



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

**0.6.0 Documentation**

---

<b>1 Markov Passwords</b>	<b>3</b>
1.1 About The Project . . . . .	3
1.1.1 Built With . . . . .	4
1.2 Getting Started . . . . .	4
1.2.1 Prerequisites . . . . .	4
1.2.2 Installing Dependencies . . . . .	4
1.2.3 Installation . . . . .	4
1.2.4 Building . . . . .	4
1.3 Linux . . . . .	5
1.4 Windows . . . . .	5
1.5 Known Common issues . . . . .	5
1.5.1 Linux . . . . .	5
1.5.1.1 Markopy - Python.h - Not found . . . . .	5
1.5.1.2 Markopy/MarkovPasswords - *.so not found, or other library related issues when building . . . . .	5
1.5.2 Windows . . . . .	5
1.5.2.1 Boost - Bootstrap.bat "ctype.h" not found . . . . .	5
1.5.2.2 Cannot open file "*.lib" . . . . .	5
1.5.2.3 Python.h not found . . . . .	5
1.5.2.4 Simplified Theory . . . . .	6
1.5.3 Contributing . . . . .	6
1.5.4 Contact . . . . .	6
<b>2 NVSight Report</b>	<b>7</b>
2.0.1 Overview . . . . .	7
2.0.2 Session . . . . .	12
2.0.3 PTX . . . . .	14
<b>3 Deprecated List</b>	<b>17</b>
<b>4 Namespace Index</b>	<b>19</b>
4.1 Namespace List . . . . .	19
<b>5 Hierarchical Index</b>	<b>21</b>
5.1 Class Hierarchy . . . . .	21
<b>6 Class Index</b>	<b>23</b>
6.1 Class List . . . . .	23
<b>7 File Index</b>	<b>25</b>
7.1 File List . . . . .	25
<b>8 Namespace Documentation</b>	<b>27</b>
8.1 markopy Namespace Reference . . . . .	27
8.1.1 Detailed Description . . . . .	27

---

---

8.1.2 Function Documentation . . . . .	27
8.1.2.1 cli_generate() . . . . .	27
8.1.2.2 cli_init() . . . . .	28
8.1.2.3 cli_train() . . . . .	29
8.1.3 Variable Documentation . . . . .	30
8.1.3.1 action . . . . .	30
8.1.3.2 args . . . . .	30
8.1.3.3 default . . . . .	30
8.1.3.4 help . . . . .	30
8.1.3.5 parser . . . . .	31
8.2 Markov Namespace Reference . . . . .	31
8.2.1 Detailed Description . . . . .	31
8.3 Markov::API Namespace Reference . . . . .	31
8.3.1 Detailed Description . . . . .	32
8.4 Markov::API::CLI Namespace Reference . . . . .	32
8.4.1 Detailed Description . . . . .	32
8.4.2 Typedef Documentation . . . . .	32
8.4.2.1 ProgramOptions . . . . .	32
8.4.3 Function Documentation . . . . .	32
8.4.3.1 operator<<() . . . . .	32
8.5 Markov::API::Concurrency Namespace Reference . . . . .	33
8.5.1 Detailed Description . . . . .	33
8.6 Markov::API::CUDA Namespace Reference . . . . .	33
8.6.1 Detailed Description . . . . .	33
8.6.2 Function Documentation . . . . .	33
8.6.2.1 FastRandomWalkCUDAKernel() . . . . .	33
8.6.2.2 strchr() . . . . .	35
8.7 Markov::API::CUDA::Random Namespace Reference . . . . .	35
8.7.1 Detailed Description . . . . .	35
8.7.2 Function Documentation . . . . .	35
8.7.2.1 devrandom() . . . . .	36
8.8 Markov::GUI Namespace Reference . . . . .	36
8.8.1 Detailed Description . . . . .	36
8.9 Markov::Markopy Namespace Reference . . . . .	36
8.9.1 Function Documentation . . . . .	37
8.9.1.1 BOOST_PYTHON_MODULE() . . . . .	37
8.10 Markov::Random Namespace Reference . . . . .	37
8.10.1 Detailed Description . . . . .	38
8.11 model_2gram Namespace Reference . . . . .	38
8.11.1 Detailed Description . . . . .	38
8.11.2 Variable Documentation . . . . .	38
8.11.2.1 alphabet . . . . .	38

