjustExtendedLoops) {
00166                     if(buffstr.find(repr)!= std::string::npos) {
00167                         continue;
00168                     }
00169                     long int weight = edge->EdgeWeight();
00170                     weight = weight*multiplier;
00171                     edge->AdjustEdge(weight);
00172                 }
00173             }
00174         }
00175         i++;
00176     }
00177
00178     this->OptimizeEdgeOrder();
00179 }
```

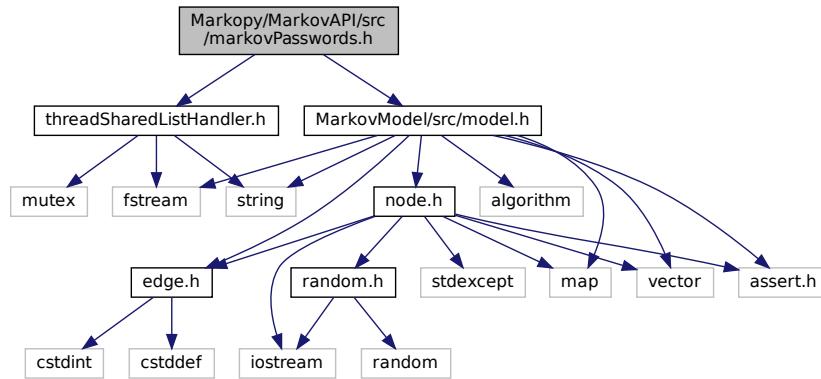
10.37 Markopy/MarkovAPI/src/markovPasswords.h File Reference

Wrapper for [Markov::Model](#) to use with char represented models.

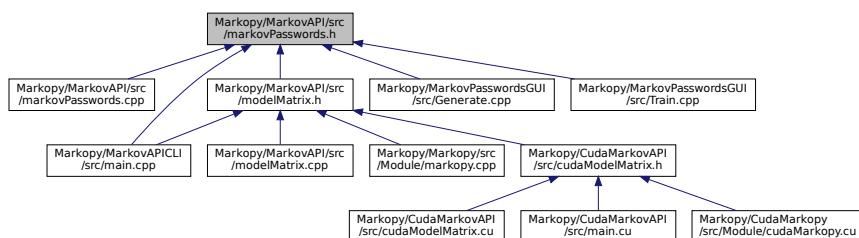
```
#include "threadSharedListHandler.h"
```

```
#include "MarkovModel/src/model.h"
```

Include dependency graph for markovPasswords.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::API::MarkovPasswords](#)
`Markov::Model` with char represented nodes.

Namespaces

- [Markov](#)
`Namespace for the markov-model related classes. Contains Model, Node and Edge classes.`
- [Markov::API](#)
`Namespace for the MarkovPasswords API.`

10.37.1 Detailed Description

Wrapper for [Markov::Model](#) to use with char represented models.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay

This file contains the declarations for [Markov::API::MarkovPasswords](#) class.

[Markov::Model](#) with char represented nodes. Includes wrappers for [Markov::Model](#) and additional helper functions to handle file I/O

This class is an extension of [Markov::Model<char>](#), with higher level abstractions such as train and generate.

Definition in file [markovPasswords.h](#).

10.38 markovPasswords.h

```

00001 /** @file markovPasswords.h
00002  * @brief Wrapper for Markov::Model to use with char represented models.
00003  * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
00004  *
00005  * This file contains the declarations for Markov::API::MarkovPasswords class.
00006  *
00007  * @copydoc Markov::API::MarkovPasswords
00008  */
00009
00010 #pragma once
00011 #include "threadSharedListHandler.h"
00012 #include "MarkovModel/src/model.h"
00013
00014
00015 /** @brief Namespace for the MarkovPasswords API
00016  */
00017 namespace Markov::API{
00018
00019     /** @brief Markov::Model with char represented nodes.
00020      *
00021      * Includes wrappers for Markov::Model and additional helper functions to handle file I/O
00022      *
00023      * This class is an extension of Markov::Model<char>, with higher level abstractions such as train
00024      and generate.
00025      *
00026     class MarkovPasswords : public Markov::Model<char>{
00027         public:
00028
00029             /** @brief Initialize the markov model from MarkovModel::Markov::Model.
00030              *
00031              * Parent constructor. Has no extra functionality.
00032              */
00033             MarkovPasswords();
00034
00035             /** @brief Initialize the markov model from MarkovModel::Markov::Model, with an import file.
00036              *
00037              * This function calls the Markov::Model::Import on the filename to construct the model.
00038              * Same thing as creating an empty model, and calling MarkovPasswords::Import on the
00039              filename.
00040              *
00041              * @param filename - Filename to import
00042              *
00043              * @b Example @b Use: Construction via filename
00044              * @code{.cpp}
00045              * MarkovPasswords mp("test.mdl");
00046              * @endcode
00047              */
00048             MarkovPasswords(const char* filename);
00049
00050             /** @brief Open dataset file and return the ifstream pointer
00051              * @param filename - Filename to open
00052              * @return ifstream* to the dataset file
00053              */
00054             std::ifstream* OpenDatasetFile(const char* filename);
00055
00056
00057             /** @brief Train the model with the dataset file.
00058              * @param datasetFileName - Ifstream* to the dataset. If null, use class member
00059              * @param delimiter - a character, same as the delimiter in dataset content
00060              * @param threads - number of OS threads to spawn
00061              *
00062              * @code{.cpp}
00063              * Markov::API::MarkovPasswords mp;
00064              * mp.Import("models/2gram.mdl");
00065              * mp.Train("password.corpus");
00066              * @endcode
00067              */
00068             void Train(const char* datasetFileName, char delimiter, int threads);

```

```

00069
00070
00071
00072     /** @brief Export model to file.
00073     * @param filename - Export filename.
00074     * @return std::ofstream* of the exported file.
00075     */
00076     std::ofstream* Save(const char* filename);
00077
00078     /** @brief Call Markov::Model::RandomWalk n times, and collect output.
00079     *
00080     * Generate from model and write results to a file.
00081     * a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5
00082     on average.
00083     *
00084     * @deprecated See Markov::API::MatrixModel::FastRandomWalk for more information.
00085     * @param n - Number of passwords to generate.
00086     * @param wordlistFileName - Filename to write to
00087     * @param minLen - Minimum password length to generate
00088     * @param maxLen - Maximum password length to generate
00089     * @param threads - number of OS threads to spawn
00090     */
00091     void Generate(unsigned long int n, const char* wordlistFileName, int minLen=6, int maxLen=12,
int threads=20);
00092
00093     /** @brief Buff expression of some characters in the model
00094     * @param str A string containing all the characters to be buffed
00095     * @param multiplier A constant value to buff the nodes with.
00096     * @param bDontAdjustSelfEdges Do not adjust weights if target node is same as source node
00097     * @param bDontAdjustExtendedLoops Do not adjust if both source and target nodes are in first
parameter
00098     */
00099     void Buff(const char* str, double multiplier, bool bDontAdjustSelfLoops=true, bool
bDontAdjustExtendedLoops=false);
00100
00101     private:
00102
00103     /** @brief A single thread invoked by the Train function.
00104     * @param listhandler - Listhandler class to read corpus from
00105     * @param delimiter - a character, same as the delimiter in dataset content
00106     *
00107     */
00108     void TrainThread(Markov::API::Concurrency::ThreadSharedListHandler *listhandler, char
delimiter);
00109
00110     /** @brief A single thread invoked by the Generate function.
00111     *
00112     * @b DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information.
This has been replaced with
00113     * a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5
on average.
00114     *
00115     * @param outputLock - shared mutex lock to lock during output operation. Prevents race
condition on write.
00116     * @param n number of lines to be generated by this thread
00117     * @param wordlist wordlistfile
00118     * @param minLen - Minimum password length to generate
00119     * @param maxLen - Maximum password length to generate
00120     *
00121     */
00122     void GenerateThread(std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int
minLen, int maxLen);
00123     std::ifstream* datasetFile; /** @brief Dataset file input of our system */
00124     std::ofstream* modelSavefile; /** @brief File to save model of our system */
00125     std::ofstream* outputFile; /** @brief Generated output file of our system */
00126 };
00127
00128
00129
00130 };

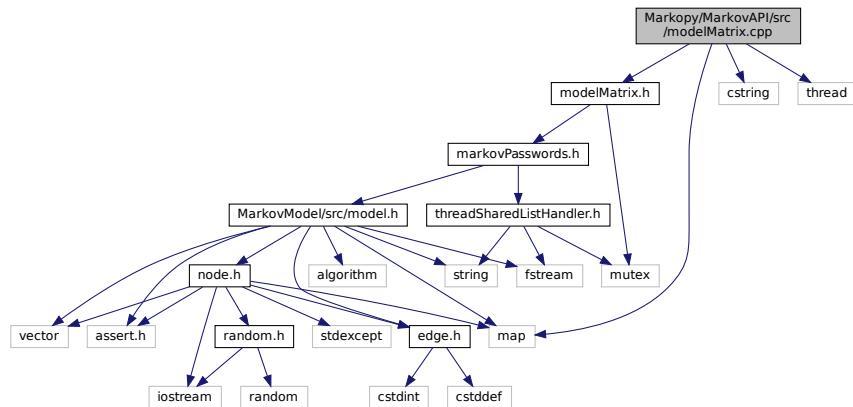
```

10.39 Markopy/MarkovAPI/src/modelMatrix.cpp File Reference

An extension of [Markov::API::MarkovPasswords](#).

```
#include "modelMatrix.h"
#include <map>
#include <cstring>
#include <thread>
```

Include dependency graph for modelMatrix.cpp:



10.39.1 Detailed Description

An extension of [Markov::API::MarkovPasswords](#).

Authors

Ata Hakçıl

This class shows superior performance compared to the traditional model at [Markov::API::MarkovPasswords](#). Class to flatten and reduce [Markov::Model](#) to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)

To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are synchronized and files are flushed every 50M operations.

Definition in file [modelMatrix.cpp](#).

10.40 modelMatrix.cpp

```

00001 /**
00002 * @file modelMatrix.cpp
00003 * @brief An extension of Markov::API::MarkovPasswords
00004 * @authors Ata Hakçıl
00005 * This class shows superior performance compared to the traditional model at
00006 * Markov::API::MarkovPasswords
00007 * @copydoc Markov::API::ModelMatrix
00008 */
00009
00010 #include "modelMatrix.h"
00011 #include <map>
00012 #include <cstring>
00013 #include <thread>
00014
00015 Markov::API::ModelMatrix::ModelMatrix() {
00016     this->ready = false;
00017 }
00018
00019 void Markov::API::ModelMatrix::Import(const char *filename) {
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
00024
00025 void Markov::API::ModelMatrix::Train(const char *datasetFileName, char delimiter, int threads) {
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName, delimiter, threads);
00028     this->ConstructMatrix();
00029 }
00030
00031 bool Markov::API::ModelMatrix::ConstructMatrix() {
00032     if(this->ready) return false;
  
```

```

00033     this->matrixSize = this->StarterNode()->edgesV.size() + 2;
00034
00035     this->matrixIndex = new char[this->matrixSize];
00036     this->totalEdgeWeights = new long int[this->matrixSize];
00037
00038     this->edgeMatrix = new char*[this->matrixSize];
00039     for(int i=0;i<this->matrixSize;i++){
00040         this->edgeMatrix[i] = new char[this->matrixSize];
00041     }
00042     this->valueMatrix = new long int*[this->matrixSize];
00043     for(int i=0;i<this->matrixSize;i++){
00044         this->valueMatrix[i] = new long int[this->matrixSize];
00045     }
00046     std::map< char, Node< char > * > *nodes;
00047     nodes = this->Nodes();
00048     int i=0;
00049     for (auto const& [repr, node] : *nodes){
00050         if(repr!=0) this->matrixIndex[i] = repr;
00051         else this->matrixIndex[i] = 199;
00052         this->totalEdgeWeights[i] = node->TotalEdgeWeights();
00053         for(int j=0;j<this->matrixSize;j++){
00054             char val = node->NodeValue();
00055             if(val < 0){
00056                 for(int k=0;k<this->matrixSize;k++){
00057                     this->valueMatrix[i][k] = 0;
00058                     this->edgeMatrix[i][k] = 255;
00059                 }
00060                 break;
00061             }
00062             else if(node->NodeValue() == 0 && j>(this->matrixSize-3)){
00063                 this->valueMatrix[i][j] = 0;
00064                 this->edgeMatrix[i][j] = 255;
00065             }else if(j==(this->matrixSize-1)) {
00066                 this->valueMatrix[i][j] = 0;
00067                 this->edgeMatrix[i][j] = 255;
00068             }else{
00069                 this->valueMatrix[i][j] = node->edgesV[j]->EdgeWeight ();
00070                 this->edgeMatrix[i][j] = node->edgesV[j]->RightNode ()->NodeValue ();
00071             }
00072         }
00073     }
00074     i++;
00075 }
00076     this->ready = true;
00077     return true;
00078 //this->DumpJSON();
00079 }
00080
00081 bool Markov::API::ModelMatrix::DeallocateMatrix(){
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085
00086     for(int i=0;i<this->matrixSize;i++){
00087         delete[] this->edgeMatrix[i];
00088     }
00089     delete[] this->edgeMatrix;
00090
00091     for(int i=0;i<this->matrixSize;i++){
00092         delete[] this->valueMatrix[i];
00093     }
00094     delete[] this->valueMatrix;
00095
00096     this->matrixSize = -1;
00097     this->ready = false;
00098     return true;
00099 }
00100
00101 void Markov::API::ModelMatrix::DumpJSON(){
00102
00103     std::cout << "(\n    \"index\":=\"";
00104     for(int i=0;i<this->matrixSize;i++){
00105         if(this->matrixIndex[i]=='"') std::cout << "\\\"\"";
00106         else if(this->matrixIndex[i]=='\\') std::cout << "\\\\\\"";
00107         else if(this->matrixIndex[i]==0) std::cout << "\\\\x00";
00108         else if(i==0) std::cout << "\\\\xff";
00109         else if(this->matrixIndex[i]=='\n') std::cout << "\\n";
00110         else std::cout << this->matrixIndex[i];
00111     }
00112     std::cout << "\\",\\n"
00113     << "    \"edgemap\": {\n";
00114
00115     for(int i=0;i<this->matrixSize;i++){
00116         if(this->matrixIndex[i]=='"') std::cout << "        \\\\"\\\"\\\"": "[";
00117         else if(this->matrixIndex[i]=='\\') std::cout << "        \\\\\\\\"\\\"\\\"": "[";
00118         else if(this->matrixIndex[i]==0) std::cout << "        \\\\"\\\"\\\"\\\"": "[";
00119

```

```

00120     else if(this->matrixIndex[i]<0) std::cout << " \\"\\x0ff\" : [";
00121     else std::cout << " \" " << this->matrixIndex[i] << "\": [";
00122     for(int j=0;j<this->matrixSize;j++) {
00123         if(this->edgeMatrix[i][j]==')') std::cout << "\\"\\\"\\\"";
00124         else if(this->edgeMatrix[i][j]=='\\') std::cout << "\\"\\\"\\\"\\\"";
00125         else if(this->edgeMatrix[i][j]==0) std::cout << "\\"\\\"\\\"\\x00\\\"";
00126         else if(this->edgeMatrix[i][j]<0) std::cout << "\\"\\\"\\\"\\xff\\\"";
00127         else if(this->matrixIndex[i]=='\\n') std::cout << "\\"\\\"\\n\\\"";
00128         else std::cout << "\\"\\\" << this->edgeMatrix[i][j] << "\\"\\\"";
00129         if(j!=this->matrixSize-1) std::cout << ", ";
00130     }
00131     std::cout << "],\\n";
00132 }
00133 std::cout << "},\\n";
00134
00135 std::cout << "\\" weightmap\\": {\\n";
00136 for(int i=0;i<this->matrixSize;i++) {
00137     if(this->matrixIndex[i]=='"') std::cout << " \\"\\\"\\\"\\\" : [";
00138     else if(this->matrixIndex[i]=='\\') std::cout << " \\"\\\"\\\"\\\" : [";
00139     else if(this->matrixIndex[i]==0) std::cout << " \\"\\\"\\\"\\x00\\\" : [";
00140     else if(this->matrixIndex[i]<0) std::cout << " \\"\\\"\\\"\\xff\\\" : [";
00141     else std::cout << " \\"\\\" << this->matrixIndex[i] << "\": [";
00142
00143     for(int j=0;j<this->matrixSize;j++) {
00144         std::cout << this->valueMatrix[i][j];
00145         if(j!=this->matrixSize-1) std::cout << ", ";
00146     }
00147     std::cout << "],\\n";
00148 }
00149 std::cout << " }\\n}\\n";
00150 }
00151
00152
00153 void Markov::API::ModelMatrix::FastRandomWalkThread(std::mutex *mlock, std::ofstream *wordlist,
00154     unsigned long int n, int minLen, int maxLen, int id, bool bFileIO){
00155     if(n==0) return;
00156
00157     Markov::Random::Marsaglia MarsagliaRandomEngine;
00158     char* e;
00159     char *res = new char[(maxLen+2)*n];
00160     int index = 0;
00161     char next;
00162     long int selection;
00163     char cur;
00164     long int bufferctr = 0;
00165     for (int i = 0; i < n; i++) {
00166         cur=199;
00167         len=0;
00168         while (true) {
00169             e = strchr(this->matrixIndex, cur);
00170             index = e - this->matrixIndex;
00171             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00172             for(int j=0;j<this->matrixSize;j++) {
00173                 selection -= this->valueMatrix[index][j];
00174                 if (selection < 0) {
00175                     next = this->edgeMatrix[index][j];
00176                     break;
00177                 }
00178             }
00179
00180             if (len >= maxLen) break;
00181             else if ((next < 0) && (len < minLen)) continue;
00182             else if (next < 0) break;
00183             cur = next;
00184             res[bufferctr + len++] = cur;
00185         }
00186         res[bufferctr + len++] = '\\n';
00187         bufferctr+=len;
00188     }
00189
00190     if(bFileIO) {
00191         mlock->lock();
00192         *wordlist << res;
00193         mlock->unlock();
00194    }else{
00195        mlock->lock();
00196        std::cout << res;
00197        mlock->unlock();
00198    }
00199    delete res;
00200
00201 }
00202
00203
00204 int Markov::API::ModelMatrix::FastRandomWalk(unsigned long int n, std::ofstream *wordlist, int minLen,
00205     int maxLen, int threads, bool bFileIO){

```

```

00205
00206     std::mutex mlock;
00207     if(n<=50000000ull) this->FastRandomWalkPartition(&mlock, wordlist, n, minLen, maxLen, bFileIO,
00208     threads);
00209     else{
00210         int numberOfPartitions = n/50000000ull;
00211         for(int i=0;i<numberOfPartitions;i++)
00212             this->FastRandomWalkPartition(&mlock, wordlist, 50000000ull, minLen, maxLen, bFileIO,
00213             threads);
00214     return 0;
00215 }
00216
00217 int Markov::API::ModelMatrix::FastRandomWalk(unsigned long int n, const char* wordlistFileName, int
00218     minLen, int maxLen, int threads, bool bFileIO){
00219     std::ofstream wordlist;
00220     if(bFileIO)
00221         wordlist.open(wordlistFileName);
00222     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);
00223     return 0;
00224 }
00225 void Markov::API::ModelMatrix::FastRandomWalkPartition(std::mutex *mlock, std::ofstream *wordlist,
00226     unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads){
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231
00232     int id = 0;
00233     for(int i=0;i<threads;i++) {
00234         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,
00235             mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));
00236         id++;
00237     }
00238     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,
00239         wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));
00240
00241     for(int i=0;i<threads;i++)
00242         threadsV[i]->join();
00243 }

```

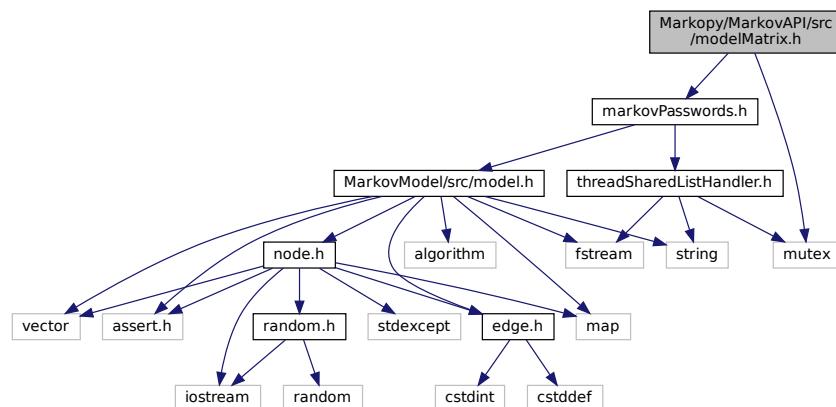
10.41 Markopy/MarkovAPI/src/modelMatrix.h File Reference

An extension of [Markov::API::MarkovPasswords](#).

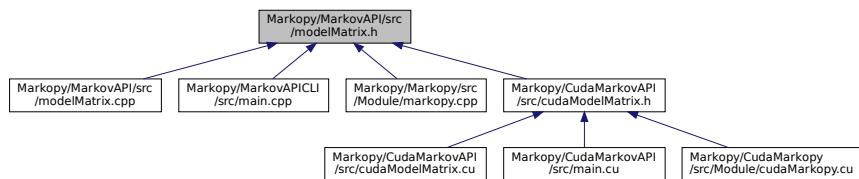
```
#include "markovPasswords.h"
```

```
#include <mutex>
```

Include dependency graph for modelMatrix.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::API::ModelMatrix](#)

Class to flatten and reduce [Markov::Model](#) to a Matrix.

Namespaces

- [Markov](#)

Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.

- [Markov::API](#)

Namespace for the [MarkovPasswords API](#).

10.41.1 Detailed Description

An extension of [Markov::API::MarkovPasswords](#).

Authors

Ata Hakçıl

This class shows superior performance compared to the traditional model at [Markov::API::MarkovPasswords](#). Class to flatten and reduce [Markov::Model](#) to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of $O(N)$ memory complexity ($O(1)$ memory space for slow mode).

To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are synchronized and files are flushed every 50M operations.

Definition in file [modelMatrix.h](#).

10.42 modelMatrix.h

```

00001 /**
00002 * @file modelMatrix.h
00003 * @brief An extension of Markov::API::MarkovPasswords
00004 * @authors Ata Hakçıl
00005 * This class shows superior performance compared to the traditional model at
00006 * Markov::API::MarkovPasswords
00007 * @copydoc Markov::API::ModelMatrix
00008 *
00009 */
00010
00011 #include "markovPasswords.h"
00012 #include <mutex>
00013
00014 namespace Markov::API{
00015
00016     /** @brief Class to flatten and reduce Markov::Model to a Matrix
00017     *
00018     * Matrix level operations can be used for Generation events, with a significant performance
00019     * optimization at the cost of  $O(N)$  memory complexity ( $O(1)$  memory space for slow mode)
00020     *
00021     * To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for
00022     * allocation. Threads are synchronized and files are flushed every 50M operations.
00023     */
00024 }
```

```

00022     */
00023     class ModelMatrix : public Markov::API::MarkovPasswords{
00024     public:
00025         ModelMatrix();
00026
00027         /** @brief Construct the related Matrix data for the model.
00028         *
00029         * This operation can be used after importing/training to allocate and populate the matrix
00030         * content.
00031         *
00032         * this will initialize:
00033         * char** edgeMatrix -> a 2D array of mapping left and right connections of each edge.
00034         * long int **valueMatrix -> a 2D array representing the edge weights.
00035         * int matrixSize -> Size of the matrix, aka total number of nodes.
00036         * char* matrixIndex -> order of nodes in the model
00037         * long int *totalEdgeWeights -> total edge weights of each Node.
00038         *
00039         * @returns True if constructed. False if already construced.
00040         */
00041     bool ConstructMatrix();
00042
00043     /** @brief Debug function to dump the model to a JSON file.
00044     *
00045     * Might not work 100%. Not meant for production use.
00046     */
00047     void DumpJSON();
00048
00049
00050     /** @brief Random walk on the Matrix-reduced Markov::Model
00051     *
00052     * This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are
00053     * partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
00054     *
00055     * If n>50M, threads are going to be synced, files are going to be flushed, and buffers will
00056     * be reallocated every 50M generations.
00057     *
00058     * This comes at a minor performance penalty.
00059     *
00060     * While it has the same functionality, this operation reduces
00061     * Markov::API::MarkovPasswords::Generate runtime by %96.5
00062     *
00063     * This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually
00064     * replace it.
00065     *
00066     * @param n - Number of passwords to generate.
00067     * @param wordlistFileName - Filename to write to
00068     * @param minLen - Minimum password length to generate
00069     * @param maxLen - Maximum password length to generate
00070     * @param threads - number of OS threads to spawn
00071     * @param bFileIO - If false, filename will be ignored and will output to stdout.
00072     *
00073     * @endcode{.cpp}
00074     * Markov::API::ModelMatrix mp;
00075     * mp.Import("models/finished.mdl");
00076     * mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
00077     * @endcode
00078     */
00079     int FastRandomWalk(unsigned long int n, const char* wordlistFileName, int minLen=6, int
00080                         maxLen=12, int threads=20, bool bFileIO=true);
00081
00082     /** @copydoc Markov::Model::Import(const char *filename)
00083     * Construct the matrix when done.
00084     */
00085     void Import(const char *filename);
00086
00087     /** @copydoc Markov::API::MarkovPasswords::Train(const char *datasetFileName, char delimiter,
00088     * int threads)
00089     * Construct the matrix when done.
00090     */
00091     protected:
00092     /** @brief Random walk on the Matrix-reduced Markov::Model
00093     *
00094     * This has an O(N) Memory complexity. To limit the maximum usage, requests with n>50M are
00095     * partitioned using Markov::API::ModelMatrix::FastRandomWalkPartition.
00096     *
00097     * If n>50M, threads are going to be synced, files are going to be flushed, and buffers will
00098     * be reallocated every 50M generations.
00099     *
00100     * This comes at a minor performance penalty.
00101     *
00102     * While it has the same functionality, this operation reduces

```

```

Markov::API::MarkovPasswords::Generate runtime by %96.5
00100      *
00101      * This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually
00102      replace it.
00103      *
00104      * @param n - Number of passwords to generate.
00105      * @param wordlistFileName - Filename to write to
00106      * @param minLen - Minimum password length to generate
00107      * @param maxLen - Maximum password length to generate
00108      * @param threads - number of OS threads to spawn
00109      * @param bFileIO - If false, filename will be ignored and will output to stdout.
00110      *
00111      * @endcode{.cpp}
00112      * Markov::API::ModelMatrix mp;
00113      * mp.Import("models/finished.mdl");
00114      * mp.FastRandomWalk(50000000, "./wordlist.txt", 6, 12, 25, true);
00115      * @endcode
00116      *
00117      */
00118      int FastRandomWalk(unsigned long int n, std::ofstream *wordlist, int minLen=6, int maxLen=12,
00119      int threads=20, bool bFileIO=true);
00120
00121      /** @brief A single partition of FastRandomWalk event
00122      *
00123      * Since FastRandomWalk has to allocate its output buffer before operation starts and writes
00124      data in chunks,
00125      * large n parameters would lead to huge memory allocations.
00126      * @b Without @b Partitioning:
00127      * - 50M results 12 characters max -> 550 Mb Memory allocation
00128      *
00129      * - 5B results 12 characters max -> 55 Gb Memory allocation
00130      *
00131      * - 50B results 12 characters max -> 550GB Memory allocation
00132      *
00133      * Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.
00134      *
00135      * @param mlock - mutex lock to distribute to child threads
00136      * @param wordlist - Reference to the wordlist file to write to
00137      * @param n - Number of passwords to generate.
00138      * @param wordlistFileName - Filename to write to
00139      * @param minLen - Minimum password length to generate
00140      * @param maxLen - Maximum password length to generate
00141      * @param threads - number of OS threads to spawn
00142      * @param bFileIO - If false, filename will be ignored and will output to stdout.
00143      *
00144      */
00145      void FastRandomWalkPartition(std::mutex *mlock, std::ofstream *wordlist, unsigned long int n,
00146      int minLen, int maxLen, bool bFileIO, int threads);
00147      /** @brief A single thread of a single partition of FastRandomWalk
00148      *
00149      * A FastRandomWalkPartition will initiate as many of this function as requested.
00150      *
00151      * This function contains the bulk of the generation algorithm.
00152      *
00153      * @param mlock - mutex lock to distribute to child threads
00154      * @param wordlist - Reference to the wordlist file to write to
00155      * @param n - Number of passwords to generate.
00156      * @param wordlistFileName - Filename to write to
00157      * @param minLen - Minimum password length to generate
00158      * @param maxLen - Maximum password length to generate
00159      * @param id - @b DEPRECATED Thread id - No longer used
00160      * @param bFileIO - If false, filename will be ignored and will output to stdout.
00161      *
00162      */
00163      */
00164      void FastRandomWalkThread(std::mutex *mlock, std::ofstream *wordlist, unsigned long int n, int
00165      minLen, int maxLen, int id, bool bFileIO);
00166      /** @brief Deallocate matrix and make it ready for re-construction
00167      *
00168      * @returns True if deallocated. False if matrix was not initialized
00169      */
00170      bool DeallocateMatrix();
00171
00172      /**
00173      * @brief 2-D Character array for the edge Matrix (The characters of Nodes)
00174      */
00175      char** edgeMatrix;
00176
00177      /**
00178      * @brief 2-d Integer array for the value Matrix (For the weights of Edges)
00179      */
00180      long int **valueMatrix;

```

```

00181      /**
00182       * @brief to hold Matrix size
00183     */
00184     int matrixSize;
00185
00186     /**
00187      * @brief to hold the Matrix index (To hold the orders of 2-D arrays')
00188     */
00189     char* matrixIndex;
00190
00191     /**
00192      * @brief Array of the Total Edge Weights
00193     */
00194     long int *totalEdgeWeights;
00195
00196     /**
00197      * @brief True when matrix is constructed. False if not.
00198     */
00199     bool ready;
00200 };
00201
00202
00203
00204
00205 };

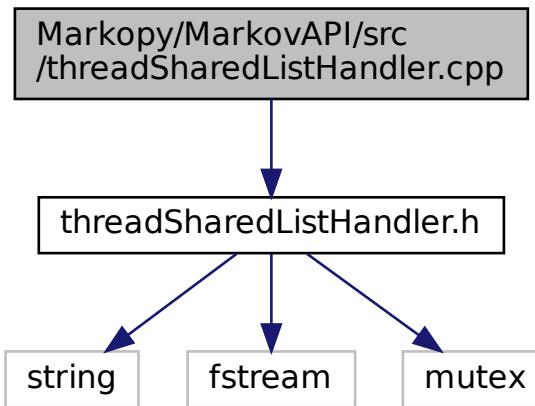
```

10.43 Markopy/MarkovAPI/src/threadSharedListHandler.cpp File Reference

Thread-safe wrapper for std::ifstream.