---

8.11.2.2 f . . . . .	38
8.12 random Namespace Reference . . . . .	38
8.12.1 Detailed Description . . . . .	38
8.13 random-model Namespace Reference . . . . .	38
8.13.1 Variable Documentation . . . . .	39
8.13.1.1 alphabet . . . . .	39
8.13.1.2 f . . . . .	39
8.14 Testing Namespace Reference . . . . .	39
8.14.1 Detailed Description . . . . .	39
8.15 Testing::MarkovModel Namespace Reference . . . . .	39
8.15.1 Detailed Description . . . . .	39
8.15.2 Function Documentation . . . . .	39
8.15.2.1 TEST_CLASS() [1/3] . . . . .	40
8.15.2.2 TEST_CLASS() [2/3] . . . . .	40
8.15.2.3 TEST_CLASS() [3/3] . . . . .	41
8.16 Testing::MarkovPasswords Namespace Reference . . . . .	42
8.16.1 Detailed Description . . . . .	42
8.17 Testing::MVP Namespace Reference . . . . .	43
8.17.1 Detailed Description . . . . .	43
8.18 Testing::MVP::MarkovModel Namespace Reference . . . . .	43
8.18.1 Detailed Description . . . . .	43
8.18.2 Function Documentation . . . . .	43
8.18.2.1 TEST_CLASS() [1/3] . . . . .	43
8.18.2.2 TEST_CLASS() [2/3] . . . . .	45
8.18.2.3 TEST_CLASS() [3/3] . . . . .	45
8.19 Testing::MVP::MarkovPasswords Namespace Reference . . . . .	48
8.19.1 Detailed Description . . . . .	49
8.19.2 Function Documentation . . . . .	49
8.19.2.1 TEST_CLASS() . . . . .	49
<b>9 Class Documentation</b>	<b>51</b>
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 Markov::API::CLI::Argparse Class Reference	55
9.3.1 Detailed Description	57
9.3.2 Constructor & Destructor Documentation	57
9.3.2.1 Argparse() [1/2]	57
9.3.2.2 Argparse() [2/2]	57
9.3.3 Member Function Documentation	59
9.3.3.1 getProgramOptions()	59
9.3.3.2 help()	59
9.3.3.3 parse()	60
9.3.3.4 setProgramOptions()	60
9.3.4 Member Data Documentation	61
9.3.4.1 po	61
9.4 Markov::GUI::CLI Class Reference	61
9.4.1 Detailed Description	62
9.4.2 Constructor & Destructor Documentation	62
9.4.2.1 CLI()	62
9.4.3 Member Function Documentation	63
9.4.3.1 about	63
9.4.3.2 start	63
9.4.3.3 statistics	64
9.4.4 Member Data Documentation	64
9.4.4.1 ui	64
9.5 Markov::API::CUDA::CUDADeviceController Class Reference	64
9.5.1 Detailed Description	66
9.5.2 Member Function Documentation	66
9.5.2.1 CudaCheckNotifyErr()	66
9.5.2.2 CudaMalloc2DToFlat()	67
9.5.2.3 CudaMemcpy2DToFlat()	68
9.5.2.4 CudaMigrate2DFlat()	69
9.5.2.5 ListCudaDevices()	70
9.6 Markov::API::CUDA::CUDAModelMatrix Class Reference	71
9.6.1 Detailed Description	75
9.6.2 Member Function Documentation	75
9.6.2.1 AdjustEdge()	76

---

---

9.6.2.2 AllocVRAMOutputBuffer()	76
9.6.2.3 Buff()	77
9.6.2.4 ConstructMatrix()	78
9.6.2.5 CudaCheckNotifyErr()	79
9.6.2.6 CudaMalloc2DToFlat()	80
9.6.2.7 CudaMemcpy2DToFlat()	81
9.6.2.8 CudaMigrate2DFlat()	82
9.6.2.9 DeallocateMatrix()	83
9.6.2.10 DumpJSON()	84
9.6.2.11 Edges()	85
9.6.2.12 Export() [1/2]	85
9.6.2.13 Export() [2/2]	85
9.6.2.14 FastRandomWalk() [1/3]	86
9.6.2.15 FastRandomWalk() [2/3]	87
9.6.2.16 FastRandomWalk() [3/3]	88
9.6.2.17 FastRandomWalkPartition()	89
9.6.2.18 FastRandomWalkThread()	90
9.6.2.19 FlattenMatrix()	92
9.6.2.20 GatherAsyncKernelOutput()	92
9.6.2.21 Generate()	92
9.6.2.22 GenerateThread()	94
9.6.2.23 Import() [1/2]	95
9.6.2.24 Import() [2/2]	96
9.6.2.25 LaunchAsyncKernel()	97
9.6.2.26 ListCudaDevices()	97
9.6.2.27 MigrateMatrix()	97
9.6.2.28 Nodes()	98
9.6.2.29 OpenDatasetFile()	98
9.6.2.30 OptimizeEdgeOrder()	99
9.6.2.31 prepKernelMemoryChannel()	99
9.6.2.32 RandomWalk()	100
9.6.2.33 Save()	101
9.6.2.34 StarterNode()	101
9.6.2.35 Train()	102
9.6.2.36 TrainThread()	103
9.6.3 Member Data Documentation	104
9.6.3.1 alternatingKernels	104
9.6.3.2 cudaBlocks	104
9.6.3.3 cudaGridSize	104
9.6.3.4 cudaMemPerGrid	104
9.6.3.5 cudaPerKernelAllocationSize	104
9.6.3.6 cudastreams	104

---

9.6.3.7 cudaThreads . . . . .	105
9.6.3.8 datasetFile . . . . .	105
9.6.3.9 device_edgeMatrix . . . . .	105
9.6.3.10 device_matrixIndex . . . . .	105
9.6.3.11 device_outputBuffer . . . . .	105
9.6.3.12 device_seeds . . . . .	105
9.6.3.13 device_totalEdgeWeights . . . . .	105
9.6.3.14 device_valueMatrix . . . . .	105
9.6.3.15 edgeMatrix . . . . .	105
9.6.3.16 edges . . . . .	106
9.6.3.17 flatEdgeMatrix . . . . .	106
9.6.3.18 flatValueMatrix . . . . .	106
9.6.3.19 iterationsPerKernelThread . . . . .	106
9.6.3.20 matrixIndex . . . . .	106
9.6.3.21 matrixSize . . . . .	106
9.6.3.22 modelSavefile . . . . .	106
9.6.3.23 nodes . . . . .	107
9.6.3.24 numberOfPartitions . . . . .	107
9.6.3.25 outputBuffer . . . . .	107
9.6.3.26 outputFile . . . . .	107
9.6.3.27 ready . . . . .	107
9.6.3.28 starterNode . . . . .	107
9.6.3.29 totalEdgeWeights . . . . .	107
9.6.3.30 totalOutputPerKernel . . . . .	107
9.6.3.31 totalOutputPerSync . . . . .	108
9.6.3.32 valueMatrix . . . . .	108
<b>9.7 Markov::Random::DefaultRandomEngine Class Reference . . . . .</b>	<b>108</b>
9.7.1 Detailed Description . . . . .	110
9.7.2 Member Function Documentation . . . . .	111
9.7.2.1 distribution() . . . . .	111
9.7.2.2 generator() . . . . .	111
9.7.2.3 random() . . . . .	112
9.7.2.4 rd() . . . . .	112
<b>9.8 Markov::Edge&lt; NodeStorageType &gt; Class Template Reference . . . . .</b>	<b>113</b>
9.8.1 Detailed Description . . . . .	114
9.8.2 Constructor & Destructor Documentation . . . . .	114
9.8.2.1 Edge() [1/2] . . . . .	114
9.8.2.2 Edge() [2/2] . . . . .	114
9.8.3 Member Function Documentation . . . . .	115
9.8.3.1 AdjustEdge() . . . . .	115
9.8.3.2 EdgeWeight() . . . . .	115
9.8.3.3 LeftNode() . . . . .	116

---

9.8.3.4 RightNode()	116
9.8.3.5 SetLeftEdge()	117
9.8.3.6 SetRightEdge()	117
9.8.3.7 TraverseNode()	117
9.8.4 Member Data Documentation	117
9.8.4.1 _left	117
9.8.4.2 _right	118
9.8.4.3 _weight	118
9.9 Markov::GUI::Generate Class Reference	118
9.9.1 Detailed Description	119
9.9.2 Constructor & Destructor Documentation	119
9.9.2.1 Generate()	119
9.9.3 Member Function Documentation	120
9.9.3.1 generation	120
9.9.3.2 home	121
9.9.3.3 train	121
9.9.3.4 vis	121
9.9.4 Member Data Documentation	122
9.9.4.1 ui	122
9.10 Markov::API::MarkovPasswords Class Reference	122
9.10.1 Detailed Description	125
9.10.2 Constructor & Destructor Documentation	126
9.10.2.1 MarkovPasswords() [1/2]	126
9.10.2.2 MarkovPasswords() [2/2]	126
9.10.3 Member Function Documentation	126
9.10.3.1 AdjustEdge()	126
9.10.3.2 Buff()	127
9.10.3.3 Edges()	128
9.10.3.4 Export() [1/2]	129
9.10.3.5 Export() [2/2]	129
9.10.3.6 Generate()	129
9.10.3.7 GenerateThread()	130
9.10.3.8 Import() [1/2]	131
9.10.3.9 Import() [2/2]	132
9.10.3.10 Nodes()	133
9.10.3.11 OpenDatasetFile()	133
9.10.3.12 OptimizeEdgeOrder()	133
9.10.3.13 RandomWalk()	134
9.10.3.14 Save()	135
9.10.3.15 StarterNode()	135
9.10.3.16 Train()	136
9.10.3.17 TrainThread()	137