```
#include "threadSharedListHandler.h"
```

Include dependency graph for threadSharedListHandler.cpp:



10.43.1 Detailed Description

Thread-safe wrapper for std::ifstream.

Authors

Ata Hakçıl

Simple class for managing shared access to file. This class maintains the handover of each line from a file to multiple threads.

When two different threads try to read from the same file while reading a line isn't completed, it can have unexpected results. Line might be split, or might be read twice. This class locks the read action on the list until a line is completed, and then proceeds with the handover.

Definition in file [threadSharedListHandler.cpp](#).

10.44 threadSharedListHandler.cpp

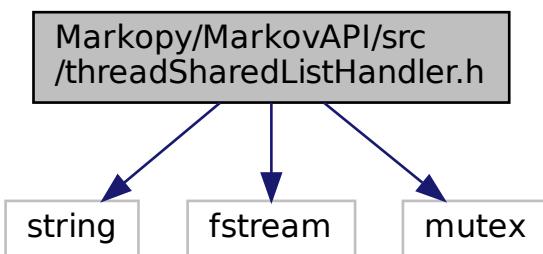
```
00001 /** @file threadSharedListHandler.cpp
00002  * @brief Thread-safe wrapper for std::ifstream
00003  * @authors Ata Hakçıl
00004  *
00005  * @copydoc Markov::API::Concurrency::ThreadSharedListHandler
00006  *
00007  */
00008
00009 #include "threadSharedListHandler.h"
00010
00011
00012 Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler(const char* filename) {
00013     this->listfile;
00014     this->listfile.open(filename, std::ios_base::binary);
00015 }
00016
00017
00018 bool Markov::API::Concurrency::ThreadSharedListHandler::next(std::string* line) {
00019     bool res = false;
00020     this->mlock.lock();
00021     res = (std::getline(this->listfile,*line,'\\n'))? true : false;
00022     this->mlock.unlock();
00023
00024     return res;
00025 }
```

10.45 Markopy/MarkovAPI/src/threadSharedListHandler.h File Reference

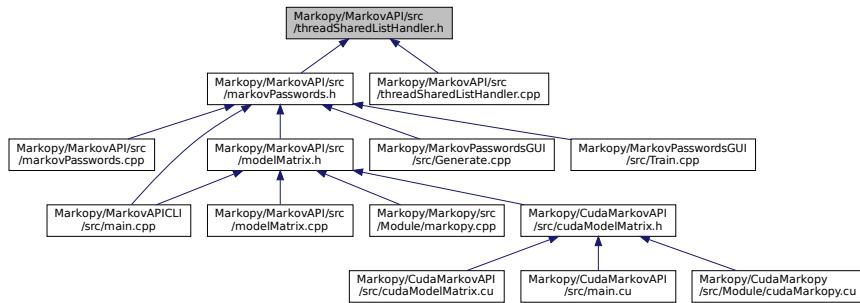
Thread-safe wrapper for std::ifstream.

```
#include <string>
#include <fstream>
#include <mutex>
```

Include dependency graph for `threadSharedListHandler.h`:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::API::Concurrency::ThreadSharedListHandler](#)
Simple class for managing shared access to file.

Namespaces

- [Markov](#)
Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.
- [Markov::API](#)
Namespace for the [MarkovPasswords API](#).
- [Markov::API::Concurrency](#)
Namespace for [Concurrency](#) related classes.

10.45.1 Detailed Description

Thread-safe wrapper for std::ifstream.

Authors

Ata Hakçıl

Simple class for managing shared access to file. This class maintains the handover of each line from a file to multiple threads.

When two different threads try to read from the same file while reading a line isn't completed, it can have unexpected results. Line might be split, or might be read twice. This class locks the read action on the list until a line is completed, and then proceeds with the handover.

Definition in file [threadSharedListHandler.h](#).

10.46 threadSharedListHandler.h

```

00001 /**
00002 * @file threadSharedListHandler.h
00003 * @brief Thread-safe wrapper for std::ifstream
00004 * @authors Ata Hakçıl
00005 * @copydoc Markov::API::Concurrency::ThreadSharedListHandler
00006 */
00007
00008 #include <string>
00009 #include <fstream>
00010 #include <mutex>
00011
00012 /** @brief Namespace for Concurrency related classes
00013 */
00014 namespace Markov::API::Concurrency{
00015
00016 /** @brief Simple class for managing shared access to file
00017 */

```

```

00018 * This class maintains the handover of each line from a file to multiple threads.
00019 *
00020 * When two different threads try to read from the same file while reading a line isn't completed, it
00021 * can have unexpected results.
00022 * Line might be split, or might be read twice.
00023 * This class locks the read action on the list until a line is completed, and then proceeds with the
00024 * handover.
00025 */
00026 class ThreadSharedListHandler{
00027 public:
00028     /** @brief Construct the Thread Handler with a filename
00029     *
00030     * Simply open the file, and initialize the locks.
00031     *
00032     * @b Example @b Use: Simple file read
00033     * @code{.cpp}
00034     * ThreadSharedListHandler listhandler("test.txt");
00035     * std::string line;
00036     * std::cout << listhandler->next(&line) << "\n";
00037     * @endcode
00038     * @b Example @b Use: Example use case from MarkovPasswords showing multithreaded access
00039     * @code{.cpp}
00040     * void MarkovPasswords::Train(const char* datasetFileName, char delimiter, int threads) {
00041     *     ThreadSharedListHandler listhandler(datasetFileName);
00042     *     auto start = std::chrono::high_resolution_clock::now();
00043     *
00044     *     std::vector<std::thread*> threadsV;
00045     *     for(int i=0;i<threads;i++) {
00046         *         threadsV.push_back(new std::thread(&MarkovPasswords::TrainThread, this, &listhandler,
00047         * datasetFileName, delimiter));
00048         *
00049         *         for(int i=0;i<threads;i++) {
00050         *             threadsV[i]->join();
00051         *             delete threadsV[i];
00052         *         }
00053         *         auto finish = std::chrono::high_resolution_clock::now();
00054         *         std::chrono::duration<double> elapsed = finish - start;
00055         *         std::cout << "Elapsed time: " << elapsed.count() << " s\n";
00056         *
00057     *     }
00058     *
00059     *     void MarkovPasswords::TrainThread(ThreadSharedListHandler *listhandler, const char*
00060     * datasetFileName, char delimiter){
00061     *         char format_str[] ="%ld,%s";
00062     *         format_str[2]=delimiter;
00063     *         std::string line;
00064     *         while (listhandler->next(&line)) {
00065     *             long int oc;
00066     *             if (line.size() > 100) {
00067     *                 line = line.substr(0, 100);
00068     *             }
00069     *             char* linebuf = new char[line.length()+5];
00070     *             sscanf_s(line.c_str(), format_str, &oc, linebuf, line.length()+5);
00071     *             this->AdjustEdge((const char*)linebuf, oc);
00072     *             delete linebuf;
00073     *         }
00074     *     @endcode
00075     *
00076     * @param filename Filename for the file to manage.
00077 */
00078 ThreadSharedListHandler(const char* filename);
00079
00080     /** @brief Read the next line from the file.
00081     *
00082     * This action will be blocked until another thread (if any) completes the read operation on the
00083     * file.
00084     *
00085     * @b Example @b Use: Simple file read
00086     * @code{.cpp}
00087     * ThreadSharedListHandler listhandler("test.txt");
00088     * std::string line;
00089     * std::cout << listhandler->next(&line) << "\n";
00090     * @endcode
00091     */
00092     bool next(std::string* line);
00093
00094 private:
00095     std::ifstream listfile;
00096     std::mutex mlock;
00097 };
00098
00099 };

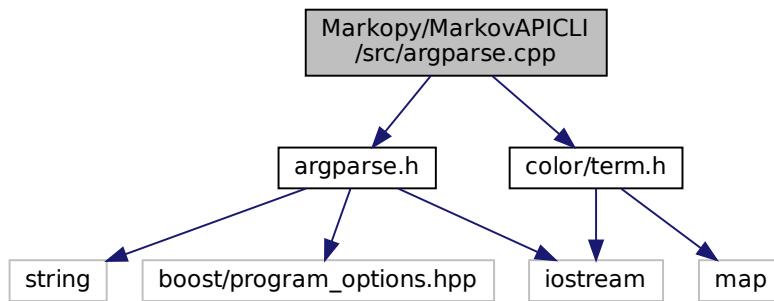
```

10.47 Markopy/MarkovAPICLI/src/argparse.cpp File Reference

Arguement handler class for native CPP cli.

```
#include "argparse.h"
#include "color/term.h"
```

Include dependency graph for argparse.cpp:



10.47.1 Detailed Description

Arguement handler class for native CPP cli.

Authors

Celal Sahir Çetiner

Parse command line arguements.

Definition in file [argparse.cpp](#).

10.48 argparse.cpp

```

00001 /**
00002 * @file argparse.cpp
00003 * @brief Arguement handler class for native CPP cli
00004 * @authors Celal Sahir Çetiner
00005 * @copydoc Markov::API::CLI::Argparse
00006 */
00007
00008 #include "argparse.h"
00009 #include "color/term.h"
00010
00011 Markov::API::CLI::ProgramOptions* Markov::API::CLI::Argparse::parse(int argc, char** argv) { return 0;
}
00012
00013
00014
00015 void Markov::API::CLI::Argparse::help() {
00016     std::cout <
00017         "Markov Passwords - Help\n"
00018         "Options:\n"
00019         "\n"
00020         "    -of --outputfilename\n"
00021         "        Filename to output the generation results\n"
00022         "    -ef --exportfilename\n"
00023         "        filename to export built model to\n"
00024         "    -if --importfilename\n"
00025         "        filename to import model from\n"
00026         "    -n (generate count)\n"
00027         "        Number of lines to generate\n"
00028         "\n"
00029         "Usage: \n"
00030         "    markov.exe -if empty_model.mdl -ef model.mdl\n"
00031         "        import empty_model.mdl and train it with data from stdin. When done, output the model to
model.mdl\n"
00032         "\n"

```

```

00033     "    markov.exe -if empty_model.mdl -n 15000 -of wordlist.txt\n"
00034     "        import empty_model.mdl and generate 15000 words to wordlist.txt\n"
00035
00036     << std::endl;
00037 }

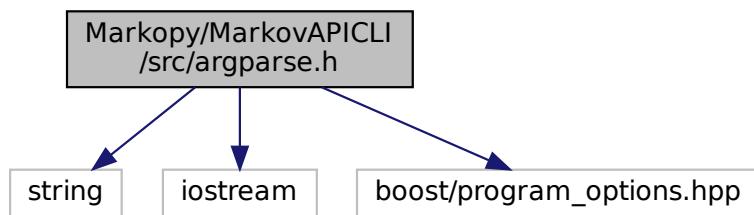
```

10.49 Markopy/MarkovAPICLI/src/argparse.h File Reference

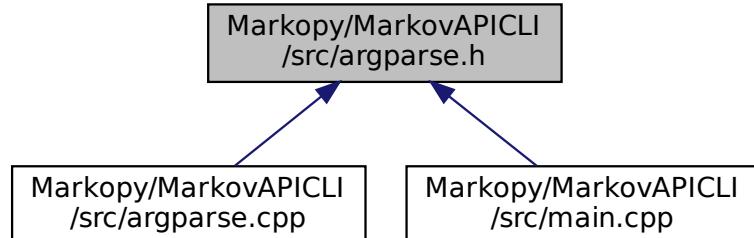
Arguement handler class for native CPP cli.

```
#include <string>
#include <iostream>
#include <boost/program_options.hpp>
```

Include dependency graph for argparse.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct [Markov::API::CLI::__programOptions](#)
Structure to hold parsed cli arguements.
- class [Markov::API::CLI::Argparse](#)
Parse command line arguements.

Namespaces

- [Markov](#)
Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- [Markov::API](#)

Namespace for the [MarkovPasswords API](#).

- [Markov::API::CLI](#)

Structure to hold parsed cli arguements.

Macros

- `#define BOOST_ALL_STATIC_LIB 1`
- `#define BOOST_PROGRAM_OPTIONS_STATIC_LIB 1`

Typedefs

- `typedef struct Markov::API::CLI::_programOptions Markov::API::CLI::ProgramOptions`

Structure to hold parsed cli arguements.

10.49.1 Detailed Description

Arguement handler class for native CPP cli.

Authors

Celal Sahir Çetiner

Definition in file [argparse.h](#).

10.49.2 Macro Definition Documentation

10.49.2.1 BOOST_ALL_STATIC_LIB

`#define BOOST_ALL_STATIC_LIB 1`

Definition at line 11 of file [argparse.h](#).

10.49.2.2 BOOST_PROGRAM_OPTIONS_STATIC_LIB

`#define BOOST_PROGRAM_OPTIONS_STATIC_LIB 1`

Definition at line 12 of file [argparse.h](#).

10.50 argparse.h

```
00001 /** @file argparse.h
00002 * @brief Arguement handler class for native CPP cli
00003 * @authors Celal Sahir Çetiner
00004 *
00005 * @copydoc Markov::API::CLI::Argparse:
00006 */
00007
00008 #include<string>
00009 #include<iostream>
00010
00011 #define BOOST_ALL_STATIC_LIB 1
00012 #define BOOST_PROGRAM_OPTIONS_STATIC_LIB 1
00013
00014 #include <boost/program_options.hpp>
00015
00016 /** @brief Structure to hold parsed cli arguements.
00017 */
00018 namespace opt = boost::program_options;
00019
00020 /**
00021      @brief Namespace for the CLI objects
00022 */
00023 namespace Markov::API::CLI{
00024
00025     /** @brief Structure to hold parsed cli arguements. */
00026     typedef struct _programOptions {
```

```

00027      /**
00028          @brief Import flag to validate import
00029      */
00030      bool bImport;
00031
00032      /**
00033          @brief Export flag to validate export
00034      */
00035      bool bExport;
00036
00037      /**
00038          @brief Failure flag to validate succesfull running
00039      */
00040      bool bFailure;
00041
00042      /**
00043          @brief Seperator character to use with training data. (character between occurence and
00044          value)"
00045      */
00046      char separator;
00047
00048      /**
00049          @brief Import name of our model
00050      */
00051      std::string importname;
00052
00053      /**
00054          @brief Import name of our given wordlist
00055      */
00056      std::string exportname;
00057
00058      /**
00059          @brief Import name of our given wordlist
00060      */
00061      std::string wordlistname;
00062
00063      /**
00064          @brief Output name of our generated password list
00065      */
00066      std::string outputfilename;
00067
00068      /**
00069          @brief The name of the given dataset
00070      */
00071      std::string datasetname;
00072
00073      /**
00074          @brief Number of passwords to be generated
00075      */
00076      int generateN;
00077  } ProgramOptions;
00078
00079
00080  /** @brief Parse command line arguements
00081  */
00082  class Argparse {
00083  public:
00084
00085      Argparse();
00086
00087      /** @brief Parse command line arguements.
00088      *
00089      * Parses command line arguements to populate ProgramOptions structure.
00090      *
00091      * @param argc Number of command line arguements
00092      * @param argv Array of command line parameters
00093      */
00094      Argparse(int argc, char** argv) {
00095
00096          /*bool bImp;
00097          bool bExp;
00098          bool bFail;
00099          char spt;
00100          std::string imports;
00101          std::string exports;
00102          std::string outputs;
00103          std::string datasets;
00104          int generateN;
00105          */
00106          opt::options_description desc("Options");
00107
00108
00109          desc.add_options()
00110              ("generate", "Generate strings with given parameters")
00111              ("train", "Train model with given parameters")
00112              ("combine", "Combine")

```

```

00113         ("import", opt::value<std::string>(), "Import model file")
00114         ("output", opt::value<std::string>(), "Output model file. This model will be exported
when done. Will be ignored for generation mode")
00115         ("dataset", opt::value<std::string>(), "Dataset file to read input from training. Will
be ignored for generation mode")
00116         ("seperator", opt::value<char>(), "Separator character to use with training data.
(character between occurrence and value)")
00117         ("wordlist", opt::value<std::string>(), "Wordlist file path to export generation
results to. Will be ignored for training mode")
00118         ("count", opt::value<int>(), "Number of lines to generate. Ignored in training mode")
00119         ("verbosity", "Output verbosity")
00120         ("help", "Option definitions");
00121
00122         opt::variables_map vm;
00123
00124         opt::store(opt::parse_command_line(argc, argv, desc), vm);
00125
00126         opt::notify(vm);
00127
00128         //std::cout << desc << std::endl;
00129         if (vm.count("help")) {
00130             std::cout << desc << std::endl;
00131         }
00132
00133         if (vm.count("output") == 0) this->po.outputfilename = "NULL";
00134         else if (vm.count("output") == 1) {
00135             this->po.outputfilename = vm["output"].as<std::string>();
00136             this->po.bExport = true;
00137         }
00138         else {
00139             this->po.bFailure = true;
00140             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00141             std::cout << desc << std::endl;
00142         }
00143
00144
00145         if (vm.count("dataset") == 0) this->po.datasetname = "NULL";
00146         else if (vm.count("dataset") == 1) {
00147             this->po.datasetname = vm["dataset"].as<std::string>();
00148         }
00149         else {
00150             this->po.bFailure = true;
00151             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00152             std::cout << desc << std::endl;
00153         }
00154
00155
00156         if (vm.count("wordlist") == 0) this->po.wordlistname = "NULL";
00157         else if (vm.count("wordlist") == 1) {
00158             this->po.wordlistname = vm["wordlist"].as<std::string>();
00159         }
00160         else {
00161             this->po.bFailure = true;
00162             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00163             std::cout << desc << std::endl;
00164         }
00165
00166         if (vm.count("import") == 0) this->po.importname = "NULL";
00167         else if (vm.count("import") == 1) {
00168             this->po.importname = vm["import"].as<std::string>();
00169             this->po.bImport = true;
00170         }
00171         else {
00172             this->po.bFailure = true;
00173             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00174             std::cout << desc << std::endl;
00175         }
00176
00177
00178         if (vm.count("count") == 0) this->po.generateN = 0;
00179         else if (vm.count("count") == 1) {
00180             this->po.generateN = vm["count"].as<int>();
00181         }
00182         else {
00183             this->po.bFailure = true;
00184             std::cout << "UNIDENTIFIED INPUT" << std::endl;
00185             std::cout << desc << std::endl;
00186         }
00187
00188         /*std::cout << vm["output"].as<std::string>() << std::endl;
00189         std::cout << vm["dataset"].as<std::string>() << std::endl;
00190         std::cout << vm["wordlist"].as<std::string>() << std::endl;
00191         std::cout << vm["output"].as<std::string>() << std::endl;
00192         std::cout << vm["count"].as<int>() << std::endl; */
00193
00194         //else if (vm.count("train")) std::cout << "train oldu" << std::endl;

```

```

00196      }
00197
00198      /** @brief Getter for command line options
00199      *
00200      * Getter for ProgramOptions populated by the arguement parser
00201      * @returns ProgramOptions structure.
00202      */
00203      Markov::API::CLI::ProgramOptions getProgramOptions(void) {
00204          return this->po;
00205      }
00206
00207      /** @brief Initialize program options structure.
00208      *
00209      * @param i boolean, true if import operation is flagged
00210      * @param e boolean, true if export operation is flagged
00211      * @param bf boolean, true if there is something wrong with the command line parameters
00212      * @param s seperator character for the import function
00213      * @param iName import filename
00214      * @param exName export filename
00215      * @param oName output filename
00216      * @param dName corpus filename
00217      * @param n number of passwords to be generated
00218      *
00219      */
00220      void setProgramOptions(bool i, bool e, bool bf, char s, std::string iName, std::string exName,
00221      std::string oName, std::string dName, int n) {
00222          this->po.bImport = i;
00223          this->po.bExport = e;
00224          this->po.seperator = s;
00225          this->po.bFailure = bf;
00226          this->po.generateN = n;
00227          this->po.importname = iName;
00228          this->po.exportname = exName;
00229          this->po.outputfilename = oName;
00230          this->po.datasetname = dName;
00231
00232          /*strcpy_s(this->po.importname,256,iName);
00233          strcpy_s(this->po.exportname,256,exName);
00234          strcpy_s(this->po.outputfilename,256,oName);
00235          strcpy_s(this->po.datasetname,256,dName);*/
00236      }
00237
00238      /** @brief parse cli commands and return
00239      * @param argc - Program arguement count
00240      * @param argv - Program arguement values array
00241      * @return ProgramOptions structure.
00242      */
00243      static Markov::API::CLI::ProgramOptions* parse(int argc, char** argv);
00244
00245
00246      /** @brief Print help string.
00247      */
00248      static void help();
00249
00250  private:
00251      /**
00252      * @brief ProgramOptions structure object
00253      */
00254
00255      Markov::API::CLI::ProgramOptions po;
00256  };
00257
00258 };

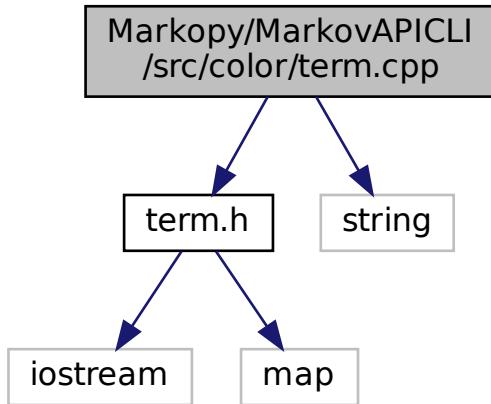
```

10.51 Markopy/MarkovAPICLI/src/color/term.cpp File Reference

Terminal handler for pretty stuff like colors.

```
#include "term.h"
#include <string>
```

Include dependency graph for term.cpp:



Functions

- std::ostream & [operator<<](#) (std::ostream &os, const Terminal::color &c)

10.51.1 Detailed Description

Terminal handler for pretty stuff like colors.

Authors

Ata Hakçıl

pretty colors for Terminal. Windows Only.

Definition in file [term.cpp](#).

10.51.2 Function Documentation

10.51.2.1 operator<<()

```
std::ostream& operator<< (
    std::ostream & os,
    const Terminal::color & c )
```

Definition at line 66 of file [term.cpp](#).

```
00066
00067     char buf[6];
00068     sprintf(buf,"%d",Terminal::colormap.find(c)->second);
00069     os << "\e[;" << buf << "m";
00070     return os;
00071 }
```

References [Markov::API::CLI::Terminal::colormap](#).

10.52 term.cpp

```
00001 /** @file term.cpp
00002 * @brief Terminal handler for pretty stuff like colors
00003 * @authors Ata Hakçıl
00004 *
```

```

00005  * @copydoc Markov::API::CLI::Terminal
00006  */
00007
00008 #include "term.h"
00009 #include <string>
00010
00011 using namespace Markov::API::CLI;
00012
00013 //Windows text processing is different from unix systems, so use windows header and text attributes
00014 #ifdef _WIN32
00015
00016 HANDLE Terminal::_stdout;
00017 HANDLE Terminal::_stderr;
00018
00019 std::map<Terminal::color, DWORD> Terminal::colormap = {
00020     {Terminal::color::BLACK, 0},
00021     {Terminal::color::BLUE, 1},
00022     {Terminal::color::GREEN, 2},
00023     {Terminal::color::CYAN, 3},
00024     {Terminal::color::RED, 4},
00025     {Terminal::color::MAGENTA, 5},
00026     {Terminal::color::BROWN, 6},
00027     {Terminal::color::LIGHTGRAY, 7},
00028     {Terminal::color::DARKGRAY, 8},
00029     {Terminal::color::YELLOW, 14},
00030     {Terminal::color::WHITE, 15},
00031     {Terminal::color::RESET, 15},
00032 };
00033
00034
00035 Terminal::Terminal() {
00036     Terminal::_stdout = GetStdHandle(STD_OUTPUT_HANDLE);
00037     Terminal::_stderr = GetStdHandle(STD_ERROR_HANDLE);
00038 }
00039
00040 std::ostream& operator<<(std::ostream& os, const Terminal::color& c) {
00041     SetConsoleTextAttribute(Terminal::_stdout, Terminal::colormap.find(c)->second);
00042     return os;
00043 }
00044
00045 #else
00046
00047 std::map<Terminal::color, int> Terminal::colormap = {
00048     {Terminal::color::BLACK, 30},
00049     {Terminal::color::BLUE, 34},
00050     {Terminal::color::GREEN, 32},
00051     {Terminal::color::CYAN, 36},
00052     {Terminal::color::RED, 31},
00053     {Terminal::color::MAGENTA, 35},
00054     {Terminal::color::BROWN, 0},
00055     {Terminal::color::LIGHTGRAY, 0},
00056     {Terminal::color::DARKGRAY, 0},
00057     {Terminal::color::YELLOW, 33},
00058     {Terminal::color::WHITE, 37},
00059     {Terminal::color::RESET, 0},
00060 };
00061
00062 Terminal::Terminal() {
00063     /*this->;*/
00064 }
00065
00066 std::ostream& operator<<(std::ostream& os, const Terminal::color& c) {
00067     char buf[6];
00068     sprintf(buf, "%d", Terminal::colormap.find(c)->second);
00069     os << "\e[1;" << buf << "m";
00070     return os;
00071 }
00072
00073
00074
00075
00076 #endif

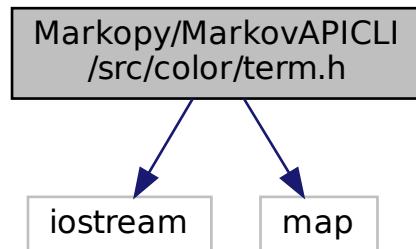
```

10.53 Markopy/MarkovAPICLI/src/color/term.h File Reference

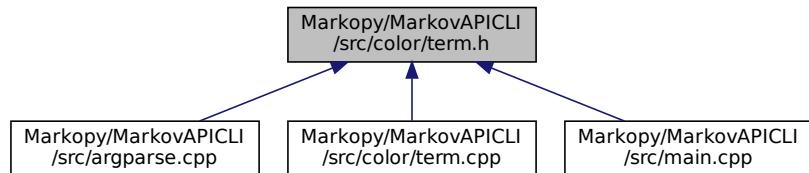
Terminal handler for pretty stuff like colors.

```
#include <iostream>
#include <map>
```

Include dependency graph for term.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::API::CLI::Terminal](#)
pretty colors for Terminal. Windows Only.

Namespaces

- [Markov](#)
Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- [Markov::API](#)
Namespace for the MarkovPasswords API.
- [Markov::API::CLI](#)
Structure to hold parsed cli arguements.

Macros

- `#define TERM_FAIL "[" << Markov::API::CLI::Terminal::color::RED << "+" << Markov::API::CLI::Terminal::color::RESET << "] "`
- `#define TERM_INFO "[" << Markov::API::CLI::Terminal::color::BLUE << "+" << Markov::API::CLI::Terminal::color::RESET << "] "`
- `#define TERM_WARN "[" << Markov::API::CLI::Terminal::color::YELLOW << "+" << Markov::API::CLI::Terminal::color::RESET << "] "`
- `#define TERM_SUCC "[" << Markov::API::CLI::Terminal::color::GREEN << "+" << Markov::API::CLI::Terminal::color::RESET << "] "`

Functions

- std::ostream & [Markov::API::CLI::operator<<](#) (std::ostream &os, const Markov::API::CLI::Terminal::color &c)

10.53.1 Detailed Description

Terminal handler for pretty stuff like colors.

Authors

Ata Hakçıl

pretty colors for Terminal. Windows Only.

Definition in file [term.h](#).

10.53.2 Macro Definition Documentation

10.53.2.1 TERM_FAIL

```
#define TERM_FAIL "[" << Markov::API::CLI::Terminal::color::RED << "+" << Markov::API::CLI::Terminal::color
<< "] "
```

Definition at line 17 of file [term.h](#).

10.53.2.2 TERM_INFO

```
#define TERM_INFO "[" << Markov::API::CLI::Terminal::color::BLUE << "+" << Markov::API::CLI::Terminal::color
<< "] "
```

Definition at line 18 of file [term.h](#).

10.53.2.3 TERM_SUCC

```
#define TERM_SUCC "[" << Markov::API::CLI::Terminal::color::GREEN << "+" << Markov::API::CLI::Terminal::color
<< "] "
```

Definition at line 20 of file [term.h](#).

10.53.2.4 TERM_WARN

```
#define TERM_WARN "[" << Markov::API::CLI::Terminal::color::YELLOW << "+" << Markov::API::CLI::Terminal::color
<< "] "
```

Definition at line 19 of file [term.h](#).

10.54 term.h

```
00001 /* @file term.h
00002 * @brief Terminal handler for pretty stuff like colors
00003 * @authors Ata Hakçıl
00004 *
00005 * @copydoc Markov::API::CLI::Terminal
00006 */
00007
00008 #pragma once
00009
00010 #ifdef _WIN32
00011 #include <Windows.h>
00012 #endif
00013
00014 #include <iostream>
00015 #include <map>
00016
00017 #define TERM_FAIL "[" << Markov::API::CLI::Terminal::color::RED << "+" <<
Markov::API::CLI::Terminal::color::RESET << "] "
```

```

00018 #define TERM_INFO "[" « Markov::API::CLI::Terminal::color::BLUE « "+" «
Markov::API::CLI::Terminal::color::RESET « "] "
00019 #define TERM_WARN "[" « Markov::API::CLI::Terminal::color::YELLOW « "+" «
Markov::API::CLI::Terminal::color::RESET « "] "
00020 #define TERM_SUCC "[" « Markov::API::CLI::Terminal::color::GREEN « "+" «
Markov::API::CLI::Terminal::color::RESET « "] "
00021
00022 namespace Markov::API::CLI{
00023     /** @brief pretty colors for Terminal. Windows Only.
00024     */
00025     class Terminal {
00026     public:
00027         /** Default constructor.
00028         * Get references to stdout and stderr handles.
00029         */
00030         Terminal();
00031
00032         enum color { RESET, BLACK, RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE, LIGHTGRAY,
DARKGRAY, BROWN };
00033         #ifdef _WIN32
00034             static HANDLE _stdout;
00035             static HANDLE _stderr;
00036             static std::map<Markov::API::CLI::Terminal::color, DWORD> colormap;
00037         #else
00038             static std::map<Markov::API::CLI::Terminal::color, int> colormap;
00039         #endif
00040
00041
00042
00043
00044         static std::ostream endl;
00045
00046
00047     };
00048
00049     /** overload for std::cout.
00050     */
00051     std::ostream& operator<<(std::ostream& os, const Markov::API::CLI::Terminal::color& c);
00052
00053 }

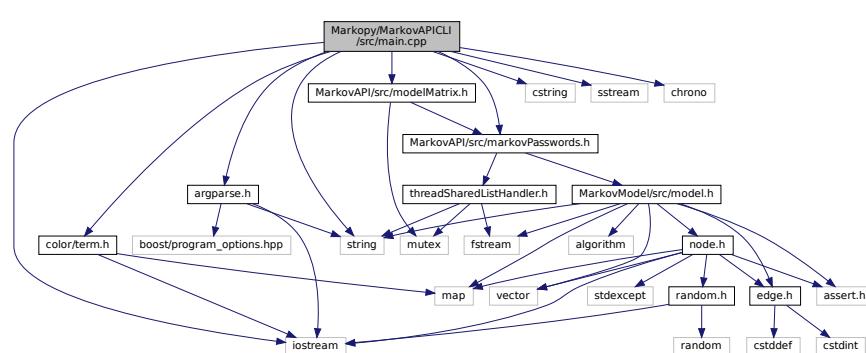
```

10.55 Markopy/MarkovAPICLI/src/main.cpp File Reference

Test cases for [Markov::API::ModelMatrix](#).

```
#include <iostream>
#include "color/term.h"
#include "argparse.h"
#include <string>
#include <cstring>
#include <sstream>
#include "MarkovAPI/src/markovPasswords.h"
#include "MarkovAPI/src/modelMatrix.h"
#include <chrono>
```

Include dependency graph for main.cpp:



Functions

- int [main](#) (int argc, char **argv)

Launch CLI tool.

10.55.1 Detailed Description

Test cases for [Markov::API::ModelMatrix](#).

Authors

Ata Hakçıl, Celal Sahir Çetiner

Parse command line arguments.

Class to flatten and reduce [Markov::Model](#) to a Matrix. Matrix level operations can be used for Generation events, with a significant performance optimization at the cost of O(N) memory complexity (O(1) memory space for slow mode)

To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for allocation. Threads are synchronized and files are flushed every 50M operations.

[Markov::Model](#) with char represented nodes. Includes wrappers for [Markov::Model](#) and additional helper functions to handle file I/O

This class is an extension of [Markov::Model<char>](#), with higher level abstractions such as train and generate. Definition in file [main.cpp](#).

10.55.2 Function Documentation

10.55.2.1 main()

```
int main (
    int argc,
    char ** argv )
```

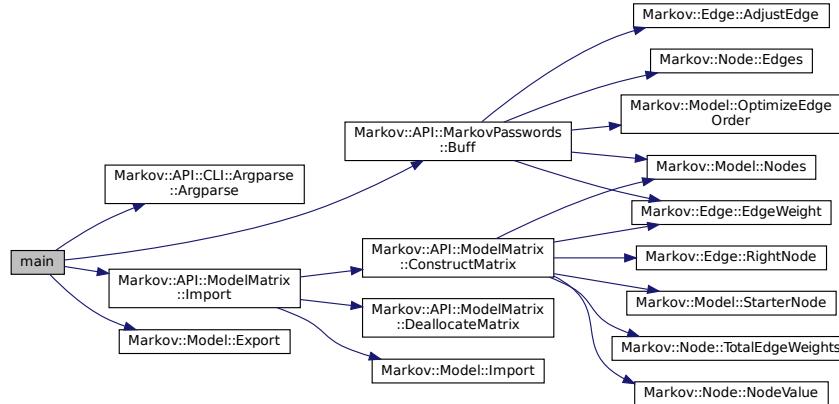
Launch CLI tool.

Definition at line 23 of file [main.cpp](#).

```
00023
00024
00025     Markov::API::CLI::Terminal t;
00026     /*
00027     ProgramOptions* p = Argparse::parse(argc, argv);
00028
00029     if (p==0 || p->bFailure) {
00030         std::cout << TERM_FAIL << "Arguments Failed to Parse" << std::endl;
00031         Argparse::help();
00032     }*/
00033     Markov::API::Argparse a(argc,argv);
00034
00035     Markov::API::ModelMatrix markovPass;
00036     std::cerr << "Importing model.\n";
00037     markovPass.Import("models/finished.mdl");
00038     std::cerr << "Import done. \n";
00039
00040     markovPass.Buff("!\\"#$%&' ()*+, -./; <=>?@[\\"\\]^_`{|}~", 100);
00041
00042     markovPass.Export("models/buffed-symbols-100fold.mdl");
00043     //markovPass.ConstructMatrix();
00044     //std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
00045     //markovPass.FastRandomWalk(50000000, "/media/ignis/Stuff/wordlist.txt", 6, 12, 25, true);
00046     //markovPass.FastRandomWalk(50000000, "/media/ignis/Stuff/wordlist2.txt", 6, 12, 25, true);
00047     //std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
00048
00049     //std::cerr << "Finished in:" << std::chrono::duration_cast<std::chrono::milliseconds> (end -
00050     begin).count() << " milliseconds" << std::endl;
00051 }
```

References [Markov::API::CLI::Argparse::Argparse\(\)](#), [Markov::API::MarkovPasswords::Buff\(\)](#), [Markov::Model< NodeStorageType >::B](#) and [Markov::API::ModelMatrix::Import\(\)](#).

Here is the call graph for this function:



10.56 main.cpp

```

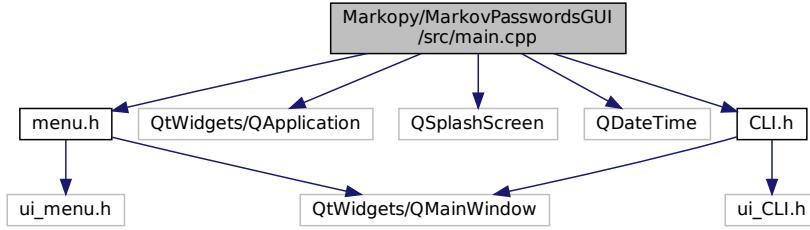
00001 /** @file main.cpp
00002 * @brief Test cases for Markov::API::ModelMatrix
00003 * @authors Ata Hakçıl, Celal Sahir Çetiner
00004 *
00005 * @copydoc Markov::API::CLI::Argparse
00006 * @copydoc Markov::API::ModelMatrix
00007 * @copydoc Markov::API::MarkovPasswords
00008 */
00009
00010 #pragma once
00011 #include <iostream>
00012 #include "color/term.h"
00013 #include "argparse.h"
00014 #include <string>
00015 #include <cstring>
00016 #include <sstream>
00017 #include "MarkovAPI/src/markovPasswords.h"
00018 #include "MarkovAPI/src/modelMatrix.h"
00019 #include <chrono>
00020
00021 /** @brief Launch CLI tool.
00022 */
00023 int main(int argc, char** argv) {
00024
00025     Markov::API::CLI::Terminal t;
00026     /*
00027     ProgramOptions* p = Argparse::parse(argc, argv);
00028
00029     if (p==0 || p->bFailure) {
00030         std::cout << TERM_FAIL << "Arguments Failed to Parse" << std::endl;
00031         Argparse::help();
00032     }*/
00033     Markov::API::CLI::Argparse a(argc, argv);
00034
00035     Markov::API::ModelMatrix markovPass;
00036     std::cerr << "Importing model.\n";
00037     markovPass.Import("models/finished.mdl");
00038     std::cerr << "Import done. \n";
00039
00040     markovPass.Buff("!\\"#$%&'()*+, -./:<=>?@[\\"\\]^_`{|}~", 100);
00041
00042     markovPass.Export("models/buffed-symbols-100fold.mdl");
00043     //markovPass.ConstructMatrix();
00044     //std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();
00045     //markovPass.FastRandomWalk(50000000, "/media/ignis/Stuff/wordlist.txt", 6, 12, 25, true);
00046     //markovPass.FastRandomWalk(500000000, "/media/ignis/Stuff/wordlist2.txt", 6, 12, 25, true);
00047     //std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
00048
00049     //std::cerr << "Finished in:" << std::chrono::duration_cast<std::chrono::milliseconds>(end -
00050     begin).count() << " milliseconds" << std::endl;
00051 }
  
```

10.57 Markopy/MarkovPasswordsGUI/src/main.cpp File Reference

Entry point for GUI.

```
#include "menu.h"
#include <QtWidgets/QApplication>
#include <QSplashScreen>
#include <QDateTime>
#include "CLI.h"
```

Include dependency graph for main.cpp:



Functions

- int [main](#) (int argc, char *argv[])

Launch UI.

10.57.1 Detailed Description

Entry point for GUI.

Authors

Yunus Emre Yılmaz

Definition in file [main.cpp](#).

10.57.2 Function Documentation

10.57.2.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Launch UI.

Definition at line 18 of file [main.cpp](#).

```
00019 {
00020
00021
00022
00023     QApplication a(argc, argv);
00024
00025     QPixmap loadingPix("views/startup.jpg");
00026     QSplashScreen splash(loadingPix);
00027     splash.show();
00028     QDateTime time = QDateTime::currentDateTime();
00029     QDateTime currentTime = QDateTime::currentDateTime(); //Record current time
00030     while (time.secsTo(currentTime) <= 5) //5 is the number of seconds to delay
00031     {
00032         currentTime = QDateTime::currentDateTime();
00033         a.processEvents();
00034     };
}
```

```

00035
00036
00037     CLI w;
00038     w.show();
00039     splash.finish(&w);
00040     return a.exec();
00041 }

```

10.58 main.cpp

```

00001 /** @file main.cpp
00002  * @brief Entry point for GUI
00003  * @authors Yunus Emre Yilmaz
00004  *
00005  */
00006
00007 //#include "MarkovPasswordsGUI.h"
00008 #include "menu.h"
00009 #include <QtWidgets/QApplication>
00010 #include <QSplashScreen>
00011 #include <QDateTime >
00012 #include "CLI.h"
00013
00014 using namespace Markov::GUI;
00015
00016 /** @brief Launch UI.
00017  */
00018 int main(int argc, char *argv[])
00019 {
00020
00021
00022
00023     QApplication a(argc, argv);
00024
00025     QPixmap loadingPix("views/startup.jpg");
00026     QSplashScreen splash(loadingPix);
00027     splash.show();
00028     QDateTime time = QDateTime::currentDateTime();
00029     QDateTime currentTime = QDateTime::currentDateTime(); //Record current time
00030     while (time.secsTo(currentTime) <= 5) //5 is the number of seconds to delay
00031     {
00032         currentTime = QDateTime::currentDateTime();
00033         a.processEvents();
00034     };
00035
00036
00037     CLI w;
00038     w.show();
00039     splash.finish(&w);
00040     return a.exec();
00041 }

```

10.59 Markopy/MarkovAPICLI/src/scripts/model_2gram.py File Reference

Namespaces

- `model_2gram`

Variables

- `model_2gram.alphabet` = `string.printable`
`password alphabet`
- `model_2gram.f` = `open('../models/2gram.mdl', "wb")`
`output file handle`

10.60 model_2gram.py

```

00001 #!/usr/bin/python3
00002 """
00003     python script for generating a 2gram model
00004 """
00005

```

```

00006 import string
00007 import re
00008
00009
00010 alphabet = string.printable
00011 alphabet = re.sub('\s', " ", alphabet)
00012 print(f"alphabet={alphabet}")
00013 #exit()
00014
00015
00016 f = open('../models/2gram.mdl', "wb")
00017 #tie start nodes
00018 for sym in alphabet:
00019     f.write(b"\x00,1," + bytes(sym, encoding='ascii') + b"\n")
00020
00021 #tie terminator nodes
00022 for sym in alphabet:
00023     f.write(bytes(sym, encoding='ascii')+ b",1,\xff\n")
00024
00025 #tie internals
00026 for src in alphabet:
00027     for target in alphabet:
00028         f.write(bytes(src, encoding='ascii') + b",1," + bytes(target, encoding='ascii') + b"\n")

```

10.61 Markopy/MarkovAPICLI/src/scripts/random_model.py File Reference

Namespaces

- [random_model](#)

Variables

- [random_model.alphabet](#) = string.printable
password alphabet
- [random_model.f](#) = open('../models/random.mdl', "wb")
output file handle

10.62 random_model.py

```

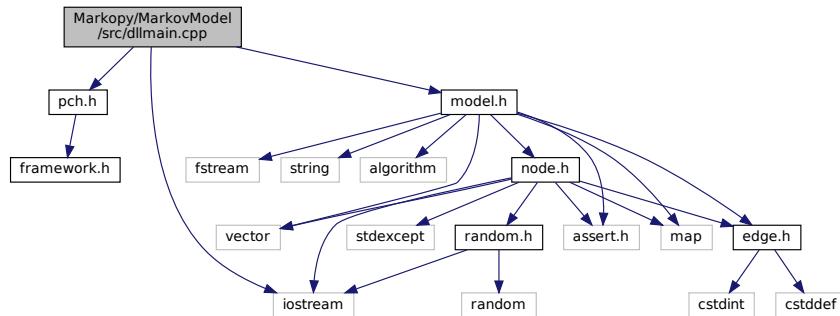
00001 #!/usr/bin/python3
00002 """
00003     python script for generating a 2gram model
00004 """
00005
00006 import string
00007 import re
00008
00009
00010 alphabet = string.printable
00011 alphabet = re.sub('\s', " ", alphabet)
00012 print(f"alphabet={alphabet}")
00013 #exit()
00014
00015
00016 f = open('../models/random.mdl', "wb")
00017 #tie start nodes
00018 for sym in alphabet:
00019     f.write(b"\x00,1," + bytes(sym, encoding='ascii') + b"\n")
00020
00021 #tie terminator nodes
00022 for sym in alphabet:
00023     f.write(bytes(sym, encoding='ascii')+ b",1,\xff\n")
00024
00025 #tie internals
00026 for src in alphabet:
00027     for target in alphabet:
00028         f.write(bytes(src, encoding='ascii') + b",1," + bytes(target, encoding='ascii') + b"\n")

```

10.63 Markopy/MarkovModel/src/dllmain.cpp File Reference

DLLMain for dynamic windows library.

```
#include "pch.h"
#include "model.h"
#include <iostream>
Include dependency graph for dllmain.cpp:
```



10.63.1 Detailed Description

DLLMain for dynamic windows library.

Authors

Ata Hakçıl

class for the final **Markov** Model, constructed from nodes and edges. Each atomic piece of the generation result is stored in a node, while edges contain the relation weights. *Extending:* To extend the class, implement the template and inherit from it, as "class MyModel : public Markov::Model<char>". For a complete demonstration of how to extend the class, see `MarkovPasswords`.

Whole model can be defined as a list of the edges, as dangling nodes are pointless. This approach is used for the import/export operations. For more information on importing/exporting model, check out the [github](#) [readme](#) and [wiki page](#).

Definition in file [dllmain.cpp](#).

10.64 dllmain.cpp

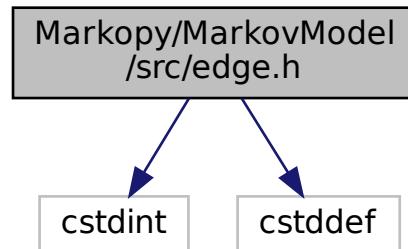
```
00001 /** @file dllmain.cpp
00002 * @brief DLLMain for dynamic windows library
00003 * @authors Ata Hakçıl
00004 *
00005 * @copydoc Markov::Model
00006 */
00007
00008 #include "pch.h"
00009 #include "model.h"
00010 #include <iostream>
00011
00012
00013 #ifdef _WIN32
00014 __declspec(dllexport) void dll_loadtest() {
00015     std::cout << "External function called.\n";
00016     //cudaTestEntry();
00017 }
00018
00019 BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)
00020 {
00021     switch (ul_reason_for_call)
00022     {
00023         case DLL_PROCESS_ATTACH:
00024         case DLL_THREAD_ATTACH:
00025         case DLL_THREAD_DETACH:
00026             case DLL_PROCESS_DETACH:
00027                 break;
00028     }
00029     return TRUE;
00030 }
00031
00032 #endif
```

10.65 Markopy/MarkovModel/src/edge.h File Reference

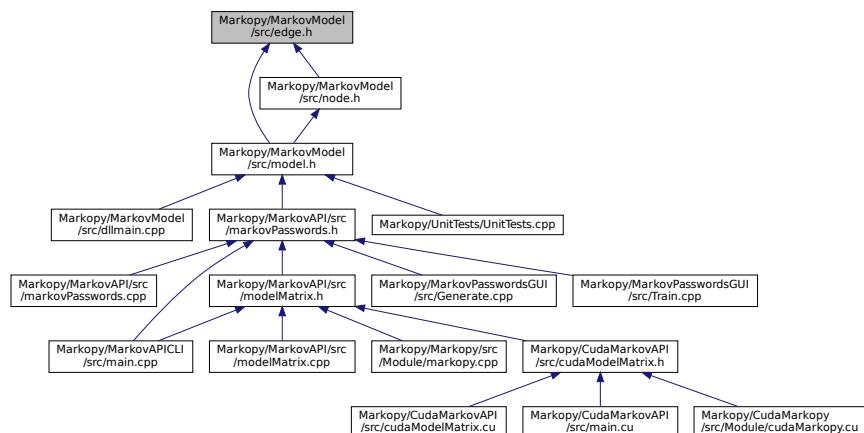
Edge class template.

```
#include <cstdint>
#include <cstddef>
```

Include dependency graph for edge.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::Edge< NodeStorageType >](#)
Edge class used to link nodes in the model together.

Namespaces

- [Markov](#)
Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.

10.65.1 Detailed Description

Edge class template.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay

Edge class used to link nodes in the model together. Has LeftNode, RightNode, and EdgeWeight of the edge. Edges are *UNIDIRECTIONAL* in this model. They can only be traversed LeftNode to RightNode.

Definition in file [edge.h](#).

10.66 edge.h

```

00001 /** @file edge.h
00002 * @brief Edge class template
00003 * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
00004 *
00005 * @copydoc Markov::Edge
00006 */
00007
00008 #pragma once
00009 #include <cstdint>
00010 #include <cstddef>
00011
00012 namespace Markov {
00013
00014     template <typename NodeStorageType>
00015     class Node;
00016
00017     /** @brief Edge class used to link nodes in the model together.
00018     *
00019     Has LeftNode, RightNode, and EdgeWeight of the edge.
00020     Edges are *UNIDIRECTIONAL* in this model. They can only be traversed LeftNode to RightNode.
00021     */
00022     template <typename NodeStorageType>
00023     class Edge {
00024     public:
00025
00026         /** @brief Default constructor.
00027         */
00028         Edge<NodeStorageType>();
00029
00030         /** @brief Constructor. Initialize edge with given RightNode and LeftNode
00031         * @param _left - Left node of this edge.
00032         * @param _right - Right node of this edge.
00033         *
00034         * @b Example @b Use: Construct edge
00035         * @code{.cpp}
00036         * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00037         * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00038         * Markov::Edge<unsigned char>* el = new Markov::Edge<unsigned char>(src, target1);
00039         * @endcode
00040         *
00041         */
00042         Edge<NodeStorageType>(Node<NodeStorageType>* _left, Node<NodeStorageType>* _right);
00043
00044         /** @brief Adjust the edge EdgeWeight with offset.
00045         * Adds the offset parameter to the edge EdgeWeight.
00046         * @param offset - NodeValue to be added to the EdgeWeight
00047         *
00048         * @b Example @b Use: Construct edge
00049         * @code{.cpp}
00050         * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00051         * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00052         * Markov::Edge<unsigned char>* el = new Markov::Edge<unsigned char>(src, target1);
00053         *
00054         * el->AdjustEdge(25);
00055         *
00056         * @endcode
00057         */
00058         void AdjustEdge(long int offset);
00059
00060         /** @brief Traverse this edge to RightNode.
00061         * @return Right node. If this is a terminator node, return NULL
00062         *
00063         *
00064         * @b Example @b Use: Traverse a node
00065         * @code{.cpp}
00066         * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00067         * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00068         * Markov::Edge<unsigned char>* el = new Markov::Edge<unsigned char>(src, target1);
00069         *
00070         * el->AdjustEdge(25);
00071         * Markov::Edge<unsigned char>* e2 = el->traverseNode();
00072         * @endcode
00073         *

```

```

00074     */
00075     inline Node<NodeStorageType>* TraverseNode();
00076
00077     /** @brief Set LeftNode of this edge.
00078      * @param node - Node to be linked with.
00079      */
00080     void SetLeftEdge (Node<NodeStorageType>* );
00081     /** @brief Set RightNode of this edge.
00082      * @param node - Node to be linked with.
00083      */
00084     void SetRightEdge(Node<NodeStorageType>* );
00085
00086     /** @brief return edge's EdgeWeight.
00087      * @return edge's EdgeWeight.
00088      */
00089     inline uint64_t EdgeWeight();
00090
00091     /** @brief return edge's LeftNode
00092      * @return edge's LeftNode.
00093      */
00094     Node<NodeStorageType>* LeftNode();
00095
00096     /** @brief return edge's RightNode
00097      * @return edge's RightNode.
00098      */
00099     inline Node<NodeStorageType>* RightNode();
00100
00101 private:
00102     /**
00103      * @brief source node
00104      */
00105     Node<NodeStorageType>* _left;
00106
00107     /**
00108      * @brief target node
00109      */
00110     Node<NodeStorageType>* _right;
00111
00112     /**
00113      * @brief Edge Edge Weight
00114      */
00115     long int _weight;
00116 };
00117
00118
00119 };
00120
00121 //default constructor of edge
00122 template <typename NodeStorageType>
00123 Markov::Edge<NodeStorageType>::Edge() {
00124     this->_left = NULL;
00125     this->_right = NULL;
00126     this->_weight = 0;
00127 }
00128 //constructor of edge
00129 template <typename NodeStorageType>
00130 Markov::Edge<NodeStorageType>::Edge(Markov::Node<NodeStorageType>* _left,
00131     Markov::Node<NodeStorageType>* _right) {
00132     this->_left = _left;
00133     this->_right = _right;
00134     this->_weight = 0;
00135 }
00136 //to AdjustEdge the edges by the edge with its offset
00137 void Markov::Edge<NodeStorageType>::AdjustEdge(long int offset) {
00138     this->_weight += offset;
00139     this->LeftNode()->UpdateTotalVerticeWeight(offset);
00140 }
00141 //to TraverseNode the node
00142 template <typename NodeStorageType>
00143 inline Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::TraverseNode() {
00144     if (this->RightNode()->NodeValue() == 0xff) //terminator node
00145         return NULL;
00146     return _right;
00147 }
00148 //to set the LeftNode of the node
00149 template <typename NodeStorageType>
00150 void Markov::Edge<NodeStorageType>::SetLeftEdge(Markov::Node<NodeStorageType>* n) {
00151     this->_left = n;
00152 }
00153 //to set the RightNode of the node
00154 template <typename NodeStorageType>
00155 void Markov::Edge<NodeStorageType>::SetRightEdge(Markov::Node<NodeStorageType>* n) {
00156     this->_right = n;
00157 }
00158 //to get the EdgeWeight of the node
00159 template <typename NodeStorageType>

```

```

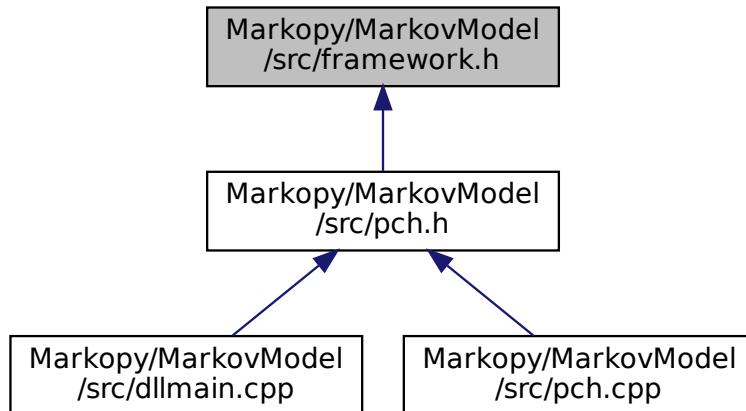
00160 inline uint64_t Markov::Edge<NodeStorageType>::EdgeWeight() {
00161     return this->_weight;
00162 }
00163 //to get the LeftNode of the node
00164 template <typename NodeStorageType>
00165 Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::LeftNode() {
00166     return this->_left;
00167 }
00168 //to get the RightNode of the node
00169 template <typename NodeStorageType>
00170 inline Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::RightNode() {
00171     return this->_right;
00172 }

```

10.67 Markopy/MarkovModel/src/framework.h File Reference

for windows dynamic library

This graph shows which files directly or indirectly include this file:



Macros

- `#define WIN32_LEAN_AND_MEAN`

10.67.1 Detailed Description

for windows dynamic library

Authors

Ata Hakçıl

Definition in file [framework.h](#).

10.67.2 Macro Definition Documentation

10.67.2.1 WIN32_LEAN_AND_MEAN

```
#define WIN32_LEAN_AND_MEAN
```

Definition at line 9 of file [framework.h](#).

10.68 framework.h

```

00001 /** @file framework.h
00002 * @brief for windows dynamic library
00003 * @authors Ata Hakçıl
00004 *
00005 */
00006
00007 #pragma once
00008
00009 #define WIN32_LEAN_AND_MEAN // Exclude rarely-used stuff from Windows headers
00010 // Windows Header Files
00011
00012 #ifdef _WIN32
00013 #include <windows.h>
00014 #endif

```

10.69 Markopy/MarkovModel/src/model.h File Reference

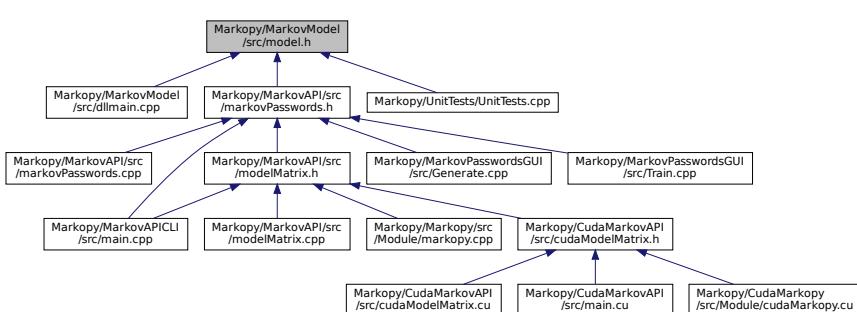
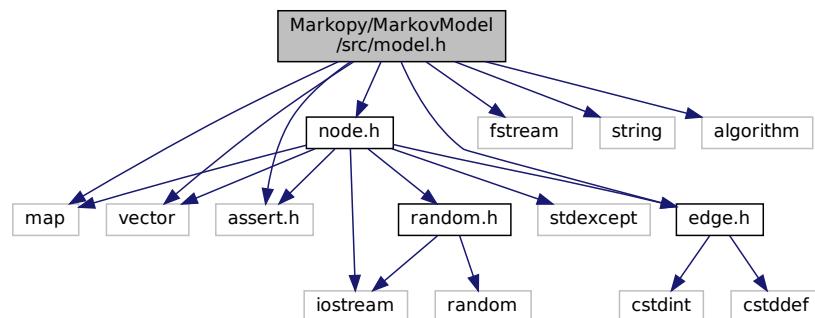
Model class template.

```

#include <map>
#include <vector>
#include <fstream>
#include <assert.h>
#include <string>
#include <algorithm>
#include "node.h"
#include "edge.h"

```

Include dependency graph for model.h:



Classes

- class [Markov::Model< NodeStorageType >](#)
class for the final [Markov Model](#), constructed from nodes and edges.

Namespaces

- [Markov](#)
Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.

10.69.1 Detailed Description

Model class template.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay

class for the final [Markov Model](#), constructed from nodes and edges. Each atomic piece of the generation result is stored in a node, while edges contain the relation weights. *Extending:* To extend the class, implement the template and inherit from it, as "class MyModel : public Markov::Model<char>". For a complete demonstration of how to extend the class, see [MarkovPasswords](#).

Whole model can be defined as a list of the edges, as dangling nodes are pointless. This approach is used for the import/export operations. For more information on importing/exporting model, check out the [github readme](#) and [wiki page](#).

Definition in file [model.h](#).

10.70 model.h

```
00001 /**
00002 * @file model.h
00003 * @brief Model class template
00004 * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
00005 * @copydoc Markov::Model
00006 */
00007
00008
00009
00010 #pragma once
00011 #include <map>
00012 #include <vector>
00013 #include <fstream>
00014 #include <assert.h>
00015 #include <string>
00016 #include <algorithm>
00017 #include "node.h"
00018 #include "edge.h"
00019
00020 /**
00021 * @brief Namespace for the markov-model related classes.
00022 * Contains Model, Node and Edge classes
00023 */
00024 namespace Markov {
00025
00026     template <typename NodeStorageType>
00027     class Node;
00028
00029     template <typename NodeStorageType>
00030     class Edge;
00031
00032     template <typename NodeStorageType>
00033
00034     /** @brief class for the final Markov Model, constructed from nodes and edges.
00035     *
00036     * Each atomic piece of the generation result is stored in a node, while edges contain the
00037     * relation weights.
00038     * *Extending:*
00039     * To extend the class, implement the template and inherit from it, as "class MyModel : public
00040     * Markov::Model<char>".
00041     * For a complete demonstration of how to extend the class, see MarkovPasswords.
00042     * Whole model can be defined as a list of the edges, as dangling nodes are pointless. This
00043     * approach is used for the import/export operations.
00044     * For more information on importing/exporting model, check out the github readme and wiki page.
```

```

00043     *
00044     */
00045     class Model {
00046     public:
00047         /** @brief Initialize a model with only start and end nodes.
00048         *
00049             * Initialize an empty model with only a starterNode
00050             * Starter node is a special kind of node that has constant 0x00 value, and will be used to
00051             * initiate the generation execution from.
00052         */
00053     Model<NodeStorageType>();
00054
00055     /** @brief Do a random walk on this model.
00056     *
00057         * Start from the starter node, on each node, invoke RandomNext using the random engine on
00058         * current node, until terminator node is reached.
00059             * If terminator node is reached before minimum length criteria is reached, ignore the last
00060             * selection and re-invoke randomNext
00061
00062             * If maximum length criteria is reached but final node is not, cut off the generation and
00063             * proceed to the final node.
00064                 * This function takes Markov::Random::RandomEngine as a parameter to generate pseudo random
00065             * numbers from
00066
00067                 * This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne
00068             * output is higher in entropy, most use cases
00069                 * don't really need super high entropy output, so Markov::Random::Marsaglia is preferable for
00070             * better performance.
00071
00072                 * This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening
00073             * via maximum length criteria.
00074
00075         * @b Example @b Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use
00076             * Marsaglia
00077                 * @code{.cpp}
00078                 * Markov::Model<char> model;
00079                 * Model.import("model.mdl");
00080                 * char* res = new char[11];
00081                 * Markov::Random::Marsaglia MarsagliaRandomEngine;
00082                 * for (int i = 0; i < 10; i++) {
00083                     *     this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
00084                     *     std::cout << res << "\n";
00085                 }
00086                 * @endcode
00087
00088                 * @param randomEngine Random Engine to use for the random walks. For examples, see
00089             * Markov::Random::Mersenne and Markov::Random::Marsaglia
00090                 * @param minSetting Minimum number of characters to generate
00091                 * @param maxSetting Maximum number of character to generate
00092                 * @param buffer buffer to write the result to
00093                 * @return Null terminated string that was generated.
00094
00095             * NodeStorageType* RandomWalk(Markov::Random::RandomEngine* randomEngine, int minSetting, int
00096             * maxSetting, NodeStorageType* buffer);
00097
00098         /** @brief Adjust the model with a single string.
00099             *
00100                 * Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from
00101                 * current node to the next, until NULL character is reached.
00102
00103                 * Then, update the edge EdgeWeight from current node, to the terminator node.
00104
00105                 * This function is used for training purposes, as it can be used for adjusting the model with
00106             * each line of the corpus file.
00107
00108                 * @b Example @b Use: Create an empty model and train it with string: "testdata"
00109                 * @code{.cpp}
00110                 * Markov::Model<char> model;
00111                 * char test[] = "testdata";
00112                 * model.AdjustEdge(test, 15);
00113                 * @endcode
00114
00115                 *
00116                 * @param string - String that is passed from the training, and will be used to AdjustEdge the
00117             * model with
00118                 * @param occurrence - Occurrence of this string.
00119
00120         void AdjustEdge(const NodeStorageType* payload, long int occurrence);
00121
00122         /** @brief Import a file to construct the model.
00123             *
00124                 * File contains a list of edges. For more info on the file format, check out the wiki and
00125             * github readme pages.
00126                 * Format is: Left_repr;EdgeWeight;right_repr

```