9.10.4 Member Data Documentation . . . . .	138
9.10.4.1 datasetFile . . . . .	138
9.10.4.2 edges . . . . .	138
9.10.4.3 modelSavefile . . . . .	138
9.10.4.4 nodes . . . . .	138
9.10.4.5 outputFile . . . . .	138
9.10.4.6 starterNode . . . . .	138
9.11 Markov::GUI::MarkovPasswordsGUI Class Reference . . . . .	139
9.11.1 Detailed Description . . . . .	140
9.11.2 Constructor & Destructor Documentation . . . . .	140
9.11.2.1 MarkovPasswordsGUI() . . . . .	140
9.11.3 Member Function Documentation . . . . .	141
9.11.3.1 benchmarkSelected . . . . .	141
9.11.3.2 home . . . . .	141
9.11.3.3 model . . . . .	141
9.11.3.4 pass . . . . .	141
9.11.4 Member Data Documentation . . . . .	141
9.11.4.1 ui . . . . .	142
9.12 Markov::Random::Marsaglia Class Reference . . . . .	142
9.12.1 Detailed Description . . . . .	145
9.12.2 Constructor & Destructor Documentation . . . . .	145
9.12.2.1 Marsaglia() . . . . .	145
9.12.3 Member Function Documentation . . . . .	145
9.12.3.1 distribution() . . . . .	145
9.12.3.2 generator() . . . . .	146
9.12.3.3 random() . . . . .	146
9.12.3.4 rd() . . . . .	147
9.12.4 Member Data Documentation . . . . .	147
9.12.4.1 x . . . . .	147
9.12.4.2 y . . . . .	147
9.12.4.3 z . . . . .	148
9.13 Markov::API::CUDA::Random::Marsaglia Class Reference . . . . .	148
9.13.1 Detailed Description . . . . .	150
9.13.2 Member Function Documentation . . . . .	150
9.13.2.1 CudaCheckNotifyErr() . . . . .	150
9.13.2.2 CudaMalloc2DToFlat() . . . . .	151
9.13.2.3 CudaMemcpy2DToFlat() . . . . .	152
9.13.2.4 CudaMigrate2DFlat() . . . . .	153
9.13.2.5 distribution() . . . . .	154
9.13.2.6 generator() . . . . .	155
9.13.2.7 ListCudaDevices() . . . . .	155
9.13.2.8 MigrateToVRAM() . . . . .	156

---

9.13.2.9 random() . . . . .	156
9.13.2.10 rd() . . . . .	157
9.13.3 Member Data Documentation . . . . .	157
9.13.3.1 x . . . . .	157
9.13.3.2 y . . . . .	157
9.13.3.3 z . . . . .	157
9.14 Markov::GUI::menu Class Reference . . . . .	158
9.14.1 Detailed Description . . . . .	159
9.14.2 Constructor & Destructor Documentation . . . . .	159
9.14.2.1 menu() . . . . .	159
9.14.3 Member Function Documentation . . . . .	159
9.14.3.1 about . . . . .	159
9.14.3.2 visualization . . . . .	160
9.14.4 Member Data Documentation . . . . .	160
9.14.4.1 ui . . . . .	160
9.15 Markov::Random::Mersenne Class Reference . . . . .	160
9.15.1 Detailed Description . . . . .	162
9.15.2 Member Function Documentation . . . . .	163
9.15.2.1 distribution() . . . . .	163
9.15.2.2 generator() . . . . .	163
9.15.2.3 random() . . . . .	164
9.15.2.4 rd() . . . . .	164
9.16 Markov::Model< NodeStorageType > Class Template Reference . . . . .	165
9.16.1 Detailed Description . . . . .	166
9.16.2 Constructor & Destructor Documentation . . . . .	166
9.16.2.1 Model() . . . . .	166
9.16.3 Member Function Documentation . . . . .	166
9.16.3.1 AdjustEdge() . . . . .	167
9.16.3.2 Edges() . . . . .	167
9.16.3.3 Export() [1/2] . . . . .	168
9.16.3.4 Export() [2/2] . . . . .	168
9.16.3.5 Import() [1/2] . . . . .	169
9.16.3.6 Import() [2/2] . . . . .	169
9.16.3.7 Nodes() . . . . .	170
9.16.3.8 OptimizeEdgeOrder() . . . . .	171
9.16.3.9 RandomWalk() . . . . .	171
9.16.3.10 StarterNode() . . . . .	173
9.16.4 Member Data Documentation . . . . .	173
9.16.4.1 edges . . . . .	173
9.16.4.2 nodes . . . . .	173
9.16.4.3 starterNode . . . . .	173
9.17 Markov::API::ModelMatrix Class Reference . . . . .	173