```

00115      *
00116      * Iterate over this list, and construct nodes and edges accordingly.
00117      * @return True if successful, False for incomplete models or corrupt file formats
00118      *
00119      * @b Example @b Use: Import a file from ifstream
00120      * @code{.cpp}
00121      * Markov::Model<char> model;
00122      * std::ifstream file("test.mdl");
00123      * model.Import(&file);
00124      * @endcode
00125      */
00126  bool Import(std::ifstream* );
00127
00128  /** @brief Open a file to import with filename, and call bool Model::Import with std::ifstream
00129  * @return True if successful, False for incomplete models or corrupt file formats
00130  *
00131  * @b Example @b Use: Import a file with filename
00132  * @code{.cpp}
00133  * Markov::Model<char> model;
00134  * model.Import("test.mdl");
00135  * @endcode
00136  */
00137  bool Import(const char* filename);
00138
00139  /** @brief Export a file of the model.
00140  *
00141  * File contains a list of edges.
00142  * Format is: Left_repr;EdgeWeight;right_repr.
00143  * For more information on the format, check out the project wiki or github readme.
00144  *
00145  * Iterate over this vertices, and their edges, and write them to file.
00146  * @return True if successful, False for incomplete models.
00147  *
00148  * @b Example @b Use: Export file to ofstream
00149  * @code{.cpp}
00150  * Markov::Model<char> model;
00151  * std::ofstream file("test.mdl");
00152  * model.Export(&file);
00153  * @endcode
00154  */
00155  bool Export(std::ofstream* );
00156
00157  /** @brief Open a file to export with filename, and call bool Model::Export with std::ofstream
00158  * @return True if successful, False for incomplete models or corrupt file formats
00159  *
00160  * @b Example @b Use: Export file to filename
00161  * @code{.cpp}
00162  * Markov::Model<char> model;
00163  * model.Export("test.mdl");
00164  * @endcode
00165  */
00166  bool Export(const char* filename);
00167
00168  /** @brief Return starter Node
00169  * @return starter node with 00 NodeValue
00170  */
00171  Node<NodeStorageType>* StarterNode(){ return starterNode; }
00172
00173  /** @brief Return a vector of all the edges in the model
00174  * @return vector of edges
00175  */
00176  std::vector<Edge<NodeStorageType>>* Edges(){ return &edges; }
00177
00178  /** @brief Return starter Node
00179  * @return starter node with 00 NodeValue
00180  */
00181  std::map<NodeStorageType, Node<NodeStorageType>>* Nodes(){ return &nodes; }
00182
00183  /** @brief Sort edges of all nodes in the model ordered by edge weights
00184  *
00185  */
00186  void OptimizeEdgeOrder();
00187
00188  private:
00189  /**
00190   * @brief Map LeftNode is the Nodes NodeValue
00191   * Map RightNode is the node pointer
00192  */
00193  std::map<NodeStorageType, Node<NodeStorageType>>* nodes;
00194
00195  /**
00196   * @brief Starter Node of this model.
00197  */
00198  Node<NodeStorageType>* starterNode;
00199
00200  /**

```

```

00202         @brief A list of all edges in this model.
00203         */
00204         std::vector<Edge<NodeStorageType*>> edges;
00205     };
00206
00207 };
00208
00209 template <typename NodeStorageType>
00210 Markov::Model<NodeStorageType>::Model() {
00211     this->starterNode = new Markov::Node<NodeStorageType>(0);
00212     this->nodes.insert({ 0, this->starterNode });
00213 }
00214
00215 template <typename NodeStorageType>
00216 bool Markov::Model<NodeStorageType>::Import(std::ifstream* f) {
00217     std::string cell;
00218
00219     char src;
00220     char target;
00221     long int oc;
00222
00223     while (std::getline(*f, cell)) {
00224         //std::cout << "cell: " << cell << std::endl;
00225         src = cell[0];
00226         target = cell[cell.length() - 1];
00227         char* j;
00228         oc = std::strtol(cell.substr(2, cell.length() - 2).c_str(), &j, 10);
00229         //std::cout << oc << "\n";
00230         Markov::Node<NodeStorageType>* srcN;
00231         Markov::Node<NodeStorageType>* targetN;
00232         Markov::Edge<NodeStorageType>* e;
00233         if (this->nodes.find(src) == this->nodes.end()) {
00234             srcN = new Markov::Node<NodeStorageType>(src);
00235             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(src, srcN));
00236             //std::cout << "Creating new node at start.\n";
00237         }
00238         else {
00239             srcN = this->nodes.find(src)->second;
00240         }
00241
00242         if (this->nodes.find(target) == this->nodes.end()) {
00243             targetN = new Markov::Node<NodeStorageType>(target);
00244             this->nodes.insert(std::pair<char, Markov::Node<NodeStorageType>>(target, targetN));
00245             //std::cout << "Creating new node at end.\n";
00246         }
00247         else {
00248             targetN = this->nodes.find(target)->second;
00249         }
00250         e = srcN->Link(targetN);
00251         e->AdjustEdge(oc);
00252         this->edges.push_back(e);
00253
00254         //std::cout << int(srcN->NodeValue()) << " --" << e->EdgeWeight() << "--> " <<
00255         int(targetN->NodeValue()) << "\n";
00256
00257     }
00258
00259     this->OptimizeEdgeOrder();
00260
00261     return true;
00262 }
00263
00264 template <typename NodeStorageType>
00265 void Markov::Model<NodeStorageType>::OptimizeEdgeOrder() {
00266     for (std::pair<unsigned char, Markov::Node<NodeStorageType>> const& x : this->nodes) {
00267         //std::cout << "Total edges in EdgesV: " << x.second->edgesV.size() << "\n";
00268         std::sort(x.second->edgesV.begin(), x.second->edgesV.end(), [] (Edge<NodeStorageType> *lhs,
00269             Edge<NodeStorageType> *rhs) ->bool{
00270             return lhs->EdgeWeight() > rhs->EdgeWeight();
00271         });
00272         //for(int i=0;i<x.second->edgesV.size();i++)
00273         //    std::cout << x.second->edgesV[i]->EdgeWeight() << ", ";
00274         //std::cout << "\n";
00275     }
00276     //std::cout << "Total number of nodes: " << this->nodes.size() << std::endl;
00277     //std::cout << "Total number of edges: " << this->edges.size() << std::endl;
00278 }
00279
00280 template <typename NodeStorageType>
00281 bool Markov::Model<NodeStorageType>::Import(const char* filename) {
00282     std::ifstream importfile;
00283     importfile.open(filename);
00284     return this->Import(&importfile);
00285 }
00286

```

```

00287 template <typename NodeStorageType>
00288 bool Markov::Model<NodeStorageType>::Export(std::ofstream* f) {
00289     Markov::Edge<NodeStorageType>* e;
00290     for (std::vector<int>::size_type i = 0; i != this->edges.size(); i++) {
00291         e = this->edges[i];
00292         //std::cout << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," <
00293         //e->RightNode()->NodeValue() << "\n";
00294         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <
00295         "\n";
00296     }
00297 }
00298
00299 template <typename NodeStorageType>
00300 bool Markov::Model<NodeStorageType>::Export(const char* filename) {
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
00305
00306 template <typename NodeStorageType>
00307 NodeStorageType* Markov::Model<NodeStorageType>::RandomWalk(Markov::Random::RandomEngine*
00308     randomEngine, int minSetting, int maxSetting, NodeStorageType* buffer) {
00309     Markov::Node<NodeStorageType>* n = this->starterNode;
00310     int len = 0;
00311     Markov::Node<NodeStorageType>* temp_node;
00312     while (true) {
00313         temp_node = n->RandomNext(randomEngine);
00314         if (len >= maxSetting) {
00315             break;
00316         }
00317         else if ((temp_node == NULL) && (len < minSetting)) {
00318             continue;
00319         }
00320         else if (temp_node == NULL) {
00321             break;
00322         }
00323         n = temp_node;
00324         buffer[len++] = n->NodeValue();
00325     }
00326     //null terminate the string
00327     buffer[len] = 0x00;
00328
00329     //do something with the generated string
00330     return buffer; //for now
00331 }
00332
00333 template <typename NodeStorageType>
00334 void Markov::Model<NodeStorageType>::AdjustEdge(const NodeStorageType* payload, long int occurrence) {
00335     NodeStorageType p = payload[0];
00336     Markov::Node<NodeStorageType>* curnode = this->starterNode;
00337     Markov::Edge<NodeStorageType>* e;
00338     int i = 0;
00339
00340     if (p == 0) return;
00341     while (p != 0) {
00342         e = curnode->FindEdge(p);
00343         if (e == NULL) return;
00344         e->AdjustEdge(occurrence);
00345         curnode = e->RightNode();
00346         p = payload[++i];
00347     }
00348
00349     e = curnode->FindEdge('\xff');
00350     e->AdjustEdge(occurrence);
00351
00352     return;
00353 }
00354
00355 }

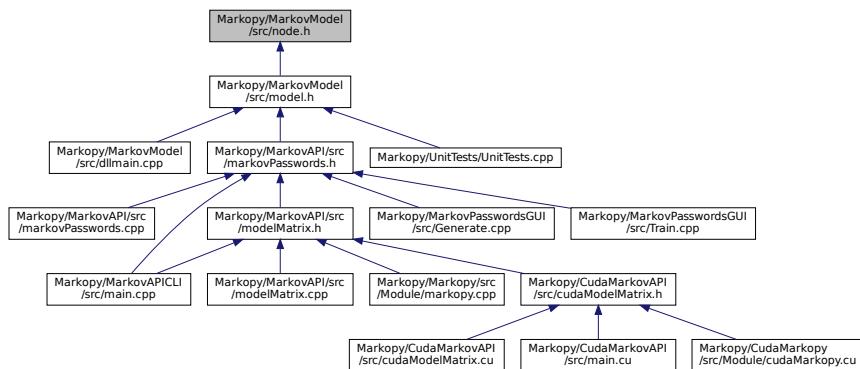
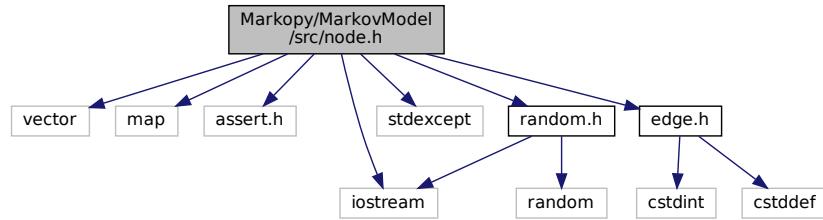
```

10.71 Markopy/MarkovModel/src/node.h File Reference

Node class template.

```
#include <vector>
#include <map>
#include <assert.h>
#include <iostream>
#include <stdexcept>
```

```
#include "edge.h"
#include "random.h"
Include dependency graph for node.h:
```



Classes

- class [Markov::Node< storageType >](#)

A node class that for the vertices of model. Connected with eachother using Edge.

Namespaces

- [Markov](#)

Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

10.71.1 Detailed Description

Node class template.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay

A node class that for the vertices of model. Connected with eachother using Edge. This class will later be templated to accept other data types than char*.

Definition in file [node.h](#).

10.72 node.h

```
00001 /** @file node.h
```

```

00002 * @brief Node class template
00003 * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
00004 *
00005 * @copydoc Markov::Node
00006 */
00007
00008
00009 #pragma once
00010 #include <vector>
00011 #include <map>
00012 #include <assert.h>
00013 #include <iostream>
00014 #include <stdexcept> // To use runtime_error
00015 #include "edge.h"
00016 #include "random.h"
00017 namespace Markov {
00018
00019     /** @brief A node class that for the vertices of model. Connected with eachother using Edge
00020     *
00021     * This class will later be templated to accept other data types than char*.
00022     */
00023     template <typename storageType>
00024     class Node {
00025     public:
00026
00027     /** @brief Default constructor. Creates an empty Node.
00028     */
00029     Node<storageType>();
00030
00031     /** @brief Constructor. Creates a Node with no edges and with given NodeValue.
00032     * @param _value - Nodes character representation.
00033     *
00034     * @b Example @b Use: Construct nodes
00035     * @code{.cpp}
00036     * Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00037     * Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00038     * @endcode
00039     */
00040     Node<storageType>(storageType _value);
00041
00042     /** @brief Link this node with another, with this node as its source.
00043     *
00044     * Creates a new Edge.
00045     * @param target - Target node which will be the RightNode() of new edge.
00046     * @return A new node with LeftNode as this, and RightNode as parameter target.
00047     *
00048     * @b Example @b Use: Construct nodes
00049     * @code{.cpp}
00050     * Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00051     * Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00052     * Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
00053     * @endcode
00054     */
00055     Edge<storageType>* Link(Node<storageType>* );
00056
00057     /** @brief Link this node with another, with this node as its source.
00058     *
00059     * *DOES NOT* create a new Edge.
00060     * @param Edge - Edge that will accept this node as its LeftNode.
00061     * @return the same edge as parameter target.
00062     *
00063     * @b Example @b Use: Construct and link nodes
00064     * @code{.cpp}
00065     * Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00066     * Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00067     * Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
00068     * LeftNode->Link(e);
00069     * @endcode
00070     */
00071     Edge<storageType>* Link(Edge<storageType>* );
00072
00073     /** @brief Chose a random node from the list of edges, with regards to its EdgeWeight, and
TraverseNode to that.
00074     *
00075     * This operation is done by generating a random number in range of 0-this.total_edge_weights,
and then iterating over the list of edges.
00076     * At each step, EdgeWeight of the edge is subtracted from the random number, and once it is
0, next node is selected.
00077     * @return Node that was chosen at EdgeWeight biased random.
00078     *
00079     * @b Example @b Use: Use randomNext to do a random walk on the model
00080     * @code{.cpp}
00081     * char* buffer[64];
00082     * Markov::Model<char> model;
00083     * model.Import("model.mdl");
00084     * Markov::Node<char>* n = model.starterNode;
00085     * int len = 0;

```

```

00086     *  Markov::Node<char>* temp_node;
00087     *  while (true) {
00088     *      temp_node = n->RandomNext(randomEngine);
00089     *      if (len >= maxSetting) {
00090     *          break;
00091     *      }
00092     *      else if ((temp_node == NULL) && (len < minSetting)) {
00093     *          continue;
00094     *      }
00095     *
00096     *      else if (temp_node == NULL) {
00097     *          break;
00098     *      }
00099     *
00100     *      n = temp_node;
00101     *
00102     *      buffer[len++] = n->NodeValue();
00103     *  }
00104     * @endcode
00105 */
00106 Node<storageType>* RandomNext (Markov::Random::RandomEngine* randomEngine);
00107
00108 /** @brief Insert a new edge to the this.edges.
00109 * @param edge - New edge that will be inserted.
00110 * @return true if insertion was successful, false if it fails.
00111 *
00112 * @b Example @b Use: Construct and update edges
00113 *
00114 * @code{.cpp}
00115 * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00116 * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00117 * Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00118 * Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00119 * Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00120 * e1->AdjustEdge(25);
00121 * src->UpdateEdges(e1);
00122 * e2->AdjustEdge(30);
00123 * src->UpdateEdges(e2);
00124 * @endcode
00125 */
00126 bool UpdateEdges (Edge<storageType>* );
00127
00128 /** @brief Find an edge with its character representation.
00129 * @param repr - character NodeValue of the target node.
00130 * @return Edge that is connected between this node, and the target node.
00131 *
00132 * @b Example @b Use: Construct and update edges
00133 *
00134 * @code{.cpp}
00135 * Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00136 * Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00137 * Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00138 * Markov::Edge<unsigned char>* res = NULL;
00139 * src->Link(target1);
00140 * src->Link(target2);
00141 * res = src->FindEdge('b');
00142 *
00143 * @endcode
00144 *
00145 */
00146 Edge<storageType>* FindEdge (storageType repr);
00147
00148 /** @brief Find an edge with its pointer. Avoid unless neccessary because computational cost
00149 * of find by character is cheaper (because of std::map)
00150 * @param target - target node.
00151 * @return Edge that is connected between this node, and the target node.
00152 */
00153 Edge<storageType>* FindEdge (Node<storageType>* target);
00154
00155 /** @brief Return character representation of this node.
00156 * @return character representation at _value.
00157 */
00158 inline unsigned char NodeValue();
00159
00160 /** @brief Change total weights with offset
00161 * @param offset to adjust the vertice weight with
00162 */
00163 void UpdateTotalVerticeWeight (long int offset);
00164
00165 /** @brief return edges
00166 */
00167 inline std::map<storageType, Edge<storageType>*>> Edges();
00168
00169 /** @brief return total edge weights
00170 */
00171 inline long int TotalEdgeWeights();
00172

```

```

00172
00173     std::vector<Edge<storageType>*> edgesV;
00174 private:
00175
00176     /**
00177      @brief Character representation of this node. 0 for starter, 0xff for terminator.
00178     */
00179     storageType _value;
00180
00181     /**
00182      @brief Total weights of the vertices, required by RandomNext
00183     */
00184     long int total_edge_weights;
00185
00186     /**
00187      @brief A map of all edges connected to this node, where this node is at the LeftNode.
00188      * Map is indexed by unsigned char, which is the character representation of the node.
00189      */
00190     std::map<storageType, Edge<storageType>*> edges;
00191 };
00192 };
00193
00194
00195
00196
00197
00198
00199
00200
00201
00202 template <typename storageType>
00203 Markov::Node<storageType>::Node(storageType _value) {
00204     this->_value = _value;
00205     this->total_edge_weights = 0L;
00206 };
00207
00208 template <typename storageType>
00209 Markov::Node<storageType>::Node() {
00210     this->_value = 0;
00211     this->total_edge_weights = 0L;
00212 };
00213
00214 template <typename storageType>
00215 inline unsigned char Markov::Node<storageType>::NodeValue() {
00216     return _value;
00217 }
00218
00219 template <typename storageType>
00220 Markov::Edge<storageType>* Markov::Node<storageType>::Link(Markov::Node<storageType>* n) {
00221     Markov::Edge<storageType>* v = new Markov::Edge<storageType>(this, n);
00222     this->UpdateEdges(v);
00223     return v;
00224 }
00225
00226 template <typename storageType>
00227 Markov::Edge<storageType>* Markov::Node<storageType>::Link(Markov::Edge<storageType>* v) {
00228     v->SetLeftEdge(this);
00229     this->UpdateEdges(v);
00230     return v;
00231 }
00232
00233 template <typename storageType>
00234 Markov::Node<storageType>* Markov::Node<storageType>::RandomNext(Markov::Random::RandomEngine* randomEngine) {
00235
00236     //get a random NodeValue in range of total_vertex_weight
00237     long int selection = randomEngine->random() %
00238         this->total_edge_weights;//distribution()(generator());// distribution(generator);
00239     //make absolute, no negative modulus values wanted
00240     //selection = (selection >= 0) ? selection : (selection + this->total_edge_weights);
00241     for(int i=0;i<this->edgesV.size();i++){
00242         selection -= this->edgesV[i]->EdgeWeight();
00243         if (selection < 0) return this->edgesV[i]->TraverseNode();
00244     }
00245     //if this assertion is reached, it means there is an implementation error above
00246     std::cout << "This should never be reached (node failed to walk to next)\n"; //cant assert from
00247     child thread
00248     assert(true && "This should never be reached (node failed to walk to next)");
00249     return NULL;
00250 }
00251
00252 template <typename storageType>
00253 bool Markov::Node<storageType>::UpdateEdges(Markov::Edge<storageType>* v) {
00254     this->edges.insert({ v->RightNode()->NodeValue(), v });
00255     this->edgesV.push_back(v);
00256     //this->total_edge_weights += v->EdgeWeight();

```

```

00256     return v->TraverseNode();
00257 }
00258
00259 template <typename storageType>
00260 Markov::Edge<storageType>* Markov::Node<storageType>::FindEdge(storageType repr) {
00261     auto e = this->edges.find(repr);
00262     if (e == this->edges.end()) return NULL;
00263     return e->second;
00264 }
00265
00266 template <typename storageType>
00267 void Markov::Node<storageType>::UpdateTotalVerticeWeight(long int offset) {
00268     this->total_edge_weights += offset;
00269 }
00270
00271 template <typename storageType>
00272 inline std::map<storageType, Markov::Edge<storageType>*>&* Markov::Node<storageType>::Edges() {
00273     return &(this->edges);
00274 }
00275
00276 template <typename storageType>
00277 inline long int Markov::Node<storageType>::TotalEdgeWeights() {
00278     return this->total_edge_weights;
00279 }

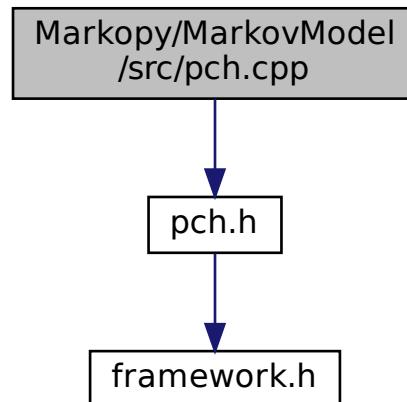
```

10.73 Markopy/MarkovModel/src/pch.cpp File Reference

For windows dynamic library.

```
#include "pch.h"
```

Include dependency graph for pch.cpp:



10.73.1 Detailed Description

For windows dynamic library.

Authors

Ata Hakçıl

Definition in file [pch.cpp](#).

10.74 pch.cpp

```

00001 /** @file pch.cpp
00002 * @brief For windows dynamic library

```

```

00003 * @authors Ata Hakçıl
00004 *
00005 */
00006
00007 // pch.cpp: source file corresponding to the pre-compiled header
00008
00009 #include "pch.h"
00010
00011 // When you are using pre-compiled headers, this source file is necessary for compilation to succeed.

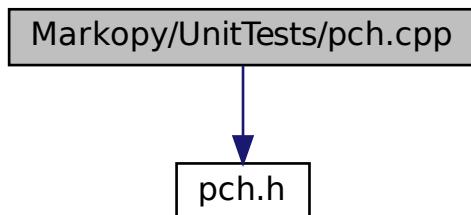
```

10.75 Markopy/UnitTests/pch.cpp File Reference

For windows dynamic library.

```
#include "pch.h"
```

Include dependency graph for pch.cpp:



10.75.1 Detailed Description

For windows dynamic library.

Authors

Ata Hakçıl

Definition in file [pch.cpp](#).

10.76 pch.cpp

```

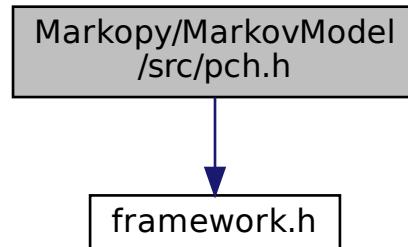
00001 /** @file pch.cpp
00002 * @brief For windows dynamic library
00003 * @authors Ata Hakçıl
00004 *
00005 */
00006 // pch.cpp: source file corresponding to the pre-compiled header
00007
00008 #include "pch.h"
00009
00010 // When you are using pre-compiled headers, this source file is necessary for compilation to succeed.

```

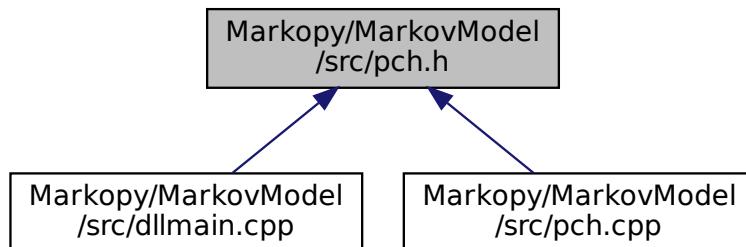
10.77 Markopy/MarkovModel/src/pch.h File Reference

For windows dynamic library.

```
#include "framework.h"
Include dependency graph for pch.h:
```



This graph shows which files directly or indirectly include this file:



10.77.1 Detailed Description

For windows dynamic library.

Authors

Ata Hakçıl

Definition in file [pch.h](#).

10.78 pch.h

```

00001 /**
00002 * @brief For windows dynamic library
00003 * @authors Ata Hakçıl
00004 *
00005 */
00006 // pch.h: This is a precompiled header file.
00007 // Files listed below are compiled only once, improving build performance for future builds.
00008 // This also affects IntelliSense performance, including code completion and many code browsing
00009 // features.
00010 // However, files listed here are ALL re-compiled if any one of them is updated between builds.
00011 // Do not add files here that you will be updating frequently as this negates the performance
00012 // advantage.
00013 #ifndef PCH_H
00014 #define PCH_H
  
```

```

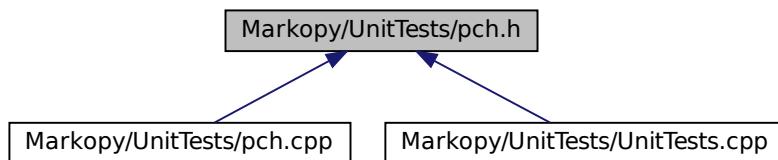
00014
00015 // add headers that you want to pre-compile here
00016 #include "framework.h"
00017
00018 #endif //PCH_H

```

10.79 Markopy/UnitTests/pch.h File Reference

For windows dynamic library.

This graph shows which files directly or indirectly include this file:



10.79.1 Detailed Description

For windows dynamic library.

Authors

Ata Hakçıl

Definition in file [pch.h](#).

10.80 pch.h

```

00001 /**
00002 * @file pch.h
00003 * @brief For windows dynamic library
00004 * @authors Ata Hakçıl
00005 */
00006 // pch.h: This is a precompiled header file.
00007 // Files listed below are compiled only once, improving build performance for future builds.
00008 // This also affects IntelliSense performance, including code completion and many code browsing
00009 // features.
00010 // However, files listed here are ALL re-compiled if any one of them is updated between builds.
00011 // Do not add files here that you will be updating frequently as this negates the performance
00012 // advantage.
00013 #ifndef PCH_H
00014 #define PCH_H
00015 // add headers that you want to pre-compile here
00016
00017 #endif //PCH_H

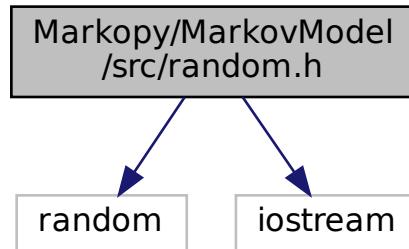
```

10.81 Markopy/MarkovModel/src/random.h File Reference

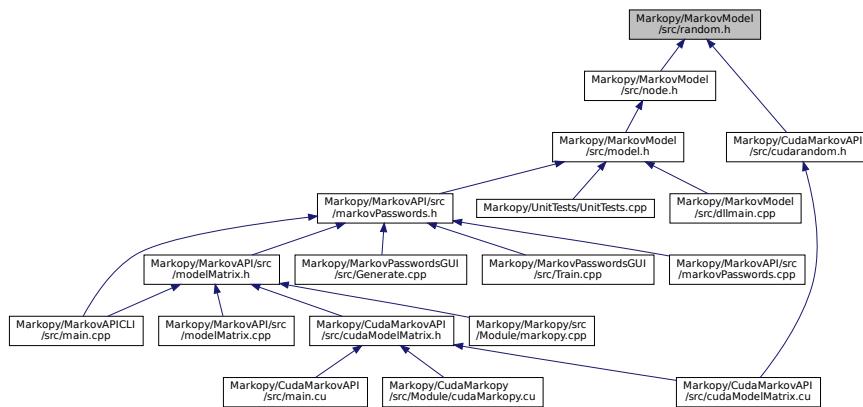
Random engine implementations for [Markov](#).

```
#include <random>
#include <iostream>
```

Include dependency graph for random.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::Random::RandomEngine](#)
An abstract class for Random Engine.
- class [Markov::Random::DefaultRandomEngine](#)
Implementation using Random.h default random engine.
- class [Markov::Random::Marsaglia](#)
Implementation of Marsaglia Random Engine.
- class [Markov::Random::Mersenne](#)
Implementation of Mersenne Twister Engine.

Namespaces

- [Markov](#)
Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- [Markov::Random](#)
Objects related to RNG.

10.81.1 Detailed Description

Random engine implementations for [Markov](#).

Authors

Ata Hakçıl

An abstract class for Random Engine. This class is used for generating random numbers, which are used for random walking on the graph.

Main reason behind allowing different random engines is that some use cases may favor performance, while some favor good random.

Mersenne can be used for truer random, while Marsaglia can be used for deterministic but fast random.

Implementation using [Random.h](#) default random engine. This engine is also used by other engines for seeding.

Example Use: Using Default Engine with RandomWalk

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::DefaultRandomEngine randomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&randomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Example Use: Generating a random number with Marsaglia Engine

```
Markov::Random::DefaultRandomEngine de;
std::cout << de.random();
```

Implementation of Marsaglia Random Engine. This is an implementation of Marsaglia Random engine, which for most use cases is a better fit than other solutions. Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.

This implementation of the Marsaglia Engine is seeded by [random.h](#) default random engine. RandomEngine is only seeded once so its not a performance issue.

Example Use: Using Marsaglia Engine with RandomWalk

```
Markov::Model<char> model;
Model.import("model.mdl");
char* res = new char[11];
Markov::Random::Marsaglia MarsagliaRandomEngine;
for (int i = 0; i < 10; i++) {
    this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
    std::cout << res << "\n";
}
```

Example Use: Generating a random number with Marsaglia Engine

```
Markov::Random::Marsaglia me;
std::cout << me.random();
```

Definition in file [random.h](#).

10.82 random.h

```
00001 /**
00002  * @file random.h
00003  * @brief Random engine implementations for Markov
00004  * @authors Ata Hakçıl
00005  *
00006  * @copydoc Markov::Random::RandomEngine
00007  * @copydoc Markov::Random::DefaultRandomEngine
00008  * @copydoc Markov::Random::Marsaglia
00009 */
00010
00011 #pragma once
00012 #include <random>
00013 #include <iostream>
00014
00015 /**
00016  * @brief Objects related to RNG
00017 */
00018 namespace Markov::Random{
00019
00020     /** @brief An abstract class for Random Engine
00021     *
00022     * This class is used for generating random numbers, which are used for random walking on the
00023     * graph.
00024     *
00025     * Main reason behind allowing different random engines is that some use cases may favor
00026     * performance,
00027     * while some favor good random.
00028     *
```

```

00027     * Mersenne can be used for truer random, while Marsaglia can be used for deterministic but fast
00028     *
00029     */
00030     class RandomEngine{
00031     public:
00032         virtual inline unsigned long random() = 0;
00033     };
00034
00035
00036
00037     /** @brief Implementation using Random.h default random engine
00038     *
00039     * This engine is also used by other engines for seeding.
00040     *
00041     *
00042     * @b Example @b Use: Using Default Engine with RandomWalk
00043     * @code{.cpp}
00044     * Markov::Model<char> model;
00045     * Model.import("model.mdl");
00046     * char* res = new char[11];
00047     * Markov::Random::DefaultRandomEngine randomEngine;
00048     * for (int i = 0; i < 10; i++) {
00049     *     this->RandomWalk(&randomEngine, 5, 10, res);
00050     *     std::cout << res << "\n";
00051     * }
00052     * @endcode
00053     *
00054     * @b Example @b Use: Generating a random number with Marsaglia Engine
00055     * @code{.cpp}
00056     * Markov::Random::DefaultRandomEngine de;
00057     * std::cout << de.random();
00058     * @endcode
00059     *
00060     */
00061     class DefaultRandomEngine : public RandomEngine{
00062     public:
00063         /** @brief Generate Random Number
00064         * @return random number in long range.
00065         */
00066         inline unsigned long random(){
00067             return this->distribution()(this->generator());
00068         }
00069     protected:
00070
00071         /** @brief Default random device for seeding
00072         *
00073         */
00074         inline std::random_device& rd() {
00075             static std::random_device _rd;
00076             return _rd;
00077         }
00078
00079         /** @brief Default random engine for seeding
00080         *
00081         */
00082         inline std::default_random_engine& generator() {
00083             static std::default_random_engine _generator(rd());
00084             return _generator;
00085         }
00086
00087         /** @brief Distribution schema for seeding.
00088         *
00089         */
00090         inline std::uniform_int_distribution<long long unsigned>& distribution() {
00091             static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffffff);
00092             return _distribution;
00093         }
00094     };
00095
00096
00097
00098     /** @brief Implementation of Marsaglia Random Engine
00099     *
00100     * This is an implementation of Marsaglia Random engine, which for most use cases is a better fit
00101     * than other solutions.
00102     * Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.
00103     * This implementation of the Marsaglia Engine is seeded by random.h default random engine.
00104     * RandomEngine is only seeded once so its not a performance issue.
00105     *
00106     * @b Example @b Use: Using Marsaglia Engine with RandomWalk
00107     * @code{.cpp}
00108     * Markov::Model<char> model;
00109     * Model.import("model.mdl");
00110     * char* res = new char[11];
00111     * Markov::Random::Marsaglia MarsagliaRandomEngine;
```

```

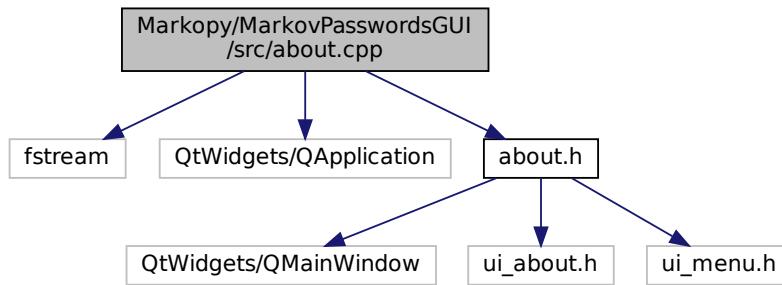
00112     * for (int i = 0; i < 10; i++) {
00113     *     this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
00114     *     std::cout << res << "\n";
00115     * }
00116     * @endcode
00117     *
00118     * @b Example @b Use: Generating a random number with Marsaglia Engine
00119     * @code{.cpp}
00120     * Markov::Random::Marsaglia me;
00121     * std::cout << me.random();
00122     * @endcode
00123     *
00124 */
00125 class Marsaglia : public DefaultRandomEngine{
00126 public:
00127     /** @brief Construct Marsaglia Engine
00128     *
00129     * Initialize x,y and z using the default random engine.
00130     */
00131     Marsaglia(){
00132         this->x = this->distribution()(this->generator());
00133         this->y = this->distribution()(this->generator());
00134         this->z = this->distribution()(this->generator());
00135         //std::cout << "x: " << x << ", y: " << y << ", z: " << z << "\n";
00136     }
00137
00138
00139     inline unsigned long random(){
00140         unsigned long t;
00141         x ^= x << 16;
00142         x ^= x >> 5;
00143         x ^= x << 1;
00144
00145         t = x;
00146         x = y;
00147         y = z;
00148         z = t ^ x ^ y;
00149
00150         return z;
00151     }
00152
00153
00154
00155     unsigned long x;
00156     unsigned long y;
00157     unsigned long z;
00158 };
00159
00160
00161     /** @brief Implementation of Mersenne Twister Engine
00162     *
00163     * This is an implementation of Mersenne Twister Engine, which is slow but is a good
00164     * implementation for high entropy pseudorandom.
00165     *
00166     * @b Example @b Use: Using Mersenne Engine with RandomWalk
00167     * @code{.cpp}
00168     * Markov::Model<char> model;
00169     * Model.import("model.mdl");
00170     * char* res = new char[11];
00171     * Markov::Random::Mersenne MersenneTwisterEngine;
00172     * for (int i = 0; i < 10; i++) {
00173     *     this->RandomWalk(&MersenneTwisterEngine, 5, 10, res);
00174     *     std::cout << res << "\n";
00175     * }
00176     * @endcode
00177     *
00178     * @b Example @b Use: Generating a random number with Marsaglia Engine
00179     * @code{.cpp}
00180     * Markov::Random::Mersenne me;
00181     * std::cout << me.random();
00182     * @endcode
00183     *
00184 */
00185 class Mersenne : public DefaultRandomEngine{
00186
00187 };
00188
00189
00190 };

```

10.83 Markopy/MarkovPasswordsGUI/src/about.cpp File Reference

About page.