---

---

9.17.1 Detailed Description . . . . .	177
9.17.2 Constructor & Destructor Documentation . . . . .	177
9.17.2.1 ModelMatrix() . . . . .	177
9.17.3 Member Function Documentation . . . . .	177
9.17.3.1 AdjustEdge() . . . . .	177
9.17.3.2 Buff() . . . . .	178
9.17.3.3 ConstructMatrix() . . . . .	179
9.17.3.4 DeallocateMatrix() . . . . .	181
9.17.3.5 DumpJSON() . . . . .	182
9.17.3.6 Edges() . . . . .	183
9.17.3.7 Export() [1/2] . . . . .	183
9.17.3.8 Export() [2/2] . . . . .	183
9.17.3.9 FastRandomWalk() [1/2] . . . . .	184
9.17.3.10 FastRandomWalk() [2/2] . . . . .	185
9.17.3.11 FastRandomWalkPartition() . . . . .	186
9.17.3.12 FastRandomWalkThread() . . . . .	187
9.17.3.13 Generate() . . . . .	188
9.17.3.14 GenerateThread() . . . . .	190
9.17.3.15 Import() [1/2] . . . . .	191
9.17.3.16 Import() [2/2] . . . . .	192
9.17.3.17 Nodes() . . . . .	193
9.17.3.18 OpenDatasetFile() . . . . .	193
9.17.3.19 OptimizeEdgeOrder() . . . . .	193
9.17.3.20 RandomWalk() . . . . .	194
9.17.3.21 Save() . . . . .	195
9.17.3.22 StarterNode() . . . . .	195
9.17.3.23 Train() . . . . .	196
9.17.3.24 TrainThread() . . . . .	197
9.17.4 Member Data Documentation . . . . .	197
9.17.4.1 datasetFile . . . . .	198
9.17.4.2 edgeMatrix . . . . .	198
9.17.4.3 edges . . . . .	198
9.17.4.4 matrixIndex . . . . .	198
9.17.4.5 matrixSize . . . . .	198
9.17.4.6 modelSavefile . . . . .	198
9.17.4.7 nodes . . . . .	198
9.17.4.8 outputFile . . . . .	198
9.17.4.9 ready . . . . .	199
9.17.4.10 starterNode . . . . .	199
9.17.4.11 totalEdgeWeights . . . . .	199
9.17.4.12 valueMatrix . . . . .	199
9.18 Markov::Node< storageType > Class Template Reference . . . . .	199

---

---

9.18.1 Detailed Description . . . . .	201
9.18.2 Constructor & Destructor Documentation . . . . .	201
9.18.2.1 Node() [1/2] . . . . .	201
9.18.2.2 Node() [2/2] . . . . .	201
9.18.3 Member Function Documentation . . . . .	202
9.18.3.1 Edges() . . . . .	202
9.18.3.2 FindEdge() [1/2] . . . . .	202
9.18.3.3 FindEdge() [2/2] . . . . .	202
9.18.3.4 Link() [1/2] . . . . .	203
9.18.3.5 Link() [2/2] . . . . .	203
9.18.3.6 NodeValue() . . . . .	204
9.18.3.7 RandomNext() . . . . .	204
9.18.3.8 TotalEdgeWeights() . . . . .	205
9.18.3.9 UpdateEdges() . . . . .	205
9.18.3.10 UpdateTotalVerticeWeight() . . . . .	206
9.18.4 Member Data Documentation . . . . .	206
9.18.4.1 _value . . . . .	206
9.18.4.2 edges . . . . .	206
9.18.4.3 edgesV . . . . .	206
9.18.4.4 total_edge_weights . . . . .	206
9.19 QMainWindow Class Reference . . . . .	207
9.20 Markov::Random::RandomEngine Class Reference . . . . .	207
9.20.1 Detailed Description . . . . .	209
9.20.2 Member Function Documentation . . . . .	209
9.20.2.1 random() . . . . .	209
9.21 Markov::API::CLI::Terminal Class Reference . . . . .	209
9.21.1 Detailed Description . . . . .	211
9.21.2 Member Enumeration Documentation . . . . .	211
9.21.2.1 color . . . . .	211
9.21.3 Constructor & Destructor Documentation . . . . .	211
9.21.3.1 Terminal() . . . . .	211
9.21.4 Member Data Documentation . . . . .	211
9.21.4.1 colormap . . . . .	212
9.21.4.2 endl . . . . .	212
9.22 Markov::API::Concurrency::ThreadSharedListHandler Class Reference . . . . .	212
9.22.1 Detailed Description . . . . .	214
9.22.2 Constructor & Destructor Documentation . . . . .	214
9.22.2.1 ThreadSharedListHandler() . . . . .	214
9.22.3 Member Function Documentation . . . . .	215
9.22.3.1 next() . . . . .	215
9.22.4 Member Data Documentation . . . . .	215
9.22.4.1 listfile . . . . .	215

---

---

9.22.4.2 mlock . . . . .	215
9.23 Markov::GUI::Train Class Reference . . . . .	216
9.23.1 Detailed Description . . . . .	217
9.23.2 Constructor & Destructor Documentation . . . . .	217
9.23.2.1 Train() . . . . .	217
9.23.3 Member Function Documentation . . . . .	218
9.23.3.1 home . . . . .	218
9.23.3.2 train . . . . .	218
9.23.4 Member Data Documentation . . . . .	219
9.23.4.1 ui . . . . .	219
<b>10 File Documentation</b> . . . . .	<b>221</b>
10.1 about.cpp File Reference . . . . .	221
10.1.1 Detailed Description . . . . .	221
10.2 about.cpp . . . . .	221
10.3 about.h File Reference . . . . .	221
10.3.1 Detailed Description . . . . .	222
10.4 about.h . . . . .	223
10.5 argparse.cpp File Reference . . . . .	223
10.5.1 Detailed Description . . . . .	223
10.6 argparse.cpp . . . . .	223
10.7 argparse.h File Reference . . . . .	224
10.7.1 Detailed Description . . . . .	225
10.7.2 Macro Definition Documentation . . . . .	225
10.7.2.1 BOOST_ALL_STATIC_LIB . . . . .	226
10.7.2.2 BOOST_PROGRAM_OPTIONS_STATIC_LIB . . . . .	226
10.8 argparse.h . . . . .	226
10.9 CLI.cpp File Reference . . . . .	229
10.9.1 Detailed Description . . . . .	229
10.10 CLI.cpp . . . . .	229
10.11 CLI.h File Reference . . . . .	230
10.11.1 Detailed Description . . . . .	231
10.12 CLI.h . . . . .	231
10.13 cudaDeviceController.cu File Reference . . . . .	231
10.13.1 Detailed Description . . . . .	232
10.14 cudaDeviceController.cu . . . . .	232
10.15 cudaDeviceController.h File Reference . . . . .	233
10.15.1 Detailed Description . . . . .	234
10.16 cudaDeviceController.h . . . . .	234
10.17 cudaModelMatrix.cu File Reference . . . . .	236
10.17.1 Detailed Description . . . . .	237
10.18 cudaModelMatrix.cu . . . . .	237

---

---

10.19 cudaModelMatrix.h File Reference . . . . .	240
10.19.1 Detailed Description . . . . .	242
10.20 cudaModelMatrix.h . . . . .	242
10.21 cudarandom.h File Reference . . . . .	244
10.21.1 Detailed Description . . . . .	245
10.22 cudarandom.h . . . . .	246
10.23 dllmain.cpp File Reference . . . . .	246
10.23.1 Detailed Description . . . . .	247
10.24 dllmain.cpp . . . . .	247
10.25 edge.h File Reference . . . . .	248
10.25.1 Detailed Description . . . . .	249
10.26 edge.h . . . . .	249
10.27 framework.h File Reference . . . . .	251
10.27.1 Detailed Description . . . . .	251
10.27.2 Macro Definition Documentation . . . . .	251
10.27.2.1 WIN32_LEAN_AND_MEAN . . . . .	252
10.28 framework.h . . . . .	252
10.29 Generate.cpp File Reference . . . . .	252
10.29.1 Detailed Description . . . . .	252
10.30 Generate.cpp . . . . .	252
10.31 Generate.h File Reference . . . . .	254
10.31.1 Detailed Description . . . . .	255
10.32 Generate.h . . . . .	255
10.33 main.cpp File Reference . . . . .	256
10.33.1 Detailed Description . . . . .	256
10.33.2 Function Documentation . . . . .	256
10.33.2.1 main() . . . . .	257
10.34 MarkovAPICLI/src/main.cpp . . . . .	257
10.35 main.cpp File Reference . . . . .	258
10.35.1 Detailed Description . . . . .	259
10.35.2 Function Documentation . . . . .	259
10.35.2.1 main() . . . . .	259
10.36 MarkovPasswordsGUI/src/main.cpp . . . . .	259
10.37 main.cu File Reference . . . . .	260
10.37.1 Detailed Description . . . . .	260
10.37.2 Function Documentation . . . . .	260
10.37.2.1 main() . . . . .	260
10.38 main.cu . . . . .	261
10.39 markopy.cpp File Reference . . . . .	261
10.39.1 Detailed Description . . . . .	262
10.39.2 Macro Definition Documentation . . . . .	263
10.39.2.1 BOOST_ALL_STATIC_LIB . . . . .	263