```
#include <fstream>
#include <QtWidgets/QApplication>
#include "about.h"
Include dependency graph for about.cpp:
```



10.83.1 Detailed Description

About page.

Authors

Yunus Emre Yilmaz

Definition in file [about.cpp](#).

10.84 about.cpp

```

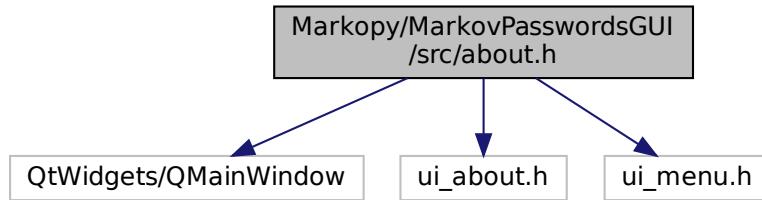
00001
00002 /** @file about.cpp
00003 * @brief About page
00004 * @authors Yunus Emre Yilmaz
00005 *
00006 */
00007
00008 #include <fstream>
00009 #include <QtWidgets/QApplication>
00010 #include "about.h"
00011
00012 using namespace Markov::GUI;
00013
00014 about::about(QWidget* parent)
00015     : QMainWindow(parent)
00016 {
00017     ui.setupUi(this);
00018 }
00019 }
```

10.85 Markopy/MarkovPasswordsGUI/src/about.h File Reference

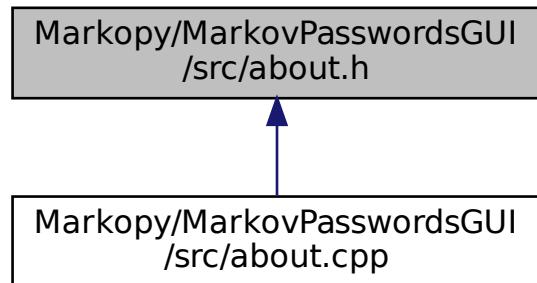
About page.

```
#include <QtWidgets/QMainWindow>
#include "ui_about.h"
#include <ui_menu.h>
```

Include dependency graph for about.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::GUI::about](#)

QT Class for about page.

Namespaces

- [Markov](#)

Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.

- [Markov::GUI](#)

namespace for MarkovPasswords [API GUI](#) wrapper

10.85.1 Detailed Description

About page.

Authors

Yunus Emre Yılmaz

Definition in file [about.h](#).

10.86 about.h

```

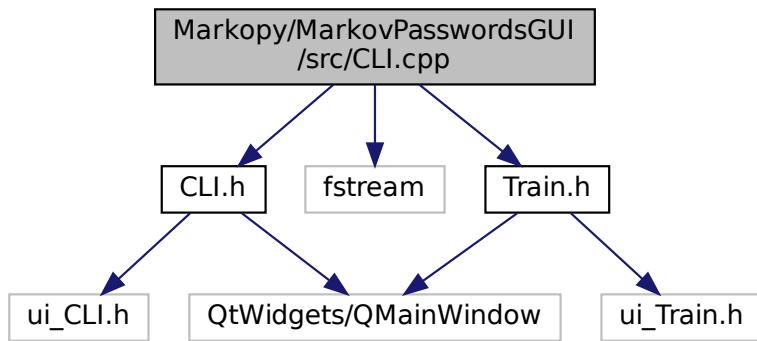
00001 /** @file about.h
00002 * @brief About page
00003 * @authors Yunus Emre Yilmaz
00004 *
00005 */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_about.h"
00010 #include <ui_menu.h>
00011
00012 /** @brief namespace for MarkovPasswords API GUI wrapper
00013 */
00014 namespace Markov::GUI{
00015
00016     /** @brief QT Class for about page
00017     */
00018     class about :public QMainWindow {
00019         Q_OBJECT
00020     public:
00021         about(QWidget* parent = Q_NULLPTR);
00022
00023     private:
00024         Ui:: main ui;
00025
00026     };
00027 }
00028 
```

10.87 Markopy/MarkovPasswordsGUI/src/CLI.cpp File Reference

CLI page.

```

#include "CLI.h"
#include <fstream>
#include "Train.h"
Include dependency graph for CLI.cpp:
```



10.87.1 Detailed Description

CLI page.

Authors

Yunus Emre Yilmaz

Definition in file [CLI.cpp](#).

10.88 CLI.cpp

```

00001 /** @file cli.cpp
00002 * @brief CLI page
00003 * @authors Yunus Emre Yilmaz
00004 *
00005 */
00006
00007 #include "CLI.h"
00008 #include <fstream>
00009 #include "Train.h"
00010
00011
00012
00013 using namespace Markov::GUI;
00014
00015 Markov::GUI::CLI::CLI(QWidget* parent)
00016     : QMainWindow(parent)
00017 {
00018     ui.setupUi(this);
00019
00020     QObject::connect(ui.startButton, &QPushButton::clicked, this, [this] {start(); });
00021     QObject::connect(ui.commandLinkButton_2, &QPushButton::clicked, this, [this] {statistics(); });
00022     QObject::connect(ui.commandLinkButton, &QPushButton::clicked, this, [this] {about(); });
00023
00024 }
00025
00026 void Markov::GUI::CLI::start() {
00027     Train* w = new Train;
00028     w->show();
00029     this->close();
00030 }
00031 void Markov::GUI::CLI::statistics() {
00032     /*
00033     statistic will show
00034     */
00035 }
00036 void Markov::GUI::CLI::about() {
00037     /*
00038     about button
00039     */
00040 }

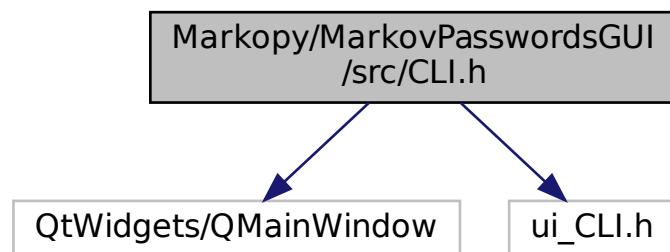
```

10.89 Markopy/MarkovPasswordsGUI/src/CLI.h File Reference

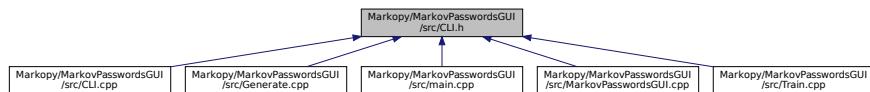
CLI page.

```
#include <QtWidgets/QMainWindow>
#include "ui_CLI.h"
```

Include dependency graph for CLI.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::GUI::CLI](#)

QT CLI Class.

Namespaces

- [Markov](#)

Namespace for the markov-model related classes. Contains Model, Node and Edge classes.

- [Markov::GUI](#)

namespace for MarkovPasswords API GUI wrapper

10.89.1 Detailed Description

CLI page.

Authors

Yunus Emre Yilmaz

Definition in file [CLI.h](#).

10.90 CLI.h

```

00001 /**
00002  * @brief CLI page
00003  * @authors Yunus Emre Yilmaz
00004 *
00005 */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_CLI.h"
00010
00011 namespace Markov::GUI{
00012     /** @brief QT CLI Class
00013     */
00014     class CLI :public QMainWindow {
00015         Q_OBJECT
00016     public:
00017         CLI(QWidget* parent = Q_NULLPTR);
00018
00019     private:
00020         Ui::CLI ui;
00021
00022     public slots:
00023         void start();
00024         void statistics();
00025         void about();
00026     };
00027 };
  
```

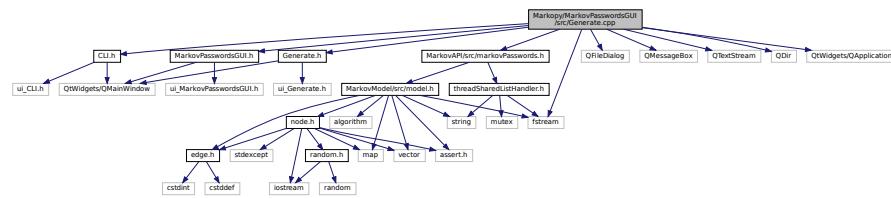
10.91 Markopy/MarkovPasswordsGUI/src/Generate.cpp File Reference

Generation Page.

```
#include "Generate.h"
#include <fstream>
```

```
#include <QFileDialog>
#include <QMessageBox>
#include <QTextStream>
#include <QDir>
#include "CLI.h"
#include "MarkovAPI/src/markovPasswords.h"
#include <QtWidgets/QApplication>
#include "MarkovPasswordsGUI.h"

Include dependency graph for Generate.cpp:
```



10.91.1 Detailed Description

Generation Page.

Authors

Yunus Emre Yilmaz

Definition in file [Generate.cpp](#).

10.92 Generate.cpp

```
00001 /** @file Generate.cpp
00002 * @brief Generation Page
00003 * @authors Yunus Emre Yilmaz
00004 *
00005 */
00006
00007 #include "Generate.h"
00008 #include <fstream>
00009 #include<QFileDialog>
00010 #include<QMessageBox>
00011 #include<QTextStream>
00012 #include<QDir>
00013 #include "CLI.h"
00014 #include "MarkovAPI/src/markovPasswords.h"
00015 #include <QtWidgets/QApplication>
00016 #include "MarkovPasswordsGUI.h"
00017 using namespace Markov::GUI;
00018
00019
00020 Generate::Generate(QWidget* parent)
00021     : QMainWindow(parent)
00022 {
00023     ui.setupUi(this);
00024
00025     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {generation(); });
00026     QObject::connect(ui.pushButton_4, &QPushButton::clicked, this, [this] {home(); });
00027     QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {train(); });
00028     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {vis(); });
00029
00030     ui.pushButton->setVisible(false);
00031     ui.lineEdit->setVisible(false);
00032     ui.lineEdit_2->setVisible(false);
00033     ui.lineEdit_3->setVisible(false);
00034     ui.label_3->setVisible(false);
00035     ui.label_4->setVisible(false);
00036     ui.label_5->setVisible(false);
00037
00038
00039
00040 }
```

```
00042 void Generate::generation() {
00043
00044
00045
00046
00047     QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00048     QFile file(file_name);
00049
00050
00051
00052     int numberPass = ui.lineEdit->text().toInt();
00053     int minLen = ui.lineEdit_2->text().toInt();
00054     int maxLen = ui.lineEdit_3->text().toInt();
00055     char* cstr;
00056     std::string fname = file_name.toStdString();
00057     cstr = new char[fname.size() + 1];
00058     strcpy(cstr, fname.c_str());
00059
00060     ui.label_6->setText("GENERATING!");
00061
00062     Markov::API::MarkovPasswords mp;
00063     mp.Import("src\\CLI\\sample_models\\2gram-trained.mdl");
00064
00065     mp.Generate(numberPass, cstr, minLen, maxLen);
00066
00067     if (!file.open(QFile::ReadOnly | QFile::Text)) {
00068         QMessageBox::warning(this, "Error", "File Not Open!");
00069     }
00070     QTextStream in(&file);
00071     QString text = in.readAll();
00072     ui.plainTextEdit->setPlainText(text);
00073
00074     ui.label_6->setText("DONE!");
00075
00076
00077
00078     file.close();
00079 }
00080
00081
00082 void Generate::train() {
00083     QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00084     QFile file(file_name);
00085
00086     if (!file.open(QFile::ReadOnly | QFile::Text)) {
00087         QMessageBox::warning(this, "Error", "File Not Open!");
00088     }
00089     QTextStream in(&file);
00090     QString text = in.readAll();
00091
00092
00093     char* cstr;
00094     std::string fname = file_name.toStdString();
00095     cstr = new char[fname.size() + 1];
00096     strcpy(cstr, fname.c_str());
00097
00098
00099
00100    char a = ',';
00101    Markov::API::MarkovPasswords mp;
00102    mp.Import("models\\2gram.mdl");
00103    mp.Train(cstr, a, 10);
00104    mp.Export("models\\finished.mdl");
00105
00106
00107
00108    ui.pushButton->setVisible(true);
00109    ui.lineEdit->setVisible(true);
00110    ui.lineEdit_2->setVisible(true);
00111    ui.lineEdit_3->setVisible(true);
00112    ui.label_3->setVisible(true);
00113    ui.label_4->setVisible(true);
00114    ui.label_5->setVisible(true);
00115
00116    file.close();
00117
00118
00119 }
00120
00121 void Generate::home() {
00122     CLI* w = new CLI;
00123     w->show();
00124     this->close();
00125 }
00126 void Generate :: vis() {
00127     MarkovPasswordsGUI* w = new MarkovPasswordsgui;
00128     w->show();
```

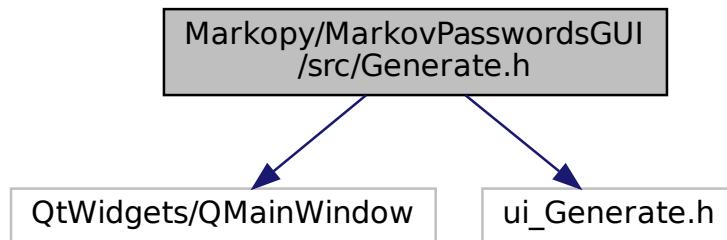
```
00129     this->close();
00130 }
```

10.93 Markopy/MarkovPasswordsGUI/src/Generate.h File Reference

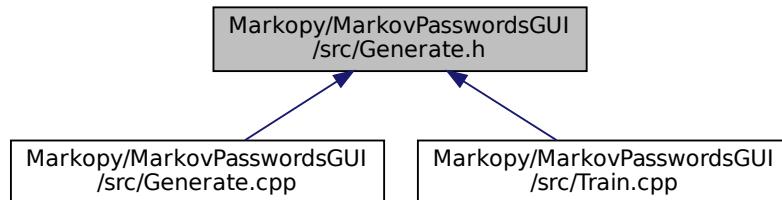
Generation Page.

```
#include <QtWidgets/QMainWindow>
#include "ui_Generate.h"
```

Include dependency graph for Generate.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::GUI::Generate](#)

QT Generation page class.

Namespaces

- [Markov](#)

Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.

- [Markov::GUI](#)

namespace for MarkovPasswords API GUI wrapper

10.93.1 Detailed Description

Generation Page.

Authors

Yunus Emre Yilmaz

Definition in file [Generate.h](#).

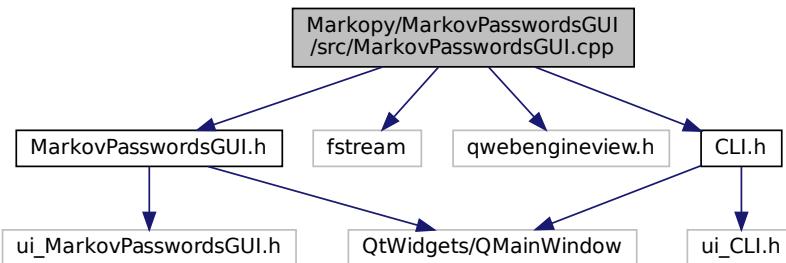
10.94 Generate.h

```
00001 /** @file Generate.h
00002 * @brief Generation Page
00003 * @authors Yunus Emre Yilmaz
00004 *
00005 */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_Generate.h"
00010
00011
00012 namespace Markov::GUI{
00013     /** @brief QT Generation page class
00014     */
00015     class Generate :public QMainWindow {
00016         Q_OBJECT
00017     public:
00018         Generate(QWidget* parent = Q_NULLPTR);
00019
00020     private:
00021         Ui::Generate ui;
00022
00023     public slots:
00024         void home();
00025         void generation();
00026         void train();
00027         void vis();
00028     };
00029 };
```

10.95 Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.cpp File Reference

Main activity page for GUI.

```
#include "MarkovPasswordsGUI.h"
#include <fstream>
#include <qwebengineview.h>
#include "CLI.h"
Include dependency graph for MarkovPasswordsGUI.cpp:
```



10.95.1 Detailed Description

Main activity page for GUI.

Authors

Yunus Emre Yilmaz

Definition in file [MarkovPasswordsGUI.cpp](#).

10.96 MarkovPasswordsGUI.cpp

```

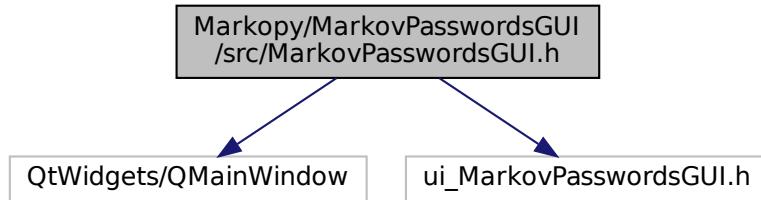
00001 /** @file MarkovPasswordsGUI.cpp
00002  * @brief Main activity page for GUI
00003  * @authors Yunus Emre Yilmaz
00004  *
00005  */
00006
00007 #include "MarkovPasswordsGUI.h"
00008 #include <fstream>
00009 #include <qwebengineview.h>
00010 #include "CLI.h"
00011
00012 using namespace Markov::GUI;
00013
00014 Markov::GUI::MarkovPasswordsGUI::MarkovPasswordsGUI(QWidget *parent) : QMainWindow(parent) {
00015     ui.setupUi(this);
00016
00017
00018     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {home(); });
00019     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {model(); });
00020     QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {pass(); });
00021 }
00022
00023
00024 void MarkovPasswordsGUI::home() {
00025     CLI* w = new CLI;
00026     w->show();
00027     this->close();
00028 }
00029 void MarkovPasswordsGUI::pass() {
00030     QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00031
00032     //get working directory
00033     char path[255];
00034     GetCurrentDirectoryA(255, path);
00035
00036     //get absolute path to the layout html
00037     std::string layout = "file:/// " + std::string(path) + "\\views\\bar.html";
00038     std::replace(layout.begin(), layout.end(), '\\', '/');
00039     webkit->setUrl(QUrl(layout.c_str()));
00040 }
00041
00042 void MarkovPasswordsGUI::model() {
00043     QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00044
00045     //get working directory
00046     char path[255];
00047     GetCurrentDirectoryA(255, path);
00048
00049     //get absolute path to the layout html
00050     std::string layout = "file:/// " + std::string(path) + "\\views\\index.html";
00051     std::replace(layout.begin(), layout.end(), '\\', '/');
00052     webkit->setUrl(QUrl(layout.c_str()));
00053 }
```

10.97 Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.h File Reference

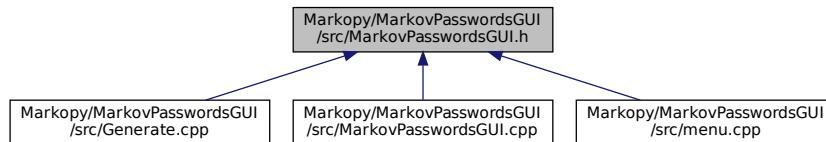
Main activity page for GUI.

```
#include <QtWidgets/QMainWindow>
#include "ui_MarkovPasswordsGUI.h"
```

Include dependency graph for MarkovPasswordsGUI.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::GUI::MarkovPasswordsGUI](#)
Reporting UI.

Namespaces

- [Markov](#)
Namespace for the markov-model related classes. Contains Model, Node and Edge classes.
- [Markov::GUI](#)
namespace for MarkovPasswords API GUI wrapper

10.97.1 Detailed Description

Main activity page for GUI.

Authors

Yunus Emre Yılmaz

Definition in file [MarkovPasswordsGUI.h](#).

10.98 MarkovPasswordsGUI.h

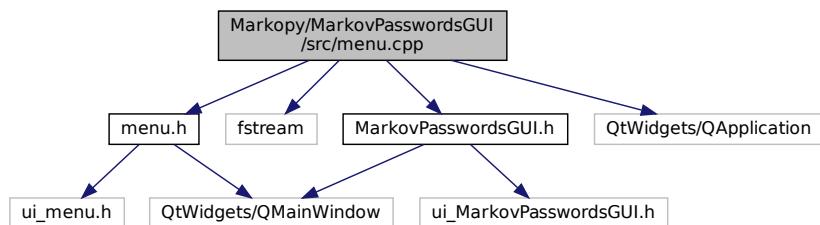
```

00001 /** @file MarkovPasswordsGUI.h
00002 * @brief Main activity page for GUI
00003 * @authors Yunus Emre Yılmaz
00004 *
00005 */
00006
00007 #pragma once
00008
00009 #include <QtWidgets/QMainWindow>
00010 #include "ui_MarkovPasswordsGUI.h"
  
```

```
00011
00012
00013
00014 namespace Markov::GUI{
00015     /** @brief Reporting UI.
00016     *
00017     * UI for reporting and debugging tools for MarkovPasswords
00018     */
00019     class MarkovPasswordsGUI : public QMainWindow {
00020         Q_OBJECT
00021     public:
00022         MarkovPasswordsGUI(QWidget* parent = Q_NULLPTR);
00023
00024     private:
00025         Ui::MarkovPasswordsGUIClass ui;
00026
00027
00028         //Slots for buttons in GUI.
00029     public slots:
00030
00031         void benchmarkSelected();
00032         //void MarkovPasswordsGUI::modelvisSelected();
00033         //void MarkovPasswordsGUI::visualDebugSelected();
00034         //void MarkovPasswordsGUI::comparisonSelected();
00035
00036
00037     public slots:
00038
00039         void home();
00040         void pass();
00041         void model();
00042     };
00043 }
```

10.99 [Markopy/MarkovPasswordsGUI/src/menu.cpp](#) File Reference

```
menu page
#include "menu.h"
#include <fstream>
#include "MarkovPasswordsGUI.h"
#include <QtWidgets/QApplication>
Include dependency graph for menu.cpp:
```



10.99.1 Detailed Description

menu page

Authors

Yunus Emre Yılmaz

Definition in file [menu.cpp](#).

10.100 menu.cpp

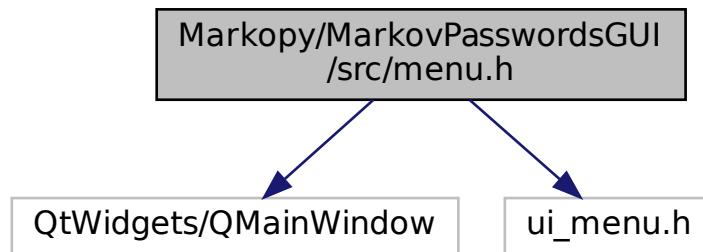
```
00001 /** @file menu.cpp  
00002 * @brief menu page
```

```

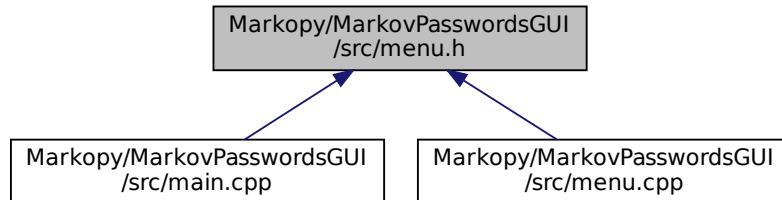
00003 * @authors Yunus Emre Yilmaz
00004 *
00005 */
00006
00007 #include "menu.h"
00008 #include <fstream>
00009 #include "MarkovPasswordsGUI.h"
00010 #include <QtWidgets/QApplication>
00011
00012 using namespace Markov::GUI;
00013
00014 menu::menu(QWidget* parent)
00015     : QMainWindow(parent)
00016 {
00017     ui.setupUi(this);
00018
00019
00020     //QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {about(); });
00021     QObject::connect(ui.visu, &QPushButton::clicked, this, [this] {visualization(); });
00022 }
00023 void menu::about() {
00024
00025
00026 }
00027 void menu::visualization() {
00028     MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
00029     w->show();
00030     this->close();
00031 }
```

10.101 Markopy/MarkovPasswordsGUI/src/menu.h File Reference

menu page
`#include <QtWidgets/QMainWindow>`
`#include "ui_menu.h"`
Include dependency graph for menu.h:



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::GUI::menu](#)

QT Menu class.

Namespaces

- [Markov](#)

Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.

- [Markov::GUI](#)

namespace for MarkovPasswords API GUI wrapper

10.101.1 Detailed Description

menu page

Authors

Yunus Emre Yılmaz

Definition in file [menu.h](#).

10.102 menu.h

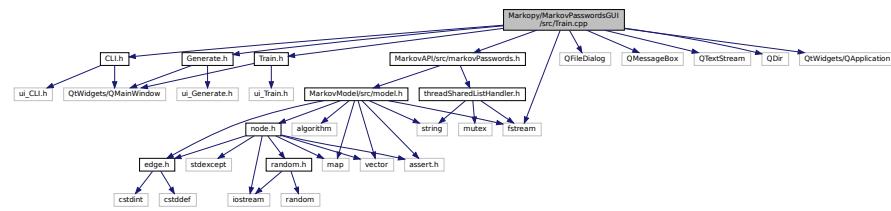
```
00001 /** @file menu.h
00002 * @brief menu page
00003 * @authors Yunus Emre Yılmaz
00004 *
00005 */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_menu.h"
00010
00011
00012 namespace Markov::GUI{
00013     /** @brief QT Menu class
00014     */
00015     class menu:public QMainWindow {
00016     Q_OBJECT
00017     public:
00018         menu(QWidget* parent = Q_NULLPTR);
00019
00020     private:
00021         Ui::main ui;
00022
00023     public slots:
00024         void about();
00025         void visualization();
00026     };
00027 };
```

10.103 Markopy/MarkovPasswordsGUI/src/Train.cpp File Reference

training page for GUI

```
#include "Train.h"
#include <fstream>
#include <QFileDialog>
#include <QMMessageBox>
#include <QTextStream>
#include <QDir>
#include "CLI.h"
#include "MarkovAPI/src/markovPasswords.h"
#include <QtWidgets/QApplication>
```

```
#include "Generate.h"
Include dependency graph for Train.cpp:
```



10.103.1 Detailed Description

training page for GUI

Authors

Yunus Emre Yilmaz

Definition in file [Train.cpp](#).

10.104 Train.cpp

```
00001 /** @file Train.cpp
00002 * @brief training page for GUI
00003 * @authors Yunus Emre Yilmaz
00004 *
00005 */
00006
00007 #include "Train.h"
00008 #include <fstream>
00009 #include<QFileDialog>
00010 #include<QMessageBox>
00011 #include<QTextStream>
00012 #include<QDir>
00013 #include "CLLI.h"
00014 #include "MarkovAPI/src/markovPasswords.h"
00015
00016 #include <QtWidgets/QApplication>
00017 #include "Generate.h"
00018
00019
00020 using namespace Markov::GUI;
00021
00022 Markov::GUI::Train::Train(QWidget* parent)
00023     : QMainWindow(parent)
00024 {
00025     ui.setupUi(this);
00026
00027
00028
00029     QObject::connect(ui.pushButton, &QPushButton::clicked, this, [this] {train(); });
00030     QObject::connect(ui.pushButton_2, &QPushButton::clicked, this, [this] {home(); });
00031 //    QObject::connect(ui.pushButton_3, &QPushButton::clicked, this, [this] {goGenerate(); });
00032
00033     //ui.pushButton_3->setVisible(false);
00034
00035
00036 }
00037
00038 void Markov::GUI::Train::train() {
00039
00040
00041
00042     QString file_name = QFileDialog::getOpenFileName(this, "Open a File", QDir::homePath());
00043     QFile file(file_name);
00044
00045     if (!file.open(QFile::ReadOnly | QFile::Text)) {
00046         QMessageBox::warning(this, "Error", "File Not Open!");
00047     }
00048     QTextStream in(&file);
00049     QString text = in.readAll();
00050     ui.plainTextEdit->setPlainText(text);
00051 }
```

```

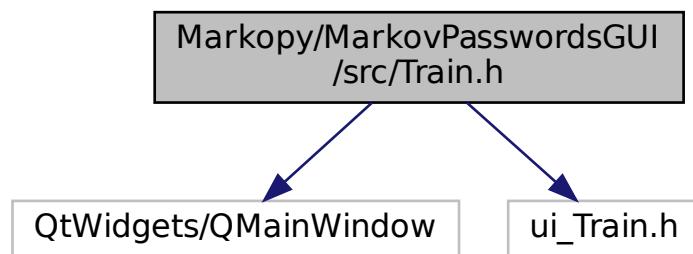
00052
00053     char* cstr;
00054     std::string fname = file_name.toStdString();
00055     cstr = new char[fname.size() + 1];
00056     strcpy(cstr, fname.c_str());
00057
00058
00059
00060     char a=','; 
00061     Markov::API::MarkovPasswords mp;
00062     mp.Import("models/2gram.mdl");
00063     mp.Train(cstr, a, 10); //please parameterize this hardcoded 10 threads
00064     mp.Export("models/finished.mdl");
00065
00066     ui.label_2->setText("Training DONE!");
00067     //ui.pushButton_3->setVisible(true);
00068
00069
00070     file.close();
00071 }
00072
00073 void Markov::GUI::Train::home() {
00074     CLI* w = new CLI;
00075     w->show();
00076     this->close();
00077 }
00078 /*void Train::goGenerate() {
00079     Generate* w = new Generate;
00080     w->show();
00081     this->close();
00082 }*/

```

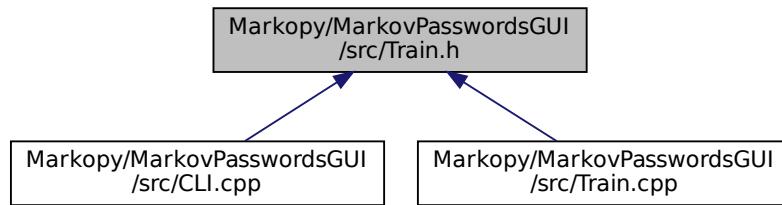
10.105 Markopy/MarkovPasswordsGUI/src/Train.h File Reference

training page for GUI

```
#include <QtWidgets/QMainWindow>
#include "ui_Train.h"
Include dependency graph for Train.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- class [Markov::GUI::Train](#)

QT Training page class.

Namespaces

- [Markov](#)

Namespace for the markov-model related classes. Contains [Model](#), [Node](#) and [Edge](#) classes.

- [Markov::GUI](#)

namespace for MarkovPasswords API GUI wrapper

10.105.1 Detailed Description

training page for GUI

Authors

Yunus Emre Yilmaz

Definition in file [Train.h](#).

10.106 Train.h

```

00001 /** @file Train.h
00002 * @brief training page for GUI
00003 * @authors Yunus Emre Yilmaz
00004 *
00005 */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_Train.h"
00010
00011 namespace Markov::GUI {
00012
00013     /** @brief QT Training page class
00014     */
00015     class Train :public QMainWindow {
00016         Q_OBJECT
00017     public:
00018         Train(QWidget* parent = Q_NULLPTR);
00019
00020     private:
00021         Ui::Train ui;
00022
00023     public slots:
00024         void home();
00025         void train();
00026     };
00027 }

```

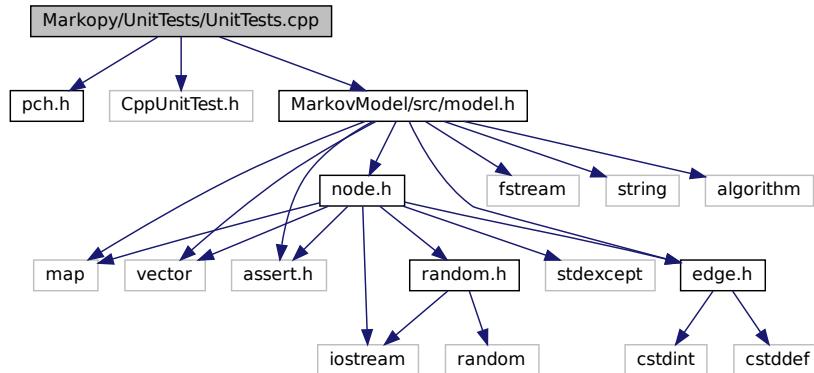
10.107 Markopy/NVSight/report.md File Reference

10.108 Markopy/README.md File Reference

10.109 Markopy/UnitTests/UnitTests.cpp File Reference

Unit tests with Microsoft::VisualStudio::CppUnitTestFramework.

```
#include "pch.h"
#include "CppUnitTest.h"
#include "MarkovModel/src/model.h"
Include dependency graph for UnitTests.cpp:
```



Namespaces

- [Testing](#)
Namespace for Microsoft Native Unit [Testing](#) Classes.
- [Testing::MVP](#)
[Testing](#) Namespace for Minimal Viable Product.
- [Testing::MVP::MarkovModel](#)
[Testing](#) Namespace for [MVP MarkovModel](#).
- [Testing::MVP::MarkovPasswords](#)
[Testing](#) namespace for [MVP MarkovPasswords](#).
- [Testing::MarkovModel](#)
[Testing](#) namespace for [MarkovModel](#).
- [Testing::MarkovPasswords](#)
[Testing](#) namespace for [MarkovPasswords](#).

Functions

- [Testing::MVP::MarkovModel::TEST_CLASS](#) (`Edge`)
Test class for minimal viable Edge.
- [Testing::MVP::MarkovModel::TEST_CLASS](#) (`Node`)
Test class for minimal viable Node.
- [Testing::MVP::MarkovModel::TEST_CLASS](#) (`Model`)
Test class for minimal viable Model.
- [Testing::MVP::MarkovPasswords::TEST_CLASS](#) (`ArgParser`)
Test Class for Argparse class.

- **Testing::MarkovModel::TEST_CLASS (Edge)**
Test class for rest of Edge cases.
- **Testing::MarkovModel::TEST_CLASS (Node)**
Test class for rest of Node cases.
- **Testing::MarkovModel::TEST_CLASS (Model)**
Test class for rest of model cases.

10.109.1 Detailed Description

Unit tests with Microsoft::VisualStudio::CppUnitTestFramework.

Authors

Ata Hakçıl, Osman Ömer Yıldıztugay, Yunus Emre Yılmaz

Definition in file [UnitTests.cpp](#).

10.110 UnitTests.cpp

```

00001 /** @file UnitTests.cpp
00002 * @brief Unit tests with Microsoft::VisualStudio::CppUnitTestFramework
00003 * @authors Ata Hakçıl, Osman Ömer Yıldıztugay, Yunus Emre Yılmaz
00004 *
00005 */
00006
00007 #include "pch.h"
00008 #include "CppUnitTest.h"
00009 #include "MarkovModel/src/model.h"
00010
00011 using namespace Microsoft::VisualStudio::CppUnitTestFramework;
00012
00013
00014 /** @brief Namespace for Microsoft Native Unit Testing Classes
00015 */
00016 namespace Testing {
00017
00018     /** @brief Testing Namespace for Minimal Viable Product
00019     */
00020     namespace MVP {
00021         /** @brief Testing Namespace for MVP MarkovModel
00022         */
00023         namespace MarkovModel {
00024             {
00025                 /** @brief Test class for minimal viable Edge
00026                 */
00027                 TEST_CLASS(Edge)
00028                 {
00029                     public:
00030
00031                         /** @brief test default constructor
00032                         */
00033                         TEST_METHOD(default_constructor) {
00034                             Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>;
00035                             Assert::IsNull(e->LeftNode());
00036                             Assert::IsNull(e->RightNode());
00037                             delete e;
00038                         }
00039
00040                         /** @brief test linked constructor with two nodes
00041                         */
00042                         TEST_METHOD(linked_constructor) {
00043                             Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>'l';
00044                             Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>'r';
00045                             Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00046                                         RightNode);
00047                             Assert::IsTrue(LeftNode == e->LeftNode());
00048                             Assert::IsTrue(RightNode == e->RightNode());
00049                             delete LeftNode;
00050                             delete RightNode;
00051                             delete e;
00052                         }
00053
00054                         /** @brief test AdjustEdge function
00055                         */
00056                         TEST_METHOD(AdjustEdge) {
00057                             Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>'l';
00058                             Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>'r';

```

```

00058     RightNode);
00059     e->AdjustEdge(15);
00060     Assert::AreEqual(15ull, e->EdgeWeight());
00061     e->AdjustEdge(15);
00062     Assert::AreEqual(30ull, e->EdgeWeight());
00063     delete LeftNode;
00064     delete RightNode;
00065     delete e;
00066 }
00067
00068 /** @brief test TraverseNode returning RightNode
00069 */
00070 TEST_METHOD(TraverseNode) {
00071     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00072     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00073     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00074     RightNode);
00075     Assert::IsTrue(RightNode == e->TraverseNode());
00076     delete LeftNode;
00077     delete RightNode;
00078     delete e;
00079 }
00080
00081 /** @brief test LeftNode/RightNode setter
00082 */
00083 TEST_METHOD(set_left_and_right) {
00084     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00085     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00086     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(LeftNode,
00087     RightNode);
00088     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>;
00089     e2->SetLeftEdge(LeftNode);
00090     e2->SetRightEdge(RightNode);
00091
00092     Assert::IsTrue(e1->LeftNode() == e2->LeftNode());
00093     Assert::IsTrue(e1->RightNode() == e2->RightNode());
00094     delete LeftNode;
00095     delete RightNode;
00096     delete e1;
00097     delete e2;
00098 }
00099
00100 /** @brief test negative adjustments
00101 */
00102 TEST_METHOD(negative_adjust) {
00103     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00104     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00105     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00106     RightNode);
00107     e->AdjustEdge(15);
00108     Assert::AreEqual(15ull, e->EdgeWeight());
00109     e->AdjustEdge(-15);
00110     Assert::AreEqual(0ull, e->EdgeWeight());
00111     delete LeftNode;
00112     delete RightNode;
00113     delete e;
00114 }
00115
00116 /** @brief Test class for minimal viable Node
00117 */
00118 TEST_CLASS(Node)
00119 {
00120     public:
00121
00122     /** @brief test default constructor
00123 */
00124     TEST_METHOD(default_constructor) {
00125         Markov::Node<unsigned char>* n = new Markov::Node<unsigned char>();
00126         Assert::AreEqual((unsigned char)0, n->NodeValue());
00127         delete n;
00128     }
00129
00130     /** @brief test custom constructor with unsigned char
00131 */
00132     TEST_METHOD(uchar_constructor) {
00133         Markov::Node<unsigned char>* n = NULL;
00134         unsigned char test_cases[] = { 'c', 0x00, 0xff, -32 };
00135         for (unsigned char tcase : test_cases) {
00136             n = new Markov::Node<unsigned char>(tcase);
00137             Assert::AreEqual(tcase, n->NodeValue());
00138             delete n;
00139         }
00140     }
}

```

```

00141     /** @brief test link function
00142     */
00143     TEST_METHOD(link_left) {
00144         Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00145         Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00146
00147         Markov::Edge<unsigned char>* e = LeftNode->Link(RightNode);
00148         delete LeftNode;
00149         delete RightNode;
00150         delete e;
00151     }
00152
00153     /** @brief test link function
00154     */
00155     TEST_METHOD(link_right) {
00156         Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00157         Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00158
00159         Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(NULL, RightNode);
00160         LeftNode->Link(e);
00161         Assert::IsTrue(LeftNode == e->LeftNode());
00162         Assert::IsTrue(RightNode == e->RightNode());
00163         delete LeftNode;
00164         delete RightNode;
00165         delete e;
00166     }
00167
00168     /** @brief test RandomNext with low values
00169     */
00170     TEST_METHOD(rand_next_low) {
00171         Markov::Random::Marsaglia MarsagliaRandomEngine;
00172         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00173         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00174         Markov::Edge<unsigned char>* e = src->Link(target1);
00175         e->AdjustEdge(15);
00176         Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00177         Assert::IsTrue(res == target1);
00178         delete src;
00179         delete target1;
00180         delete e;
00181     }
00182
00183
00184     /** @brief test RandomNext with 32 bit high values
00185     */
00186     TEST_METHOD(rand_next_u32) {
00187         Markov::Random::Marsaglia MarsagliaRandomEngine;
00188         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00189         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00190         Markov::Edge<unsigned char>* e = src->Link(target1);
00191         e->AdjustEdge(1 << 31);
00192         Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00193         Assert::IsTrue(res == target1);
00194         delete src;
00195         delete target1;
00196         delete e;
00197     }
00198
00199
00200     /** @brief random next on a node with no follow-ups
00201     */
00202     TEST_METHOD(rand_next_choice_1) {
00203         Markov::Random::Marsaglia MarsagliaRandomEngine;
00204         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00205         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00206         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00207         Markov::Edge<unsigned char>* e1 = src->Link(target1);
00208         Markov::Edge<unsigned char>* e2 = src->Link(target2);
00209         e1->AdjustEdge(1);
00210         e2->AdjustEdge((unsigned long)(ull << 31));
00211         Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00212         Assert::IsNotNull(res);
00213         Assert::IsTrue(res == target2);
00214         delete src;
00215         delete target1;
00216         delete e1;
00217         delete e2;
00218     }
00219
00220
00221     /** @brief random next on a node with no follow-ups
00222     */
00223     TEST_METHOD(rand_next_choice_2) {
00224         Markov::Random::Marsaglia MarsagliaRandomEngine;
00225         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00226         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00227         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00228         Markov::Edge<unsigned char>* e1 = src->Link(target1);

```

```

00228     Markov::Edge<unsigned char>* e2 = src->Link(target2);
00229     e2->AdjustEdge(1);
00230     e1->AdjustEdge((unsigned long)(ull << 31));
00231     Markov::Node<unsigned char>* res = src->RandomNext (&MarsagliaRandomEngine);
00232     Assert::IsNotNull(res);
00233     Assert::IsTrue(res == target1);
00234     delete src;
00235     delete target1;
00236     delete e1;
00237     delete e2;
00238 }
00239
00240 /**
00241 */
00242 TEST_METHOD(update_edges_count) {
00243
00244     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>'a';
00245     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>'b';
00246     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>'c';
00247     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00248     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00249     e1->AdjustEdge(25);
00250     src->UpdateEdges(e1);
00251     e2->AdjustEdge(30);
00252     src->UpdateEdges(e2);
00253
00254     Assert::AreEqual((size_t)2, src->Edges()->size());
00255
00256     delete src;
00257     delete target1;
00258     delete e1;
00259     delete e2;
00260
00261 }
00262
00263 /**
00264 */
00265 TEST_METHOD(update_edges_total) {
00266
00267     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>'a';
00268     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>'b';
00269     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00270     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target1);
00271     e1->AdjustEdge(25);
00272     src->UpdateEdges(e1);
00273     e2->AdjustEdge(30);
00274     src->UpdateEdges(e2);
00275
00276     //Assert::AreEqual(55ull, src->TotalEdgeWeights());
00277
00278     delete src;
00279     delete target1;
00280     delete e1;
00281     delete e2;
00282
00283 }
00284
00285 /**
00286 */
00287 TEST_METHOD(find_vertice) {
00288
00289     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>'a';
00290     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>'b';
00291     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>'c';
00292     Markov::Edge<unsigned char>* res = NULL;
00293     src->Link(target1);
00294     src->Link(target2);
00295
00296
00297     res = src->FindEdge('b');
00298     Assert::IsNotNull(res);
00299     Assert::AreEqual((unsigned char)'b', res->TraverseNode()->NodeValue());
00300     res = src->FindEdge('c');
00301     Assert::IsNotNull(res);
00302     Assert::AreEqual((unsigned char)'c', res->TraverseNode()->NodeValue());
00303
00304     delete src;
00305     delete target1;
00306     delete target2;
00307
00308
00309 }
00310
00311
00312 /**
00313 */
00314

```

```

00315     TEST_METHOD(find_vertice_without_any) {
00316
00317     auto _invalid_next = [] {
00318         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00319         Markov::Edge<unsigned char>* res = NULL;
00320
00321         res = src->FindEdge('b');
00322         Assert::IsNull(res);
00323
00324         delete src;
00325     };
00326
00327     //Assert::ExpectException<std::logic_error>(_invalid_next);
00328 }
00329
00330 /**
00331 */
00332 TEST_METHOD(find_vertice_nonexistent) {
00333
00334     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00335     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00336     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00337     Markov::Edge<unsigned char>* res = NULL;
00338     src->Link(target1);
00339     src->Link(target2);
00340
00341     res = src->FindEdge('D');
00342     Assert::IsNull(res);
00343
00344     delete src;
00345     delete target1;
00346     delete target2;
00347
00348 }
00349 }
00350
00351 /**
00352 */
00353 TEST_CLASS(Model)
00354 {
00355     public:
00356         /**
00357             @brief test model constructor for starter node
00358         */
00359         TEST_METHOD(model_constructor) {
00360             Markov::Model<unsigned char> m;
00361             Assert::AreEqual((unsigned char)'\\0', m.StarterNode()->NodeValue());
00362         }
00363
00364         /**
00365             @brief test import
00366         */
00367         TEST_METHOD(import_filename) {
00368             Markov::Model<unsigned char> m;
00369             Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
00370         }
00371
00372         /**
00373             @brief test export
00374         */
00375         TEST_METHOD(export_filename) {
00376             Markov::Model<unsigned char> m;
00377             Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
00378         }
00379
00380         /**
00381             @brief test random walk
00382         */
00383         TEST_METHOD(random_walk) {
00384             unsigned char* res = new unsigned char[12 + 5];
00385             Markov::Random::Marsaglia MarsagliaRandomEngine;
00386             Markov::Model<unsigned char> m;
00387             Assert::IsTrue(m.Import("../Models/finished2.mdl"));
00388             Assert::IsNotNull(m.RandomWalk(&MarsagliaRandomEngine, 1, 12, res));
00389         }
00390     };
00391 }
00392
00393 /**
00394     @brief Testing namespace for MVP MarkovPasswords
00395 */
00396 namespace MarkovPasswords
00397 {
00398     /**
00399         @brief Test Class for Argparse class
00400     */
00401     TEST_CLASS(ArgParser)
00402     {
00403         public:
00404             /**
00405                 @brief test basic generate
00406             */
00407             TEST_METHOD(generate_basic) {
00408                 int argc = 8;

```

```

00402     char *argv[] = {"markov.exe", "generate", "-if", "model.mdl", "-of",
00403     "passwords.txt", "-n", "100"};
00404
00405     /*ProgramOptions *p = Argparse::parse(argc, argv);
00406     Assert::IsNotNull(p);
00407
00408     Assert::AreEqual(p->bImport, true);
00409     Assert::AreEqual(p->bExport, false);
00410     Assert::AreEqual(p->importname, "model.mdl");
00411     Assert::AreEqual(p->outputfilename, "passwords.txt");
00412     Assert::AreEqual(p->generateN, 100); */
00413 }
00414
00415 /** @brief test basic generate reordered params
00416 */
00417 TEST_METHOD(generate_basic_reordered) {
00418     int argc = 8;
00419     char *argv[] = {"markov.exe", "generate", "-n", "100", "-if", "model.mdl", "-of",
00420     "passwords.txt" };
00421
00422     /*ProgramOptions* p = Argparse::parse(argc, argv);
00423     Assert::IsNotNull(p);
00424
00425     Assert::AreEqual(p->bImport, true);
00426     Assert::AreEqual(p->bExport, false);
00427     Assert::AreEqual(p->importname, "model.mdl");
00428     Assert::AreEqual(p->outputfilename, "passwords.txt");
00429     Assert::AreEqual(p->generateN, 100);*/
00430 }
00431
00432 /** @brief test basic generate param longnames
00433 */
00434 TEST_METHOD(generate_basic_longname) {
00435     int argc = 8;
00436     char *argv[] = {"markov.exe", "generate", "-n", "100", "--inputfilename",
00437     "model.mdl", "--outputfilename", "passwords.txt" };
00438
00439     /*ProgramOptions* p = Argparse::parse(argc, argv);
00440     Assert::IsNotNull(p);
00441
00442     Assert::AreEqual(p->bImport, true);
00443     Assert::AreEqual(p->bExport, false);
00444     Assert::AreEqual(p->importname, "model.mdl");
00445     Assert::AreEqual(p->outputfilename, "passwords.txt");
00446     Assert::AreEqual(p->generateN, 100); */
00447 }
00448
00449 /** @brief test basic generate
00450 */
00451 TEST_METHOD(generate_fail_badmethod) {
00452     int argc = 8;
00453     char *argv[] = {"markov.exe", "junk", "-n", "100", "--inputfilename",
00454     "model.mdl", "--outputfilename", "passwords.txt" };
00455
00456     /*ProgramOptions* p = Argparse::parse(argc, argv);
00457     Assert::IsNull(p); */
00458 }
00459
00460 /** @brief test basic train
00461 */
00462 TEST_METHOD(train_basic) {
00463     int argc = 4;
00464     char *argv[] = {"markov.exe", "train", "-ef", "model.mdl" };
00465
00466     /*ProgramOptions* p = Argparse::parse(argc, argv);
00467     Assert::IsNotNull(p);
00468
00469     Assert::AreEqual(p->bImport, false);
00470     Assert::AreEqual(p->bExport, true);
00471     Assert::AreEqual(p->exportname, "model.mdl"); */
00472 }
00473
00474 /** @brief test basic generate
00475 */
00476 TEST_METHOD(train_basic_longname) {
00477     int argc = 4;
00478     char *argv[] = {"markov.exe", "train", "--exportfilename", "model.mdl" };
00479
00480     /*ProgramOptions* p = Argparse::parse(argc, argv);
00481     Assert::IsNotNull(p);
00482
00483     Assert::AreEqual(p->bImport, false);
00484     Assert::AreEqual(p->bExport, true);
00485     Assert::AreEqual(p->exportname, "model.mdl"); */
00486 }

```

```

00485
00486
00487
00488     };
00489
00490     }
00491 }
00492
00493
00494 /** @brief Testing namespace for MarkovModel
00495 */
00496 namespace MarkovModel {
00497
00498     /** @brief Test class for rest of Edge cases
00499     */
00500     TEST_CLASS(Edge)
00501     {
00502         public:
00503             /** @brief send exception on integer underflow
00504             */
00505             TEST_METHOD(except_integer_underflow) {
00506                 auto _underflow_adjust = [] {
00507                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00508                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00509                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00510                                         RightNode);
00511                     e->AdjustEdge(15);
00512                     e->AdjustEdge(-30);
00513                     delete LeftNode;
00514                     delete RightNode;
00515                     delete e;
00516                 };
00517                 Assert::ExpectException<std::underflow_error>(_underflow_adjust);
00518             }
00519
00520             /** @brief test integer overflows
00521             */
00522             TEST_METHOD(except_integer_overflow) {
00523                 auto _overflow_adjust = [] {
00524                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00525                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00526                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00527                                         RightNode);
00528                     e->AdjustEdge(~0ull);
00529                     e->AdjustEdge(1);
00530                     delete LeftNode;
00531                     delete RightNode;
00532                     delete e;
00533                 };
00534                 Assert::ExpectException<std::underflow_error>(_overflow_adjust);
00535             }
00536
00537             /** @brief Test class for rest of Node cases
00538             */
00539             TEST_CLASS(Node)
00540             {
00541                 public:
00542
00543                 /** @brief test RandomNext with 64 bit high values
00544                 */
00545                 TEST_METHOD(rand_next_u64) {
00546                     Markov::Random::Marsaglia MarsagliaRandomEngine;
00547                     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00548                     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00549                     Markov::Edge<unsigned char>* e = src->Link(target1);
00550                     e->AdjustEdge((unsigned long)(ull << 63));
00551                     Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00552                     Assert::IsTrue(res == target1);
00553                     delete src;
00554                     delete target1;
00555                     delete e;
00556                 }
00557
00558                 /** @brief test RandomNext with 64 bit high values
00559                 */
00560                 TEST_METHOD(rand_next_u64_max) {
00561                     Markov::Random::Marsaglia MarsagliaRandomEngine;
00562                     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00563                     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00564                     Markov::Edge<unsigned char>* e = src->Link(target1);
00565                     e->AdjustEdge((0xffffffff));
00566                     Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00567                     Assert::IsTrue(res == target1);
00568                     delete src;
00569                     delete target1;

```

```

00570         delete e;
00571     }
00572
00573     /** @brief randomNext when no edges are present
00574     */
00575     TEST_METHOD(uninitialized_rand_next) {
00576
00577         auto _invalid_next = [] {
00578             Markov::Random::Marsaglia MarsagliaRandomEngine;
00579             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00580             Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00581             Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(src, target1);
00582             Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00583
00584             delete src;
00585             delete target1;
00586             delete e;
00587         };
00588
00589         Assert::ExpectException<std::logic_error>(_invalid_next);
00590     }
00591
00592
00593     /**
00594     */
00595
00596     /** @brief Test class for rest of model cases
00597     */
00598     TEST_CLASS(Model)
00599     {
00600         public:
00601             TEST_METHOD(functional_random_walk) {
00602                 unsigned char* res2 = new unsigned char[12 + 5];
00603                 Markov::Random::Marsaglia MarsagliaRandomEngine;
00604                 Markov::Model<unsigned char> m;
00605                 Markov::Node<unsigned char>* starter = m.StarterNode();
00606                 Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00607                 Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00608                 Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00609                 Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00610                 starter->Link(a)->AdjustEdge(1);
00611                 a->Link(b)->AdjustEdge(1);
00612                 b->Link(c)->AdjustEdge(1);
00613                 c->Link(end)->AdjustEdge(1);
00614
00615                 char* res = (char*)m.RandomWalk(&MarsagliaRandomEngine, 1, 12, res2);
00616                 Assert::IsFalse(strcmp(res, "abc"));
00617             }
00618             TEST_METHOD(functionoal_random_walk_without_any) {
00619                 Markov::Model<unsigned char> m;
00620                 Markov::Node<unsigned char>* starter = m.StarterNode();
00621                 Markov::Node<unsigned char>* a = new Markov::Node<unsigned char>('a');
00622                 Markov::Node<unsigned char>* b = new Markov::Node<unsigned char>('b');
00623                 Markov::Node<unsigned char>* c = new Markov::Node<unsigned char>('c');
00624                 Markov::Node<unsigned char>* end = new Markov::Node<unsigned char>(0xff);
00625                 Markov::Edge<unsigned char>* res = NULL;
00626                 starter->Link(a)->AdjustEdge(1);
00627                 a->Link(b)->AdjustEdge(1);
00628                 b->Link(c)->AdjustEdge(1);
00629                 c->Link(end)->AdjustEdge(1);
00630
00631                 res = starter->FindEdge('D');
00632                 Assert::IsNull(res);
00633             }
00634         };
00635     };
00636
00637 }
00638
00639 /** @brief Testing namespace for MarkovPasswords
00640 */
00641 namespace MarkovPasswords {
00642
00643 };
00644
00645 }

```

Index

`__init__`
 Python.CudaMarkopy.CudaMarkopyCLI, 134
 Python.CudaMarkopy.CudaModelMatrixCLI, 308
 Python.Markopy.BaseCLI, 102
 Python.Markopy.MarkopyCLI, 401
 Python.Markopy.MarkovPasswordsCLI, 500
 Python.Markopy.ModelMatrixCLI, 627

`_generate`
 Python.CudaMarkopy.CudaMarkopyCLI, 134
 Python.CudaMarkopy.CudaModelMatrixCLI, 308
 Python.Markopy.AbstractGenerationModelCLI, 58
 Python.Markopy.AbstractTrainingModelCLI, 73
 Python.Markopy.BaseCLI, 103
 Python.Markopy.MarkopyCLI, 401
 Python.Markopy.MarkovPasswordsCLI, 500
 Python.Markopy.ModelMatrixCLI, 627

`_left`
 Markov::Edge< NodeStorageType >, 389

`_right`
 Markov::Edge< NodeStorageType >, 389

`_value`
 Markov::Node< storageType >, 667

`_weight`
 Markov::Edge< NodeStorageType >, 390

`about`
 Markov::GUI::about, 55
 Markov::GUI::CLI, 116
 Markov::GUI::menu, 554

`add_arguments`
 Python.CudaMarkopy.CudaMarkopyCLI, 135, 136
 Python.CudaMarkopy.CudaModelMatrixCLI, 309
 Python.Markopy.AbstractGenerationModelCLI, 59
 Python.Markopy.AbstractTrainingModelCLI, 73
 Python.Markopy.BaseCLI, 104
 Python.Markopy.MarkopyCLI, 402
 Python.Markopy.MarkovPasswordsCLI, 501
 Python.Markopy.ModelMatrixCLI, 628

`AdjustEdge`
 Markov::API::CUDA::CUDAModelMatrix, 268
 Markov::API::MarkovPasswords, 483
 Markov::API::ModelMatrix, 574
 Markov::Edge< NodeStorageType >, 387
 Markov::Model< NodeStorageType >, 562
 Python.CudaMarkopy.CudaMarkopyCLI, 136, 138, 139
 Python.CudaMarkopy.CudaModelMatrixCLI, 309, 310
 Python.Markopy.MarkopyCLI, 403
 Python.Markopy.MarkovModel, 466
 Python.Markopy.MarkovPasswordsCLI, 501
 Python.Markopy.ModelMatrix, 599
 Python.Markopy.ModelMatrixCLI, 628

`AllocVRAMOutputBuffer`

Markov::API::CUDA::CUDAModelMatrix, 268
Python.CudaMarkopy.CudaMarkopyCLI, 140
Python.CudaMarkopy.CudaModelMatrixCLI, 311

`alphabet`
 model_2gram, 36
 random_model, 38

`alternatingKernels`
 Markov::API::CUDA::CUDAModelMatrix, 296
 Python.CudaMarkopy.CudaMarkopyCLI, 253
 Python.CudaMarkopy.CudaModelMatrixCLI, 374

`Argparse`
 Markov::API::CLI::Argparse, 95

`argparse.h`
 BOOST_ALL_STATIC_LIB, 736
 BOOST_PROGRAM_OPTIONS_STATIC_LIB, 736

`args`
 Python.CudaMarkopy.CudaMarkopyCLI, 254
 Python.CudaMarkopy.CudaModelMatrixCLI, 374
 Python.Markopy.AbstractGenerationModelCLI, 68
 Python.Markopy.AbstractTrainingModelCLI, 91, 92
 Python.Markopy.BaseCLI, 113
 Python.Markopy.MarkopyCLI, 458
 Python.Markopy.MarkovPasswordsCLI, 531
 Python.Markopy.ModelMatrixCLI, 657

`base`, 25

`benchmarkSelected`
 Markov::GUI::MarkovPasswordsGUI, 535

`bExport`
 Markov::API::CLI::__programOptions, 52

`bFailure`
 Markov::API::CLI::__programOptions, 52

`bImport`
 Markov::API::CLI::__programOptions, 52

`bInfinite`
 Python.CudaMarkopy.CudaMarkopyCLI, 254
 Python.CudaMarkopy.CudaModelMatrixCLI, 374

`BLACK`
 Markov::API::CLI::Terminal, 672

`BLUE`
 Markov::API::CLI::Terminal, 672

`BOOST_ALL_STATIC_LIB`
 argparse.h, 736
 markopy.cpp, 715

`BOOST_PROGRAM_OPTIONS_STATIC_LIB`
 argparse.h, 736

`BOOST_PYTHON_MODULE`
 Markov::Markopy, 33
 Markov::Markopy::CUDA, 34

`BOOST_PYTHON_STATIC_LIB`
 cudaMarkopy.cu, 685
 markopy.cpp, 715

`BROWN`
 Markov::API::CLI::Terminal, 672

Buff
 Markov::API::CUDA::CUDAModelMatrix, 269
 Markov::API::MarkovPasswords, 484
 Markov::API::ModelMatrix, 574
 Python.CudaMarkopy.CudaMarkopyCLI, 140, 141, 143, 144
 Python.CudaMarkopy.CudaModelMatrixCLI, 311, 312
 Python.Markopy.MarkopyCLI, 404, 405
 Python.Markopy.MarkovModel, 466
 Python.Markopy.MarkovPasswordsCLI, 502
 Python.Markopy.ModelMatrix, 600
 Python.Markopy.ModelMatrixCLI, 629

check_corpus_path
 Python.CudaMarkopy.CudaMarkopyCLI, 145–148
 Python.CudaMarkopy.CudaModelMatrixCLI, 313, 314
 Python.Markopy.AbstractGenerationModelCLI, 60
 Python.Markopy.AbstractTrainingModelCLI, 74
 Python.Markopy.BaseCLI, 104
 Python.Markopy.MarkopyCLI, 407, 408
 Python.Markopy.MarkovPasswordsCLI, 503, 504
 Python.Markopy.ModelMatrixCLI, 630

check_export_path
 Python.CudaMarkopy.CudaMarkopyCLI, 148–151
 Python.CudaMarkopy.CudaModelMatrixCLI, 314, 315
 Python.Markopy.AbstractGenerationModelCLI, 60
 Python.Markopy.AbstractTrainingModelCLI, 75
 Python.Markopy.BaseCLI, 105
 Python.Markopy.MarkopyCLI, 409, 410
 Python.Markopy.MarkovPasswordsCLI, 504, 505
 Python.Markopy.ModelMatrixCLI, 631

check_import_path
 Python.CudaMarkopy.CudaMarkopyCLI, 151–154
 Python.CudaMarkopy.CudaModelMatrixCLI, 315, 316
 Python.Markopy.AbstractGenerationModelCLI, 61
 Python.Markopy.AbstractTrainingModelCLI, 76
 Python.Markopy.BaseCLI, 105
 Python.Markopy.MarkopyCLI, 411, 412
 Python.Markopy.MarkovPasswordsCLI, 505, 506
 Python.Markopy.ModelMatrixCLI, 631