---

---

10.39.2.2 BOOST_PYTHON_STATIC_LIB . . . . .	263
10.40 markopy.cpp . . . . .	263
10.41 markopy.py File Reference . . . . .	264
10.42 markopy.py . . . . .	264
10.43 markovPasswords.cpp File Reference . . . . .	266
10.43.1 Detailed Description . . . . .	267
10.43.2 Function Documentation . . . . .	267
10.43.2.1 intHandler() . . . . .	267
10.43.3 Variable Documentation . . . . .	268
10.43.3.1 keepRunning . . . . .	268
10.44 markovPasswords.cpp . . . . .	268
10.45 markovPasswords.h File Reference . . . . .	270
10.45.1 Detailed Description . . . . .	271
10.46 markovPasswords.h . . . . .	271
10.47 MarkovPasswordsGUI.cpp File Reference . . . . .	273
10.47.1 Detailed Description . . . . .	273
10.48 MarkovPasswordsGUI.cpp . . . . .	273
10.49 MarkovPasswordsGUI.h File Reference . . . . .	274
10.49.1 Detailed Description . . . . .	275
10.50 MarkovPasswordsGUI.h . . . . .	275
10.51 menu.cpp File Reference . . . . .	276
10.51.1 Detailed Description . . . . .	276
10.52 menu.cpp . . . . .	276
10.53 menu.h File Reference . . . . .	277
10.53.1 Detailed Description . . . . .	278
10.54 menu.h . . . . .	278
10.55 model.h File Reference . . . . .	278
10.55.1 Detailed Description . . . . .	279
10.56 model.h . . . . .	279
10.57 model_2gram.py File Reference . . . . .	284
10.58 model_2gram.py . . . . .	284
10.59 modelMatrix.cpp File Reference . . . . .	284
10.59.1 Detailed Description . . . . .	285
10.60 modelMatrix.cpp . . . . .	285
10.61 modelMatrix.h File Reference . . . . .	288
10.61.1 Detailed Description . . . . .	289
10.62 modelMatrix.h . . . . .	289
10.63 node.h File Reference . . . . .	292
10.63.1 Detailed Description . . . . .	293
10.64 node.h . . . . .	293
10.65 pch.cpp File Reference . . . . .	297
10.65.1 Detailed Description . . . . .	297

---

---

10.66 MarkovModel/src/pch.cpp . . . . .	297
10.67 pch.cpp File Reference . . . . .	297
10.67.1 Detailed Description . . . . .	298
10.68 UnitTests/pch.cpp . . . . .	298
10.69 pch.h File Reference . . . . .	298
10.69.1 Detailed Description . . . . .	299
10.70 MarkovModel/src/pch.h . . . . .	299
10.71 pch.h File Reference . . . . .	299
10.71.1 Detailed Description . . . . .	300
10.72 UnitTests/pch.h . . . . .	300
10.73 random-model.py File Reference . . . . .	300
10.74 random-model.py . . . . .	301
10.75 random.h File Reference . . . . .	301
10.75.1 Detailed Description . . . . .	302
10.76 random.h . . . . .	303
10.77 README.md File Reference . . . . .	305
10.78 report.md File Reference . . . . .	305
10.79 term.cpp File Reference . . . . .	305
10.79.1 Detailed Description . . . . .	306
10.79.2 Function Documentation . . . . .	306
10.79.2.1 operator<<() . . . . .	306
10.80 term.cpp . . . . .	306
10.81 term.h File Reference . . . . .	307
10.81.1 Detailed Description . . . . .	309
10.81.2 Macro Definition Documentation . . . . .	309
10.81.2.1 TERM_FAIL . . . . .	309
10.81.2.2 TERM_INFO . . . . .	309
10.81.2.3 TERM_SUCC . . . . .	309
10.81.2.4 TERM_WARN . . . . .	309
10.82 term.h . . . . .	309
10.83 threadSharedListHandler.cpp File Reference . . . . .	310
10.83.1 Detailed Description . . . . .	310
10.84 threadSharedListHandler.cpp . . . . .	311
10.85 threadSharedListHandler.h File Reference . . . . .	311
10.85.1 Detailed Description . . . . .	312
10.86 threadSharedListHandler.h . . . . .	312
10.87 Train.cpp File Reference . . . . .	314
10.87.1 Detailed Description . . . . .	314
10.88 Train.cpp . . . . .	314
10.89 Train.h File Reference . . . . .	315
10.89.1 Detailed Description . . . . .	316
10.90 Train.h . . . . .	317

---

10.91 UnitTests.cpp File Reference . . . . .	317
10.91.1 Detailed Description . . . . .	318
10.92 UnitTests.cpp . . . . .	318
<b>Index</b>	<b>327</b>



# CHAPTER1

---

## Markov Passwords

---

### Markov Passwords

Generate wordlists with markov models.