CLI
 Markov::GUI::CLI, 115

cli
 Python.CudaMarkopy.CudaMarkopyCLI, 254
 Python.Markopy.MarkopyCLI, 458

color
 Markov::API::CLI::Terminal, 672

colormap
 Markov::API::CLI::Terminal, 672

ConstructMatrix
 Markov::API::CUDA::CUDAModelMatrix, 270
 Markov::API::ModelMatrix, 576
 Python.CudaMarkopy.CudaMarkopyCLI, 154, 156, 158

Python.CudaMarkopy.CudaModelMatrixCLI, 316, 318
 Python.Markopy.MarkopyCLI, 413
 Python.Markopy.ModelMatrix, 601
 Python.Markopy.ModelMatrixCLI, 632

cudaBlocks
 Markov::API::CUDA::CUDAModelMatrix, 296
 Python.CudaMarkopy.CudaMarkopyCLI, 254
 Python.CudaMarkopy.CudaModelMatrixCLI, 374

CudaCheckNotifyErr
 Markov::API::CUDA::CUDADeviceController, 120
 Markov::API::CUDA::CUDAModelMatrix, 272
 Markov::API::CUDA::Random::Marsaglia, 539
 Python.CudaMarkopy.CudaMarkopyCLI, 159
 Python.CudaMarkopy.CudaModelMatrixCLI, 320

cudaGridSize
 Markov::API::CUDA::CUDAModelMatrix, 296
 Python.CudaMarkopy.CudaMarkopyCLI, 254
 Python.CudaMarkopy.CudaModelMatrixCLI, 374

CudaMalloc2DToFlat
 Markov::API::CUDA::CUDADeviceController, 120
 Markov::API::CUDA::CUDAModelMatrix, 272
 Markov::API::CUDA::Random::Marsaglia, 540
 Python.CudaMarkopy.CudaMarkopyCLI, 160
 Python.CudaMarkopy.CudaModelMatrixCLI, 320

cudamarkopy, 25
 cudammx, 25
 markopy, 25
 mp, 25
 spec, 25

cudaMarkopy.cu
 BOOST_PYTHON_STATIC_LIB, 685

CudaMemcpy2DToFlat
 Markov::API::CUDA::CUDADeviceController, 121
 Markov::API::CUDA::CUDAModelMatrix, 273
 Markov::API::CUDA::Random::Marsaglia, 541
 Python.CudaMarkopy.CudaMarkopyCLI, 161
 Python.CudaMarkopy.CudaModelMatrixCLI, 321

cudaMemPerGrid
 Markov::API::CUDA::CUDAModelMatrix, 296
 Python.CudaMarkopy.CudaMarkopyCLI, 254
 Python.CudaMarkopy.CudaModelMatrixCLI, 375

CudaMigrate2DFlat
 Markov::API::CUDA::CUDADeviceController, 122
 Markov::API::CUDA::CUDAModelMatrix, 274
 Markov::API::CUDA::Random::Marsaglia, 542
 Python.CudaMarkopy.CudaMarkopyCLI, 162
 Python.CudaMarkopy.CudaModelMatrixCLI, 322

cudammx, 25
 cudamarkopy, 25
 ext, 25
 markopy, 26
 mp, 26
 spec, 26

cudaPerKernelAllocationSize
 Markov::API::CUDA::CUDAModelMatrix, 297
 Python.CudaMarkopy.CudaMarkopyCLI, 254
 Python.CudaMarkopy.CudaModelMatrixCLI, 375

cudastreams
 Markov::API::CUDA::CUDAModelMatrix, 297
 Python.CudaMarkopy.CudaMarkopyCLI, 254
 Python.CudaMarkopy.CudaModelMatrixCLI, 375

cudaThreads
 Markov::API::CUDA::CUDAModelMatrix, 297
 Python.CudaMarkopy.CudaMarkopyCLI, 255
 Python.CudaMarkopy.CudaModelMatrixCLI, 375

CYAN
 Markov::API::CLI::Terminal, 672

DARKGRAY
 Markov::API::CLI::Terminal, 672

datasetFile
 Markov::API::CUDA::CUDAModelMatrix, 297
 Markov::API::MarkovPasswords, 495
 Markov::API::ModelMatrix, 594
 Python.CudaMarkopy.CudaMarkopyCLI, 255
 Python.CudaMarkopy.CudaModelMatrixCLI, 375
 Python.Markopy.MarkopyCLI, 458
 Python.Markopy.ModelMatrix, 478
 Python.Markopy.MarkovPasswordsCLI, 531
 Python.Markopy.ModelMatrix, 620
 Python.Markopy.ModelMatrixCLI, 657

datasetname
 Markov::API::CLI::_programOptions, 52

DeallocateMatrix
 Markov::API::CUDA::CUDAModelMatrix, 275
 Markov::API::ModelMatrix, 577
 Python.CudaMarkopy.CudaMarkopyCLI, 163, 164
 Python.CudaMarkopy.CudaModelMatrixCLI, 323, 324
 Python.Markopy.MarkopyCLI, 414
 Python.Markopy.ModelMatrix, 603
 Python.Markopy.ModelMatrixCLI, 633

device_edgeMatrix
 Markov::API::CUDA::CUDAModelMatrix, 297
 Python.CudaMarkopy.CudaMarkopyCLI, 255
 Python.CudaMarkopy.CudaModelMatrixCLI, 375

device_matrixIndex
 Markov::API::CUDA::CUDAModelMatrix, 297
 Python.CudaMarkopy.CudaMarkopyCLI, 255
 Python.CudaMarkopy.CudaModelMatrixCLI, 375

device_outputBuffer
 Markov::API::CUDA::CUDAModelMatrix, 297
 Python.CudaMarkopy.CudaMarkopyCLI, 255
 Python.CudaMarkopy.CudaModelMatrixCLI, 375

device_seeds
 Markov::API::CUDA::CUDAModelMatrix, 297
 Python.CudaMarkopy.CudaMarkopyCLI, 255
 Python.CudaMarkopy.CudaModelMatrixCLI, 376

device_totalEdgeWeights
 Markov::API::CUDA::CUDAModelMatrix, 298
 Python.CudaMarkopy.CudaMarkopyCLI, 255
 Python.CudaMarkopy.CudaModelMatrixCLI, 376

device_valueMatrix
 Markov::API::CUDA::CUDAModelMatrix, 298
 Python.CudaMarkopy.CudaMarkopyCLI, 255
 Python.CudaMarkopy.CudaModelMatrixCLI, 376

devrandom
 Markov::API::CUDA::Random, 32

distribution
 Markov::API::CUDA::Random::Marsaglia, 543
 Markov::Random::DefaultRandomEngine, 383
 Markov::Random::Marsaglia, 549
 Markov::Random::Mersenne, 557

DumpJSON
 Markov::API::CUDA::CUDAModelMatrix, 276
 Markov::API::ModelMatrix, 578
 Python.CudaMarkopy.CudaMarkopyCLI, 165–167
 Python.CudaMarkopy.CudaModelMatrixCLI, 325, 326
 Python.Markopy.MarkopyCLI, 415
 Python.Markopy.ModelMatrix, 604
 Python.Markopy.ModelMatrixCLI, 634

Edge
 Markov::Edge< NodeStorageType >, 386

edgeMatrix
 Markov::API::CUDA::CUDAModelMatrix, 298
 Markov::API::ModelMatrix, 594
 Python.CudaMarkopy.CudaMarkopyCLI, 256
 Python.CudaMarkopy.CudaModelMatrixCLI, 376
 Python.Markopy.MarkopyCLI, 458
 Python.Markopy.ModelMatrix, 620
 Python.Markopy.ModelMatrixCLI, 657

Edges
 Markov::API::CUDA::CUDAModelMatrix, 277
 Markov::API::MarkovPasswords, 485
 Markov::API::ModelMatrix, 579
 Markov::Model< NodeStorageType >, 562
 Markov::Node< storageType >, 662
 Python.CudaMarkopy.CudaMarkopyCLI, 168, 169
 Python.CudaMarkopy.CudaModelMatrixCLI, 327
 Python.Markopy.MarkopyCLI, 416
 Python.Markopy.ModelMatrix, 468
 Python.Markopy.MarkovPasswordsCLI, 506
 Python.Markopy.ModelMatrix, 605
 Python.Markopy.ModelMatrixCLI, 635

edges
 Markov::API::CUDA::CUDAModelMatrix, 298
 Markov::API::MarkovPasswords, 495
 Markov::API::ModelMatrix, 594
 Markov::Model< NodeStorageType >, 568
 Markov::Node< storageType >, 667
 Python.CudaMarkopy.CudaMarkopyCLI, 256
 Python.CudaMarkopy.CudaModelMatrixCLI, 376
 Python.Markopy.MarkopyCLI, 459
 Python.Markopy.ModelMatrix, 478
 Python.Markopy.MarkovPasswordsCLI, 532
 Python.Markopy.ModelMatrix, 620
 Python.Markopy.ModelMatrixCLI, 658

edgesV
 Markov::Node< storageType >, 667

EdgeWeight
 Markov::Edge< NodeStorageType >, 387

endl
 Markov::API::CLI::Terminal, 673

Export
 Markov::API::CUDA::CUDAModelMatrix, 277
 Markov::API::MarkovPasswords, 486
 Markov::API::ModelMatrix, 579
 Markov::Model< NodeStorageType >, 563
 Python.CudaMarkopy.CudaMarkopyCLI, 169, 170, 174–176
 Python.CudaMarkopy.CudaModelMatrixCLI, 327, 329, 330
 Python.Markopy.MarkopyCLI, 416, 417, 420, 421
 Python.Markopy.MarkovModel, 468, 469
 Python.Markopy.MarkovPasswordsCLI, 506, 508, 509
 Python.Markopy.ModelMatrix, 605
 Python.Markopy.ModelMatrixCLI, 635, 636

export
 Python.CudaMarkopy.CudaMarkopyCLI, 170–174
 Python.CudaMarkopy.CudaModelMatrixCLI, 328
 Python.Markopy.AbstractGenerationModelCLI, 61
 Python.Markopy.AbstractTrainingModelCLI, 77, 78
 Python.Markopy.BaseCLI, 106
 Python.Markopy.MarkopyCLI, 417–419
 Python.Markopy.MarkovPasswordsCLI, 507
 Python.Markopy.ModelMatrixCLI, 636

exportname
 Markov::API::CLI::__programOptions, 53

ext
 cudammx, 25
 markopy, 27

f
 model_2gram, 36
 random_model, 38

FastRandomWalk
 Markov::API::CUDA::CUDAModelMatrix, 278, 280, 281
 Markov::API::ModelMatrix, 580, 581
 Python.CudaMarkopy.CudaMarkopyCLI, 176, 177, 179–184
 Python.CudaMarkopy.CudaModelMatrixCLI, 330, 332–335
 Python.Markopy.MarkopyCLI, 421, 422
 Python.Markopy.ModelMatrix, 606, 607
 Python.Markopy.ModelMatrixCLI, 637, 638

FastRandomWalkCUDAKernel
 Markov::API::CUDA, 30

FastRandomWalkPartition
 Markov::API::CUDA::CUDAModelMatrix, 282
 Markov::API::ModelMatrix, 582
 Python.CudaMarkopy.CudaMarkopyCLI, 185–187
 Python.CudaMarkopy.CudaModelMatrixCLI, 336, 337
 Python.Markopy.MarkopyCLI, 423
 Python.Markopy.ModelMatrix, 608
 Python.Markopy.ModelMatrixCLI, 639

FastRandomWalkThread
 Markov::API::CUDA::CUDAModelMatrix, 283
 Markov::API::ModelMatrix, 583

Python.CudaMarkopy.CudaMarkopyCLI, 188, 190, 191
 Python.CudaMarkopy.CudaModelMatrixCLI, 339, 340
 Python.Markopy.MarkopyCLI, 424
 Python.Markopy.ModelMatrix, 609
 Python.Markopy.ModelMatrixCLI, 640

fileIO
 Python.CudaMarkopy.CudaMarkopyCLI, 256
 Python.CudaMarkopy.CudaModelMatrixCLI, 376
 Python.Markopy.MarkopyCLI, 459
 Python.Markopy.ModelMatrixCLI, 658

FindEdge
 Markov::Node< storageType >, 662, 663

flatEdgeMatrix
 Markov::API::CUDA::CUDAModelMatrix, 298
 Python.CudaMarkopy.CudaMarkopyCLI, 256
 Python.CudaMarkopy.CudaModelMatrixCLI, 376

FlattenMatrix
 Markov::API::CUDA::CUDAModelMatrix, 284
 Python.CudaMarkopy.CudaMarkopyCLI, 193
 Python.CudaMarkopy.CudaModelMatrixCLI, 342

flatValueMatrix
 Markov::API::CUDA::CUDAModelMatrix, 298
 Python.CudaMarkopy.CudaMarkopyCLI, 256
 Python.CudaMarkopy.CudaModelMatrixCLI, 377

framework.h
 WIN32_LEAN_AND_MEAN, 754

GatherAsyncKernelOutput
 Markov::API::CUDA::CUDAModelMatrix, 285
 Python.CudaMarkopy.CudaMarkopyCLI, 193
 Python.CudaMarkopy.CudaModelMatrixCLI, 342

Generate
 Markov::API::CUDA::CUDAModelMatrix, 285
 Markov::API::MarkovPasswords, 486
 Markov::API::ModelMatrix, 585
 Markov::GUI::Generate, 391
 Python.CudaMarkopy.CudaMarkopyCLI, 193, 200–203
 Python.CudaMarkopy.CudaModelMatrixCLI, 344, 345
 Python.Markopy.MarkopyCLI, 426, 430, 431
 Python.Markopy.MarkovModel, 469
 Python.Markopy.MarkovPasswordsCLI, 509, 511
 Python.Markopy.ModelMatrix, 611
 Python.Markopy.ModelMatrixCLI, 643

generate
 Python.CudaMarkopy.CudaMarkopyCLI, 194–199
 Python.CudaMarkopy.CudaModelMatrixCLI, 342, 343
 Python.Markopy.AbstractGenerationModelCLI, 62
 Python.Markopy.AbstractTrainingModelCLI, 78, 79
 Python.Markopy.BaseCLI, 106
 Python.Markopy.MarkopyCLI, 426–429
 Python.Markopy.MarkovPasswordsCLI, 509, 510
 Python.Markopy.ModelMatrixCLI, 642

generateN
 Markov::API::CLI::__programOptions, 53

GenerateThread
Markov::API::CUDA::CUDAModelMatrix, 286
Markov::API::MarkovPasswords, 487
Markov::API::ModelMatrix, 586
Python.CudaMarkopy.CudaMarkopyCLI, 205
Python.CudaMarkopy.CudaModelMatrixCLI, 346
Python.Markopy.MarkopyCLI, 432
Python.Markopy.MarkovModel, 470
Python.Markopy.MarkovPasswordsCLI, 512
Python.Markopy.ModelMatrix, 612
Python.Markopy.ModelMatrixCLI, 644

generation
Markov::GUI::Generate, 392

generator
Markov::API::CUDA::Random::Marsaglia, 543
Markov::Random::DefaultRandomEngine, 383
Markov::Random::Marsaglia, 550
Markov::Random::Mersenne, 557

getProgramOptions
Markov::API::CLI::Argparse, 97

GREEN
Markov::API::CLI::Terminal, 672

help
Markov::API::CLI::Argparse, 97
Python.CudaMarkopy.CudaMarkopyCLI, 205
Python.CudaMarkopy.CudaModelMatrixCLI, 347, 348
Python.Markopy.AbstractGenerationModelCLI, 63
Python.Markopy.AbstractTrainingModelCLI, 80, 81
Python.Markopy.BaseCLI, 107
Python.Markopy.MarkopyCLI, 433
Python.Markopy.MarkovPasswordsCLI, 513, 514
Python.Markopy.ModelMatrixCLI, 645

home
Markov::GUI::Generate, 393
Markov::GUI::MarkovPasswordsGUI, 535
Markov::GUI::Train, 679

Import
Markov::API::CUDA::CUDAModelMatrix, 287, 288
Markov::API::MarkovPasswords, 488, 489
Markov::API::ModelMatrix, 587, 588
Markov::Model< NodeStorageType >, 564, 565
Python.CudaMarkopy.CudaMarkopyCLI, 206–213
Python.CudaMarkopy.CudaModelMatrixCLI, 348–351
Python.Markopy.MarkopyCLI, 434–437
Python.Markopy.MarkovModel, 471–473
Python.Markopy.MarkovPasswordsCLI, 514, 515
Python.Markopy.ModelMatrix, 613, 614
Python.Markopy.ModelMatrixCLI, 645, 646

import_markopy
importer, 26

import_model
Python.CudaMarkopy.CudaMarkopyCLI, 213–218
Python.CudaMarkopy.CudaModelMatrixCLI, 352, 353
Python.Markopy.AbstractGenerationModelCLI, 63

Python.Markopy.AbstractTrainingModelCLI, 81, 82
Python.Markopy.BaseCLI, 108
Python.Markopy.MarkopyCLI, 437–440
Python.Markopy.MarkovPasswordsCLI, 515, 516
Python.Markopy.ModelMatrixCLI, 647

importer, 26
import_markopy, 26

importname
Markov::API::CLI::__programOptions, 53

init_post_arguments
Python.CudaMarkopy.CudaMarkopyCLI, 219
Python.CudaMarkopy.CudaModelMatrixCLI, 354
Python.Markopy.AbstractGenerationModelCLI, 64
Python.Markopy.AbstractTrainingModelCLI, 83
Python.Markopy.BaseCLI, 109
Python.Markopy.MarkopyCLI, 441
Python.Markopy.MarkovPasswordsCLI, 517, 518
Python.Markopy.ModelMatrixCLI, 648

intHandler
markovPasswords.cpp, 717

iterationsPerKernelThread
Markov::API::CUDA::CUDAModelMatrix, 298
Python.CudaMarkopy.CudaMarkopyCLI, 257
Python.CudaMarkopy.CudaModelMatrixCLI, 377

keepRunning
markovPasswords.cpp, 717

LaunchAsyncKernel
Markov::API::CUDA::CUDAModelMatrix, 289
Python.CudaMarkopy.CudaMarkopyCLI, 220
Python.CudaMarkopy.CudaModelMatrixCLI, 354

LeftNode
Markov::Edge< NodeStorageType >, 388

LIGHTGRAY
Markov::API::CLI::Terminal, 672

Link
Markov::Node< storageType >, 663, 664

ListCudaDevices
Markov::API::CUDA::CUDADeviceController, 123
Markov::API::CUDA::CUDAModelMatrix, 289
Markov::API::CUDA::Random::Marsaglia, 544
Python.CudaMarkopy.CudaMarkopyCLI, 220
Python.CudaMarkopy.CudaModelMatrixCLI, 354

listfile
Markov::API::Concurrency::ThreadSharedListHandler, 676

MAGENTA
Markov::API::CLI::Terminal, 672

main
main.cpp, 745, 747
main.cu, 701

main.cpp
main, 745, 747

main.cu
main, 701

markopy, 27
cudamarkopy, 25

cudammx, 26
 ext, 27
 markopy, 27
 mm, 35
 mmx, 36
 mp, 27, 36
 spec, 27
 markopy.cpp
 BOOST_ALL_STATIC_LIB, 715
 BOOST_PYTHON_STATIC_LIB, 715
 Markopy/CudaMarkopy/src/CLI/cudamarkopy.py, 681
 Markopy/CudaMarkopy/src/CLI/cudammx.py, 682, 683
 Markopy/CudaMarkopy/src/Module/cudaMarkopy.cu,
 684, 685
 Markopy/CudaMarkovAPI/src/cudaDeviceController.cu,
 685, 686
 Markopy/CudaMarkovAPI/src/cudaDeviceController.h,
 687, 688
 Markopy/CudaMarkovAPI/src/cudaModelMatrix.cu, 690,
 691
 Markopy/CudaMarkovAPI/src/cudaModelMatrix.h, 694,
 696
 Markopy/CudaMarkovAPI/src/cudarandom.h, 698, 700
 Markopy/CudaMarkovAPI/src/main.cu, 701, 702
 Markopy/documentation/dirs.docs, 702
 Markopy/Markopy/src/CLI/base.py, 703, 704
 Markopy/Markopy/src/CLI/importer.py, 707, 708
 Markopy/Markopy/src/CLI/markopy.py, 708, 709
 Markopy/Markopy/src/CLI/mm.py, 710, 711
 Markopy/Markopy/src/CLI/mmx.py, 711, 712
 Markopy/Markopy/src/CLI/mp.py, 712, 713
 Markopy/Markopy/src/Module/markopy.cpp, 713, 715
 Markopy/MarkovAPI/src/markovPasswords.cpp, 716,
 717
 Markopy/MarkovAPI/src/markovPasswords.h, 720, 721
 Markopy/MarkovAPI/src/modelMatrix.cpp, 722, 723
 Markopy/MarkovAPI/src/modelMatrix.h, 726, 727
 Markopy/MarkovAPI/src/threadSharedListHandler.cpp,
 730, 731
 Markopy/MarkovAPI/src/threadSharedListHandler.h,
 731, 732
 Markopy/MarkovAPICLI/src/argparse.cpp, 734
 Markopy/MarkovAPICLI/src/argparse.h, 735, 736
 Markopy/MarkovAPICLI/src/color/term.cpp, 739, 740
 Markopy/MarkovAPICLI/src/color/term.h, 741, 743
 Markopy/MarkovAPICLI/src/main.cpp, 744, 746
 Markopy/MarkovAPICLI/src/scripts/model_2gram.py,
 748
 Markopy/MarkovAPICLI/src/scripts/random_model.py,
 749
 Markopy/MarkovModel/src/dllmain.cpp, 749, 750
 Markopy/MarkovModel/src/edge.h, 751, 752
 Markopy/MarkovModel/src/framework.h, 754, 755
 Markopy/MarkovModel/src/model.h, 755, 756
 Markopy/MarkovModel/src/node.h, 760, 761
 Markopy/MarkovModel/src/pch.cpp, 765
 Markopy/MarkovModel/src/pch.h, 766, 767
 Markopy/MarkovModel/src/random.h, 768, 770
 Markopy/MarkovPasswordsGUI/src/about.cpp, 772, 773
 Markopy/MarkovPasswordsGUI/src/about.h, 773, 775
 Markopy/MarkovPasswordsGUI/src/CLI.cpp, 775, 776
 Markopy/MarkovPasswordsGUI/src/CLI.h, 776, 777
 Markopy/MarkovPasswordsGUI/src/Generate.cpp, 777,
 778
 Markopy/MarkovPasswordsGUI/src/Generate.h, 780,
 781
 Markopy/MarkovPasswordsGUI/src/main.cpp, 747, 748
 Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.cpp,
 781, 782
 Markopy/MarkovPasswordsGUI/src/MarkovPasswordsGUI.h,
 782, 783
 Markopy/MarkovPasswordsGUI/src/menu.cpp, 784
 Markopy/MarkovPasswordsGUI/src/menu.h, 785, 786
 Markopy/MarkovPasswordsGUI/src/Train.cpp, 786, 787
 Markopy/MarkovPasswordsGUI/src/Train.h, 788, 789
 Markopy/NVSight/report.md, 790
 Markopy/README.md, 790
 Markopy/UnitTests/pch.cpp, 766
 Markopy/UnitTests/pch.h, 768
 Markopy/UnitTests/UnitTests.cpp, 790, 791
 Markov, 27
 Markov::API, 28
 Markov::API::CLI, 28
 operator<<, 29
 ProgramOptions, 29
 Markov::API::CLI::__programOptions, 51
 bExport, 52
 bFailure, 52
 bImport, 52
 datasetname, 52
 exportname, 53
 generateN, 53
 importname, 53
 outputfilename, 53
 seperator, 53
 wordlistname, 53
 Markov::API::CLI::Argparse, 93
 Argparse, 95
 getProgramOptions, 97
 help, 97
 parse, 98
 po, 99
 setProgramOptions, 98
 Markov::API::CLI::Terminal, 670
 BLACK, 672
 BLUE, 672
 BROWN, 672
 color, 672
 colormap, 672
 CYAN, 672
 DARKGRAY, 672
 endl, 673
 GREEN, 672
 LIGHTGRAY, 672
 MAGENTA, 672
 RED, 672

RESET, 672
Terminal, 672
WHITE, 672
YELLOW, 672
Markov::API::Concurrency, 29
Markov::API::Concurrency::ThreadSharedListHandler,
 673
 listfile, 676
 mlock, 676
 next, 676
 ThreadSharedListHandler, 675
Markov::API::CUDA, 29
 FastRandomWalkCUDAKernel, 30
 strchr, 31
Markov::API::CUDA::CUDADeviceController, 117
 CudaCheckNotifyErr, 120
 CudaMalloc2DToFlat, 120
 CudaMemcpy2DToFlat, 121
 CudaMigrate2DFlat, 122
 ListCudaDevices, 123
Markov::API::CUDA::CUDAModelMatrix, 262
 AdjustEdge, 268
 AllocVRAMOutputBuffer, 268
 alternatingKernels, 296
 Buff, 269
 ConstructMatrix, 270
 cudaBlocks, 296
 CudaCheckNotifyErr, 272
 cudaGridSize, 296
 CudaMalloc2DToFlat, 272
 CudaMemcpy2DToFlat, 273
 cudaMemPerGrid, 296
 CudaMigrate2DFlat, 274
 cudaPerKernelAllocationSize, 297
 cudastreams, 297
 cudaThreads, 297
 datasetFile, 297
 DeallocateMatrix, 275
 device_edgeMatrix, 297
 device_matrixIndex, 297
 device_outputBuffer, 297
 device_seeds, 297
 device_totalEdgeWeights, 298
 device_valueMatrix, 298
 DumpJSON, 276
 edgeMatrix, 298
 Edges, 277
 edges, 298
 Export, 277
 FastRandomWalk, 278, 280, 281
 FastRandomWalkPartition, 282
 FastRandomWalkThread, 283
 flatEdgeMatrix, 298
 FlattenMatrix, 284
 flatValueMatrix, 298
 GatherAsyncKernelOutput, 285
 Generate, 285
 GenerateThread, 286
 Import, 287, 288
 iterationsPerKernelThread, 298
 LaunchAsyncKernel, 289
 ListCudaDevices, 289
 matrixIndex, 298
 matrixSize, 299
 MigrateMatrix, 290
 modelSavefile, 299
 Nodes, 290
 nodes, 299
 numberOfPartitions, 299
 OpenDatasetFile, 290
 OptimizeEdgeOrder, 291
 outputBuffer, 299
 outputFile, 299
 prepKernelMemoryChannel, 291
 RandomWalk, 292
 ready, 299
 Save, 293
 StarterNode, 294
 starterNode, 299
 totalEdgeWeights, 300
 totalOutputPerKernel, 300
 totalOutputPerSync, 300
 Train, 294
 TrainThread, 295
 valueMatrix, 300
Markov::API::CUDA::Random, 32
 devrandom, 32
Markov::API::CUDA::Random::Marsaglia, 537
 CudaCheckNotifyErr, 539
 CudaMalloc2DToFlat, 540
 CudaMemcpy2DToFlat, 541
 CudaMigrate2DFlat, 542
 distribution, 543
 generator, 543
 ListCudaDevices, 544
 MigrateToVRAM, 544
 random, 545
 rd, 546
 x, 546
 y, 546
 z, 546
Markov::API::MarkovPasswords, 479
 AdjustEdge, 483
 Buff, 484
 datasetFile, 495
 Edges, 485
 edges, 495
 Export, 486
 Generate, 486
 GenerateThread, 487
 Import, 488, 489
 MarkovPasswords, 483
 modelSavefile, 495
 Nodes, 490
 nodes, 495
 OpenDatasetFile, 490

OptimizeEdgeOrder, 490
 outputFile, 495
 RandomWalk, 491
 Save, 492
 StarterNode, 492
 starterNode, 495
 Train, 493
 TrainThread, 494
 Markov::API::ModelMatrix, 569
 AdjustEdge, 574
 Buff, 574
 ConstructMatrix, 576
 datasetFile, 594
 DeallocateMatrix, 577
 DumpJSON, 578
 edgeMatrix, 594
 Edges, 579
 edges, 594
 Export, 579
 FastRandomWalk, 580, 581
 FastRandomWalkPartition, 582
 FastRandomWalkThread, 583
 Generate, 585
 GenerateThread, 586
 Import, 587, 588
 matrixIndex, 594
 matrixSize, 594
 ModelMatrix, 573
 modelSavefile, 594
 Nodes, 589
 nodes, 594
 OpenDatasetFile, 589
 OptimizeEdgeOrder, 589
 outputFile, 595
 RandomWalk, 590
 ready, 595
 Save, 591
 StarterNode, 591
 starterNode, 595
 totalEdgeWeights, 595
 Train, 592
 TrainThread, 593
 valueMatrix, 595
 Markov::Edge< NodeStorageType >, 385
 _left, 389
 _right, 389
 _weight, 390
 AdjustEdge, 387
 Edge, 386
 EdgeWeight, 387
 LeftNode, 388
 RightNode, 388
 SetLeftEdge, 388
 SetRightEdge, 389
 TraverseNode, 389
 Markov::GUI, 32
 Markov::GUI::about, 54
 about, 55
 ui, 55
 Markov::GUI::CLI, 114
 about, 116
 CLI, 115
 start, 116
 statistics, 117
 ui, 117
 Markov::GUI::Generate, 390
 Generate, 391
 generation, 392
 home, 393
 train, 393
 ui, 394
 vis, 394
 Markov::GUI::MarkovPasswordsGUI, 533
 benchmarkSelected, 535
 home, 535
 MarkovPasswordsGUI, 535
 model, 535
 pass, 536
 ui, 536
 Markov::GUI::menu, 552
 about, 554
 menu, 553
 ui, 554
 visualization, 554
 Markov::GUI::Train, 677
 home, 679
 Train, 678
 train, 679
 ui, 680
 Markov::Markopy, 33
 BOOST_PYTHON_MODULE, 33
 Markov::Markopy::CUDA, 34
 BOOST_PYTHON_MODULE, 34
 Markov::Model< NodeStorageType >, 559
 AdjustEdge, 562
 Edges, 562
 edges, 568
 Export, 563
 Import, 564, 565
 Model, 561
 Nodes, 566
 nodes, 569
 OptimizeEdgeOrder, 566
 RandomWalk, 567
 StarterNode, 568
 starterNode, 569
 Markov::Node< storageType >, 660
 _value, 667
 Edges, 662
 edges, 667
 edgesV, 667
 FindEdge, 662, 663
 Link, 663, 664
 Node, 661, 662
 NodeValue, 664
 RandomNext, 664

total_edge_weights, 667
TotalEdgeWeights, 665
UpdateEdges, 666
UpdateTotalVerticeWeight, 666
Markov::Random, 35
Markov::Random::DefaultRandomEngine, 380
 distribution, 383
 generator, 383
 random, 384
 rd, 384
Markov::Random::Marsaglia, 547
 distribution, 549
 generator, 550
 Marsaglia, 549
 random, 550
 rd, 551
 x, 551
 y, 551
 z, 551
Markov::Random::Mersenne, 555
 distribution, 557
 generator, 557
 random, 558
 rd, 558
Markov::Random::RandomEngine, 668
 random, 670
MarkovPasswords
 Markov::API::MarkovPasswords, 483
markovPasswords.cpp
 intHandler, 717
 keepRunning, 717
MarkovPasswordsGUI
 Markov::GUI::MarkovPasswordsGUI, 535
Marsaglia
 Markov::Random::Marsaglia, 549
matrixIndex
 Markov::API::CUDA::CUDAModelMatrix, 298
 Markov::API::ModelMatrix, 594
 Python.CudaMarkopy.CudaMarkopyCLI, 257
 Python.CudaMarkopy.CudaModelMatrixCLI, 377
 Python.Markopy.MarkopyCLI, 459
 Python.Markopy.ModelMatrix, 620
 Python.Markopy.ModelMatrixCLI, 658
matrixSize
 Markov::API::CUDA::CUDAModelMatrix, 299
 Markov::API::ModelMatrix, 594
 Python.CudaMarkopy.CudaMarkopyCLI, 257
 Python.CudaMarkopy.CudaModelMatrixCLI, 377
 Python.Markopy.MarkopyCLI, 459
 Python.Markopy.ModelMatrix, 621
 Python.Markopy.ModelMatrixCLI, 658
menu
 Markov::GUI::menu, 553
MigrateMatrix
 Markov::API::CUDA::CUDAModelMatrix, 290
 Python.CudaMarkopy.CudaMarkopyCLI, 220
 Python.CudaMarkopy.CudaModelMatrixCLI, 355
MigrateToVRAM
 Markov::API::CUDA::Random::Marsaglia, 544
mlock
 Markov::API::Concurrency::ThreadSharedListHandler, 676
mm, 35
 markopy, 35
mmx, 35
 markopy, 36
 mp, 36
Model
 Markov::Model< NodeStorageType >, 561
model
 Markov::GUI::MarkovPasswordsGUI, 535
 Python.CudaMarkopy.CudaMarkopyCLI, 258
 Python.CudaMarkopy.CudaModelMatrixCLI, 377
 Python.Markopy.AbstractGenerationModelCLI, 69
 Python.Markopy.AbstractTrainingModelCLI, 92
 Python.Markopy.BaseCLI, 113
 Python.Markopy.MarkopyCLI, 459
 Python.Markopy.MarkovPasswordsCLI, 532
 Python.Markopy.ModelMatrixCLI, 658
model_2gram, 36
 alphabet, 36
 f, 36
ModelMatrix
 Markov::API::ModelMatrix, 573
modelSavefile
 Markov::API::CUDA::CUDAModelMatrix, 299
 Markov::API::MarkovPasswords, 495
 Markov::API::ModelMatrix, 594
 Python.CudaMarkopy.CudaMarkopyCLI, 258
 Python.CudaMarkopy.CudaModelMatrixCLI, 378
 Python.Markopy.MarkopyCLI, 460
 Python.Markopy.MarkovModel, 478
 Python.Markopy.MarkovPasswordsCLI, 532
 Python.Markopy.ModelMatrix, 621
 Python.Markopy.ModelMatrixCLI, 658
mp, 36
 cudamarkopy, 25
 cudammx, 26
 markopy, 27, 36
 mmx, 36
 mp, 37
next
 Markov::API::Concurrency::ThreadSharedListHandler, 676
Node
 Markov::Node< storageType >, 661, 662
Nodes
 Markov::API::CUDA::CUDAModelMatrix, 290
 Markov::API::MarkovPasswords, 490
 Markov::API::ModelMatrix, 589
 Markov::Model< NodeStorageType >, 566
 Python.CudaMarkopy.CudaMarkopyCLI, 221
 Python.CudaMarkopy.CudaModelMatrixCLI, 355
 Python.Markopy.MarkopyCLI, 441
 Python.Markopy.MarkovModel, 473
 Python.Markopy.MarkovPasswordsCLI, 518

Python.Markopy.ModelMatrix, 615
 Python.Markopy.ModelMatrixCLI, 649

nodes
 Markov::API::CUDA::CUDAModelMatrix, 299
 Markov::API::MarkovPasswords, 495
 Markov::API::ModelMatrix, 594
 Markov::Model< NodeStorageType >, 569
 Python.CudaMarkopy.CudaMarkopyCLI, 258
 Python.CudaMarkopy.CudaModelMatrixCLI, 378
 Python.Markopy.MarkopyCLI, 460
 Python.Markopy.MarkovModel, 478
 Python.Markopy.MarkovPasswordsCLI, 532
 Python.Markopy.ModelMatrix, 621
 Python.Markopy.ModelMatrixCLI, 658

nodeValue
 Markov::Node< storageType >, 664

numberOfPartitions
 Markov::API::CUDA::CUDAModelMatrix, 299
 Python.CudaMarkopy.CudaMarkopyCLI, 258
 Python.CudaMarkopy.CudaModelMatrixCLI, 378

OpenDatasetFile
 Markov::API::CUDA::CUDAModelMatrix, 290
 Markov::API::MarkovPasswords, 490
 Markov::API::ModelMatrix, 589
 Python.CudaMarkopy.CudaMarkopyCLI, 222–224
 Python.CudaMarkopy.CudaModelMatrixCLI, 356
 Python.Markopy.MarkopyCLI, 442
 Python.Markopy.MarkovModel, 473
 Python.Markopy.MarkovPasswordsCLI, 518
 Python.Markopy.ModelMatrix, 615
 Python.Markopy.ModelMatrixCLI, 649

operator<<
 Markov::API::CLI, 29
 term.cpp, 740

OptimizeEdgeOrder
 Markov::API::CUDA::CUDAModelMatrix, 291
 Markov::API::MarkovPasswords, 490
 Markov::API::ModelMatrix, 589
 Markov::Model< NodeStorageType >, 566
 Python.CudaMarkopy.CudaMarkopyCLI, 224, 225
 Python.CudaMarkopy.CudaModelMatrixCLI, 357
 Python.Markopy.MarkopyCLI, 443
 Python.Markopy.MarkovModel, 473
 Python.Markopy.MarkovPasswordsCLI, 519
 Python.Markopy.ModelMatrix, 616
 Python.Markopy.ModelMatrixCLI, 649

outputBuffer
 Markov::API::CUDA::CUDAModelMatrix, 299
 Python.CudaMarkopy.CudaMarkopyCLI, 259
 Python.CudaMarkopy.CudaModelMatrixCLI, 378

outputFile
 Markov::API::CUDA::CUDAModelMatrix, 299
 Markov::API::MarkovPasswords, 495
 Markov::API::ModelMatrix, 595
 Python.CudaMarkopy.CudaMarkopyCLI, 259
 Python.CudaMarkopy.CudaModelMatrixCLI, 378
 Python.Markopy.MarkopyCLI, 460
 Python.Markopy.MarkovModel, 478

Python.Markopy.MarkovPasswordsCLI, 532
 Python.Markopy.ModelMatrix, 621
 Python.Markopy.ModelMatrixCLI, 659

outputfilename
 Markov::API::CLI::__programOptions, 53

parse
 Markov::API::CLI::Argparse, 98
 Python.CudaMarkopy.CudaMarkopyCLI, 225
 Python.CudaMarkopy.CudaModelMatrixCLI, 357, 358
 Python.Markopy.AbstractGenerationModelCLI, 65
 Python.Markopy.AbstractTrainingModelCLI, 84, 85
 Python.Markopy.BaseCLI, 109
 Python.Markopy.MarkopyCLI, 443
 Python.Markopy.MarkovPasswordsCLI, 519, 520
 Python.Markopy.ModelMatrixCLI, 650

parse_arguments
 Python.CudaMarkopy.CudaMarkopyCLI, 225–227
 Python.CudaMarkopy.CudaModelMatrixCLI, 359
 Python.Markopy.AbstractGenerationModelCLI, 65
 Python.Markopy.AbstractTrainingModelCLI, 85, 86
 Python.Markopy.BaseCLI, 110
 Python.Markopy.MarkopyCLI, 444, 445
 Python.Markopy.MarkovPasswordsCLI, 521
 Python.Markopy.ModelMatrixCLI, 650

parse_fail
 Python.CudaMarkopy.CudaMarkopyCLI, 228
 Python.Markopy.MarkopyCLI, 446

parser
 Python.CudaMarkopy.CudaMarkopyCLI, 259, 260
 Python.CudaMarkopy.CudaModelMatrixCLI, 378
 Python.Markopy.AbstractGenerationModelCLI, 69
 Python.Markopy.AbstractTrainingModelCLI, 92
 Python.Markopy.BaseCLI, 113
 Python.Markopy.MarkopyCLI, 460, 461
 Python.Markopy.MarkovPasswordsCLI, 532
 Python.Markopy.ModelMatrixCLI, 659

pass
 Markov::GUI::MarkovPasswordsGUI, 536

po
 Markov::API::CLI::Argparse, 99

prepKernelMemoryChannel
 Markov::API::CUDA::CUDAModelMatrix, 291
 Python.CudaMarkopy.CudaMarkopyCLI, 228
 Python.CudaMarkopy.CudaModelMatrixCLI, 360

print_help
 Python.CudaMarkopy.CudaMarkopyCLI, 260
 Python.CudaMarkopy.CudaModelMatrixCLI, 379
 Python.Markopy.AbstractGenerationModelCLI, 69
 Python.Markopy.AbstractTrainingModelCLI, 92
 Python.Markopy.BaseCLI, 113
 Python.Markopy.MarkopyCLI, 461
 Python.Markopy.MarkovPasswordsCLI, 533
 Python.Markopy.ModelMatrixCLI, 659

process
 Python.CudaMarkopy.CudaMarkopyCLI, 229
 Python.CudaMarkopy.CudaModelMatrixCLI, 360, 362

Python.Markopy.AbstractGenerationModelCLI, 66
Python.Markopy.AbstractTrainingModelCLI, 86, 87
Python.Markopy.BaseCLI, 110
Python.Markopy.MarkopyCLI, 446
Python.Markopy.MarkovPasswordsCLI, 522, 523
Python.Markopy.ModelMatrixCLI, 651
ProgramOptions
 Markov::API::CLI, 29
Python.CudaMarkopy, 37
Python.CudaMarkopy.CudaMarkopyCLI, 124
 __init__, 134
 _generate, 134
 add_arguments, 135, 136
 AdjustEdge, 136, 138, 139
 AllocVRAMOutputBuffer, 140
 alternatingKernels, 253
 args, 254
 bInfinite, 254
 Buff, 140, 141, 143, 144
 check_corpus_path, 145–148
 check_export_path, 148–151
 check_import_path, 151–154
 cli, 254
 ConstructMatrix, 154, 156, 158
 cudaBlocks, 254
 CudaCheckNotifyErr, 159
 cudaGridSize, 254
 CudaMalloc2DToFlat, 160
 CudaMemcpy2DToFlat, 161
 cudaMemPerGrid, 254
 CudaMigrate2DFlat, 162
 cudaPerKernelAllocationSize, 254
 cudastreams, 254
 cudaThreads, 255
 datasetFile, 255
 DeallocateMatrix, 163, 164
 device_edgeMatrix, 255
 device_matrixIndex, 255
 device_outputBuffer, 255
 device_seeds, 255
 device_totalEdgeWeights, 255
 device_valueMatrix, 255
 DumpJSON, 165–167
 edgeMatrix, 256
 Edges, 168, 169
 edges, 256
 Export, 169, 170, 174–176
 export, 170–174
 FastRandomWalk, 176, 177, 179–184
 FastRandomWalkPartition, 185–187
 FastRandomWalkThread, 188, 190, 191
 fileIO, 256
 flatEdgeMatrix, 256
 FlattenMatrix, 193
 flatValueMatrix, 256
 GatherAsyncKernelOutput, 193
 Generate, 193, 200–203
 generate, 194–199
 GenerateThread, 205
 help, 205
 Import, 206–213
 import_model, 213–218
 init_post_arguments, 219
 iterationsPerKernelThread, 257
 LaunchAsyncKernel, 220
 ListCudaDevices, 220
 matrixIndex, 257
 matrixSize, 257
 MigrateMatrix, 220
 model, 258
 modelSavefile, 258
 Nodes, 221
 nodes, 258
 numberOfPartitions, 258
 OpenDatasetFile, 222–224
 OptimizeEdgeOrder, 224, 225
 outputBuffer, 259
 outputFile, 259
 parse, 225
 parse_arguments, 225–227
 parse_fail, 228
 parser, 259, 260
 prepKernelMemoryChannel, 228
 print_help, 260
 process, 229
 RandomWalk, 229, 230, 232, 234
 ready, 261
 Save, 236–238
 StarterNode, 238, 239
 starterNode, 261
 stub, 239
 totalEdgeWeights, 261
 totalOutputPerKernel, 262
 totalOutputPerSync, 262
 Train, 240–242, 252
 train, 243–245, 247, 248, 251
 TrainThread, 252
 valueMatrix, 262
Python.CudaMarkopy.CudaModelMatrixCLI, 300
 __init__, 308
 _generate, 308
 add_arguments, 309
 AdjustEdge, 309, 310
 AllocVRAMOutputBuffer, 311
 alternatingKernels, 374
 args, 374
 bInfinite, 374
 Buff, 311, 312
 check_corpus_path, 313, 314
 check_export_path, 314, 315
 check_import_path, 315, 316
 ConstructMatrix, 316, 318
 cudaBlocks, 374
 CudaCheckNotifyErr, 320
 cudaGridSize, 374
 CudaMalloc2DToFlat, 320

CudaMemcpy2DToFlat, 321
 cudaMemcpyPerGrid, 375
 CudaMigrate2DFlat, 322
 cudaPerKernelAllocationSize, 375
 cudastreams, 375
 cudaThreads, 375
 datasetFile, 375
 DeallocateMatrix, 323, 324
 device_edgeMatrix, 375
 device_matrixIndex, 375
 device_outputBuffer, 375
 device_seeds, 376
 device_totalEdgeWeights, 376
 device_valueMatrix, 376
 DumpJSON, 325, 326
 edgeMatrix, 376
 Edges, 327
 edges, 376
 Export, 327, 329, 330
 export, 328
 FastRandomWalk, 330, 332–335
 FastRandomWalkPartition, 336, 337
 FastRandomWalkThread, 339, 340
 fileIO, 376
 flatEdgeMatrix, 376
 FlattenMatrix, 342
 flatValueMatrix, 377
 GatherAsyncKernelOutput, 342
 Generate, 344, 345
 generate, 342, 343
 GenerateThread, 346
 help, 347, 348
 Import, 348–351
 import_model, 352, 353
 init_post_arguments, 354
 iterationsPerKernelThread, 377
 LaunchAsyncKernel, 354
 ListCudaDevices, 354
 matrixIndex, 377
 matrixSize, 377
 MigrateMatrix, 355
 model, 377
 modelSavefile, 378
 Nodes, 355
 nodes, 378
 numberOfPartitions, 378
 OpenDatasetFile, 356
 OptimizeEdgeOrder, 357
 outputBuffer, 378
 outputFile, 378
 parse, 357, 358
 parse_arguments, 359
 parser, 378
 prepKernelMemoryChannel, 360
 print_help, 379
 process, 360, 362
 RandomWalk, 363, 364
 ready, 379
 Save, 366, 367
 StarterNode, 367
 starterNode, 379
 totalEdgeWeights, 379
 totalOutputPerKernel, 380
 totalOutputPerSync, 380
 Train, 368, 369
 train, 370, 371
 TrainThread, 373
 valueMatrix, 380
 Python.Markopy, 37
 Python.Markopy.AbstractGenerationModelCLI, 55
 _generate, 58
 add_arguments, 59
 args, 68
 check_corpus_path, 60
 check_export_path, 60
 check_import_path, 61
 export, 61
 generate, 62
 help, 63
 import_model, 63
 init_post_arguments, 64
 model, 69
 parse, 65
 parse_arguments, 65
 parser, 69
 print_help, 69
 process, 66
 train, 67
 Python.Markopy.AbstractTrainingModelCLI, 69
 _generate, 73
 add_arguments, 73
 args, 91, 92
 check_corpus_path, 74
 check_export_path, 75
 check_import_path, 76
 export, 77, 78
 generate, 78, 79
 help, 80, 81
 import_model, 81, 82
 init_post_arguments, 83
 model, 92
 parse, 84, 85
 parse_arguments, 85, 86
 parser, 92
 print_help, 92
 process, 86, 87
 train, 89, 90
 Python.Markopy.BaseCLI, 99
 __init__, 102
 _generate, 103
 add_arguments, 104
 args, 113
 check_corpus_path, 104
 check_export_path, 105
 check_import_path, 105
 export, 106

generate, 106
help, 107
import_model, 108
init_post_arguments, 109
model, 113
parse, 109
parse_arguments, 110
parser, 113
print_help, 113
process, 110
train, 111
Python.Markopy.MarkopyCLI, 394
 __init__, 401
 _generate, 401
 add_arguments, 402
 AdjustEdge, 403
 args, 458
 Buff, 404, 405
 check_corpus_path, 407, 408
 check_export_path, 409, 410
 check_import_path, 411, 412
 cli, 458
 ConstructMatrix, 413
 datasetFile, 458
 DeallocateMatrix, 414
 DumpJSON, 415
 edgeMatrix, 458
 Edges, 416
 edges, 459
 Export, 416, 417, 420, 421
 export, 417–419
 FastRandomWalk, 421, 422
 FastRandomWalkPartition, 423
 FastRandomWalkThread, 424
 fileIO, 459
 Generate, 426, 430, 431
 generate, 426–429
 GenerateThread, 432
 help, 433
 Import, 434–437
 import_model, 437–440
 init_post_arguments, 441
 matrixIndex, 459
 matrixSize, 459
 model, 459
 modelSavefile, 460
 Nodes, 441
 nodes, 460
 OpenDatasetFile, 442
 OptimizeEdgeOrder, 443
 outputFile, 460
 parse, 443
 parse_arguments, 444, 445
 parse_fail, 446
 parser, 460, 461
 print_help, 461
 process, 446
 RandomWalk, 446, 447
 ready, 461
 Save, 448, 449
 StarterNode, 449, 450
 starterNode, 461
 stub, 450
 totalEdgeWeights, 461
 Train, 450, 457
 train, 451, 453, 454, 456
 TrainThread, 457
 valueMatrix, 462
Python.Markopy.MarkovModel, 462
 AdjustEdge, 466
 Buff, 466
 datasetFile, 478
 Edges, 468
 edges, 478
 Export, 468, 469
 Generate, 469
 GenerateThread, 470
 Import, 471–473
 modelSavefile, 478
 Nodes, 473
 nodes, 478
 OpenDatasetFile, 473
 OptimizeEdgeOrder, 473
 outputFile, 478
 RandomWalk, 474
 Save, 475
 StarterNode, 475
 starterNode, 479
 Train, 476, 477
 TrainThread, 477
Python.Markopy.MarkovPasswordsCLI, 496
 __init__, 500
 _generate, 500
 add_arguments, 501
 AdjustEdge, 501
 args, 531
 Buff, 502
 check_corpus_path, 503, 504
 check_export_path, 504, 505
 check_import_path, 505, 506
 datasetFile, 531
 Edges, 506
 edges, 532
 Export, 506, 508, 509
 export, 507
 Generate, 509, 511
 generate, 509, 510
 GenerateThread, 512
 help, 513, 514
 Import, 514, 515
 import_model, 515, 516
 init_post_arguments, 517, 518
 model, 532
 modelSavefile, 532
 Nodes, 518
 nodes, 532

OpenDatasetFile, 518
 OptimizeEdgeOrder, 519
 outputFile, 532
 parse, 519, 520
 parse_arguments, 521
 parser, 532
 print_help, 533
 process, 522, 523
 RandomWalk, 524
 Save, 525
 StarterNode, 526
 starterNode, 533
 Train, 526, 530
 train, 527, 529
 TrainThread, 530
Python.Markopy.ModelMatrix, 595
 AdjustEdge, 599
 Buff, 600
 ConstructMatrix, 601
 datasetFile, 620
 DeallocateMatrix, 603
 DumpJSON, 604
 edgeMatrix, 620
 Edges, 605
 edges, 620
 Export, 605
 FastRandomWalk, 606, 607
 FastRandomWalkPartition, 608
 FastRandomWalkThread, 609
 Generate, 611
 GenerateThread, 612
 Import, 613, 614
 matrixIndex, 620
 matrixSize, 621
 modelSavefile, 621
 Nodes, 615
 nodes, 621
 OpenDatasetFile, 615
 OptimizeEdgeOrder, 616
 outputFile, 621
 RandomWalk, 616
 ready, 621
 Save, 617
 StarterNode, 618
 starterNode, 621
 totalEdgeWeights, 621
 Train, 618
 TrainThread, 619
 valueMatrix, 622
Python.Markopy.ModelMatrixCLI, 622
 __init__, 627
 _generate, 627
 add_arguments, 628
 AdjustEdge, 628
 args, 657
 Buff, 629
 check_corpus_path, 630
 check_export_path, 631
 check_import_path, 631
 ConstructMatrix, 632
 datasetFile, 657
 DeallocateMatrix, 633
 DumpJSON, 634
 edgeMatrix, 657
 Edges, 635
 edges, 658
 Export, 635, 636
 export, 636
 FastRandomWalk, 637, 638
 FastRandomWalkPartition, 639
 FastRandomWalkThread, 640
 fileIO, 658
 Generate, 643
 generate, 642
 GenerateThread, 644
 help, 645
 Import, 645, 646
 import_model, 647
 init_post_arguments, 648
 matrixIndex, 658
 matrixSize, 658
 model, 658
 modelSavefile, 658
 Nodes, 649
 nodes, 658
 OpenDatasetFile, 649
 OptimizeEdgeOrder, 649
 outputFile, 659
 parse, 650
 parse_arguments, 650
 parser, 659
 print_help, 659
 process, 651
 RandomWalk, 652
 ready, 659
 Save, 653
 StarterNode, 654
 starterNode, 659
 totalEdgeWeights, 659
 Train, 654
 train, 655
 TrainThread, 656
 valueMatrix, 659
QMainWindow, 667
random
 Markov::API::CUDA::Random::Marsaglia, 545
 Markov::Random::DefaultRandomEngine, 384
 Markov::Random::Marsaglia, 550
 Markov::Random::Mersenne, 558
 Markov::Random::RandomEngine, 670
 random_model, 38
 alphabet, 38
 f, 38
RandomNext
 Markov::Node<storageType>, 664

RandomWalk
Markov::API::CUDA::CUDAModelMatrix, 292
Markov::API::MarkovPasswords, 491
Markov::API::ModelMatrix, 590
Markov::Model< NodeStorageType >, 567
Python.CudaMarkopy.CudaMarkopyCLI, 229, 230, 232, 234
Python.CudaMarkopy.CudaModelMatrixCLI, 363, 364
Python.Markopy.MarkopyCLI, 446, 447
Python.Markopy.MarkovModel, 474
Python.Markopy.MarkovPasswordsCLI, 524
Python.Markopy.ModelMatrix, 616
Python.Markopy.ModelMatrixCLI, 652

rd
Markov::API::CUDA::Random::Marsaglia, 546
Markov::Random::DefaultRandomEngine, 384
Markov::Random::Marsaglia, 551
Markov::Random::Mersenne, 558

ready
Markov::API::CUDA::CUDAModelMatrix, 299
Markov::API::ModelMatrix, 595
Python.CudaMarkopy.CudaMarkopyCLI, 261
Python.CudaMarkopy.CudaModelMatrixCLI, 379
Python.Markopy.MarkopyCLI, 461
Python.Markopy.ModelMatrix, 621
Python.Markopy.ModelMatrixCLI, 659

RED
Markov::API::CLI::Terminal, 672

RESET
Markov::API::CLI::Terminal, 672

RightNode
Markov::Edge< NodeStorageType >, 388

Save
Markov::API::CUDA::CUDAModelMatrix, 293
Markov::API::MarkovPasswords, 492
Markov::API::ModelMatrix, 591
Python.CudaMarkopy.CudaMarkopyCLI, 236–238
Python.CudaMarkopy.CudaModelMatrixCLI, 366, 367
Python.Markopy.MarkopyCLI, 448, 449
Python.Markopy.MarkovModel, 475
Python.Markopy.MarkovPasswordsCLI, 525
Python.Markopy.ModelMatrix, 617
Python.Markopy.ModelMatrixCLI, 653

seperator
Markov::API::CLI::__programOptions, 53

SetLeftEdge
Markov::Edge< NodeStorageType >, 388

setProgramOptions
Markov::API::CLI::Argparse, 98

SetRightEdge
Markov::Edge< NodeStorageType >, 389

spec
cudamarkopy, 25
cudammx, 26
markopy, 27

start
Markov::GUI::CLI, 116

StarterNode
Markov::API::CUDA::CUDAModelMatrix, 294
Markov::API::MarkovPasswords, 492
Markov::API::ModelMatrix, 591
Markov::Model< NodeStorageType >, 568
Python.CudaMarkopy.CudaMarkopyCLI, 238, 239
Python.CudaMarkopy.CudaModelMatrixCLI, 367
Python.Markopy.MarkopyCLI, 449, 450
Python.Markopy.MarkovModel, 475
Python.Markopy.MarkovPasswordsCLI, 526
Python.Markopy.ModelMatrix, 618
Python.Markopy.ModelMatrixCLI, 654

starterNode
Markov::API::CUDA::CUDAModelMatrix, 299
Markov::API::MarkovPasswords, 495
Markov::API::ModelMatrix, 595
Markov::Model< NodeStorageType >, 569
Python.CudaMarkopy.CudaMarkopyCLI, 261
Python.CudaMarkopy.CudaModelMatrixCLI, 379
Python.Markopy.MarkopyCLI, 461
Python.Markopy.MarkovModel, 479
Python.Markopy.MarkovPasswordsCLI, 533
Python.Markopy.ModelMatrix, 621
Python.Markopy.ModelMatrixCLI, 659

statistics
Markov::GUI::CLI, 117

strchr
Markov::API::CUDA, 31

stub
Python.CudaMarkopy.CudaMarkopyCLI, 239
Python.Markopy.MarkopyCLI, 450

term.cpp
operator<<, 740

term.h
TERM_FAIL, 743
TERM_INFO, 743
TERM_SUCC, 743
TERM_WARN, 743

TERM_FAIL
term.h, 743

TERM_INFO
term.h, 743

TERM_SUCC
term.h, 743

TERM_WARN
term.h, 743

Terminal
Markov::API::CLI::Terminal, 672

TEST_CLASS
Testing::MarkovModel, 39–41
Testing::MVP::MarkovModel, 43–45
Testing::MVP::MarkovPasswords, 48

Testing, 38
Testing::MarkovModel, 38
TEST_CLASS, 39–41
Testing::MarkovPasswords, 42
Testing::MVP, 42

Testing::MVP::MarkovModel, 42
 TEST_CLASS, 43–45

Testing::MVP::MarkovPasswords, 48
 TEST_CLASS, 48

ThreadSharedListHandler
 Markov::API::Concurrency::ThreadSharedListHandler, 675

total_edge_weights
 Markov::Node< storageType >, 667

TotalEdgeWeights
 Markov::Node< storageType >, 665

totalEdgeWeights
 Markov::API::CUDA::CUDAModelMatrix, 300
 Markov::API::ModelMatrix, 595
 Python.CudaMarkopy.CudaMarkopyCLI, 261
 Python.CudaMarkopy.CudaModelMatrixCLI, 379
 Python.Markopy.MarkopyCLI, 461
 Python.Markopy.ModelMatrix, 621
 Python.Markopy.ModelMatrixCLI, 659

totalOutputPerKernel
 Markov::API::CUDA::CUDAModelMatrix, 300
 Python.CudaMarkopy.CudaMarkopyCLI, 262
 Python.CudaMarkopy.CudaModelMatrixCLI, 380

totalOutputPerSync
 Markov::API::CUDA::CUDAModelMatrix, 300
 Python.CudaMarkopy.CudaMarkopyCLI, 262
 Python.CudaMarkopy.CudaModelMatrixCLI, 380

Train
 Markov::API::CUDA::CUDAModelMatrix, 294
 Markov::API::MarkovPasswords, 493
 Markov::API::ModelMatrix, 592
 Markov::GUI::Train, 678
 Python.CudaMarkopy.CudaMarkopyCLI, 240–242, 252
 Python.CudaMarkopy.CudaModelMatrixCLI, 368, 369
 Python.Markopy.MarkopyCLI, 450, 457
 Python.Markopy.ModelMatrix, 476, 477
 Python.Markopy.MarkovPasswordsCLI, 526, 530
 Python.Markopy.ModelMatrix, 618
 Python.Markopy.ModelMatrixCLI, 654

train
 Markov::GUI::Generate, 393
 Markov::GUI::Train, 679
 Python.CudaMarkopy.CudaMarkopyCLI, 243–245, 247, 248, 251
 Python.CudaMarkopy.CudaModelMatrixCLI, 370, 371
 Python.Markopy.AbstractGenerationModelCLI, 67
 Python.Markopy.AbstractTrainingModelCLI, 89, 90
 Python.Markopy.BaseCLI, 111
 Python.Markopy.MarkopyCLI, 451, 453, 454, 456
 Python.Markopy.MarkovPasswordsCLI, 527, 529
 Python.Markopy.ModelMatrixCLI, 655

TrainThread
 Markov::API::CUDA::CUDAModelMatrix, 295
 Markov::API::MarkovPasswords, 494
 Markov::API::ModelMatrix, 593

Python.CudaMarkopy.CudaMarkopyCLI, 252
 Python.CudaMarkopy.CudaModelMatrixCLI, 373
 Python.Markopy.MarkopyCLI, 457
 Python.Markopy.MarkovModel, 477
 Python.Markopy.MarkovPasswordsCLI, 530
 Python.Markopy.ModelMatrix, 619
 Python.Markopy.ModelMatrixCLI, 656

TraverseNode
 Markov::Edge< NodeStorageType >, 389

ui
 Markov::GUI::about, 55
 Markov::GUI::CLI, 117
 Markov::GUI::Generate, 394
 Markov::GUI::MarkovPasswordsGUI, 536
 Markov::GUI::menu, 554
 Markov::GUI::Train, 680

UpdateEdges
 Markov::Node< storageType >, 666

UpdateTotalVerticeWeight
 Markov::Node< storageType >, 666

valueMatrix
 Markov::API::CUDA::CUDAModelMatrix, 300
 Markov::API::ModelMatrix, 595
 Python.CudaMarkopy.CudaMarkopyCLI, 262
 Python.CudaMarkopy.CudaModelMatrixCLI, 380
 Python.Markopy.MarkopyCLI, 462
 Python.Markopy.ModelMatrix, 622
 Python.Markopy.ModelMatrixCLI, 659

vis
 Markov::GUI::Generate, 394

visualization
 Markov::GUI::menu, 554

WHITE
 Markov::API::CLI::Terminal, 672

WIN32_LEAN_AND_MEAN
 framework.h, 754

wordlistname
 Markov::API::CLI::__programOptions, 53

x
 Markov::API::CUDA::Random::Marsaglia, 546
 Markov::Random::Marsaglia, 551

y
 Markov::API::CUDA::Random::Marsaglia, 546
 Markov::Random::Marsaglia, 551

YELLOW
 Markov::API::CLI::Terminal, 672

z
 Markov::API::CUDA::Random::Marsaglia, 546
 Markov::Random::Marsaglia, 551