[Wiki](#) · [Complete documentation](#) · [Report Bug](#) · [Add a Bug](#)

### Table of Contents

1. [About The Project](#)
    - [Built With](#)
  2. [Getting Started](#)
    - [Prerequisites](#)
    - [Installation](#)
  3. [Contributing](#)
  4. [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 wrağger for [CudaMarkovAPI](#)

### 1.1.1 Built With

- CPP, with dependencies: boost, python3-dev, QT-5.

## 1.2 Getting Started

If you'd just like to use the project without contributing, check out the releases page. If you want to build, check out wiki for building the project.

---

### 1.2.1 Prerequisites

#### 1.2.1.0.1 MarkovModel

- Make for linux, Visual Studio/MSBuild for Windows.

#### 1.2.1.0.2 MarkovPasswords

- Boost.ProgramOptions (tested on 1.76.0)

#### 1.2.1.0.3 Markopy

- Boost.Python (tested on 1.76.0)
- Python development package (tested on python 3.8)

#### 1.2.1.0.4 MarkovPasswordsGUI

- QT development environment.
- 

## 1.2.2 Installing Dependencies

### 1.2.2.0.1 Windows

- QT: Install [QT For Windows](#)
- Boost:
  - Download Boost from [its website](#)
  - Unzip the contents.
  - Launch "Visual Studio Developer Command Prompt"
  - Move to the boost installation directory. Run `bootstrap.bat`
  - Run `b2`.
- Python: You can use the windows app store to download python runtime and libraries.

### 1.2.2.0.2 Linux

- QT: Follow [this guide](#) to install QT on Linux.
  - Boost: `run sudo apt-get install libboost-all-dev`
  - Python: `run sudo apt-get install python3`
- 

## 1.2.3 Installation

See the Wiki Page

---

## 1.2.4 Building

Building process can be fairly complicated depending on the environment.

---

## 1.3 Linux

If you've set up the dependencies, you can just build the project with make. List of directives is below.

```
.PHONY: all
all: model mp
model: $(INCLUDE) /$(MM_LIB)
mp: $(BIN) /$(MP_EXEC)
markopy: $(BIN) /$(MPY_SO)
.PHONY: clean
clean:
    $(RM) -r $(BIN) /*
```

## 1.4 Windows

Set up correct environment variables for BOOST\_ROOT% (folder containing boost, libs, stage, tools) and PYTHONPATH% (folder containing include, lib, libs, Tools, python.exe/python3.exe).

If you've set up the dependencies and environment variables correctly, you can open the solution with Visual Studio and build with that.

## 1.5 Known Common issues

### 1.5.1 Linux

#### 1.5.1.1 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\*

If it doesn't, run sudo apt-get install python3-dev

#### 1.5.1.2 Markopy/MarkovPasswords - \*.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

### 1.5.2 Windows

#### 1.5.2.1 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\uum;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
```

#### 1.5.2.2 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.

#### 1.5.2.3 Python.h not found

Make sure you have python installed, and make sure you set PYTHON\_PATH environment variable.

#### 1.5.2.4 Simplified Theory

**What is a markov model** Below, is the example [Markov](#) Model which can generate strings with the alphabet "a,b,c"

**Iteration 1** Below is a demonstration of how training will be done. For this example, we are going to adjust the model with string "ab", and our occurrence will be "3" From MarkovPasswords, inside the train function, `Model::adjust` is called with "ab" and "3" parameters.

Now, `Model::adjust` will iteratively adjust the edge weights accordingly. It starts by adjusting weight between start and "a" node. This is done by calling `Edge::adjust` of the edge between the nodes.

After adjustment, `ajust` function iterates to the next character, "b", and does the same thing.

As this string is finished, it will adjust the final weight, `b->"end"`

**Iteration 2** This time, same procedure will be applied for "bacb" string, with occurrence value of 12.

**Iteration 38271** As the model is trained, hidden linguistical patterns start to appear, and our model looks like this With our dataset, without doing any kind of linugistic analysis ourselves, our [Markov](#) Model has highlighted that strings are more likely to start with a, b tends to follow a, and a is likely to be repeated in the string.

---

### 1.5.3 Contributing

Feel free to contribute.

### 1.5.4 Contact

Twitter - [@ahakcil](#)

# CHAPTER2

## NVInsight Report

### 2.0.1 Overview

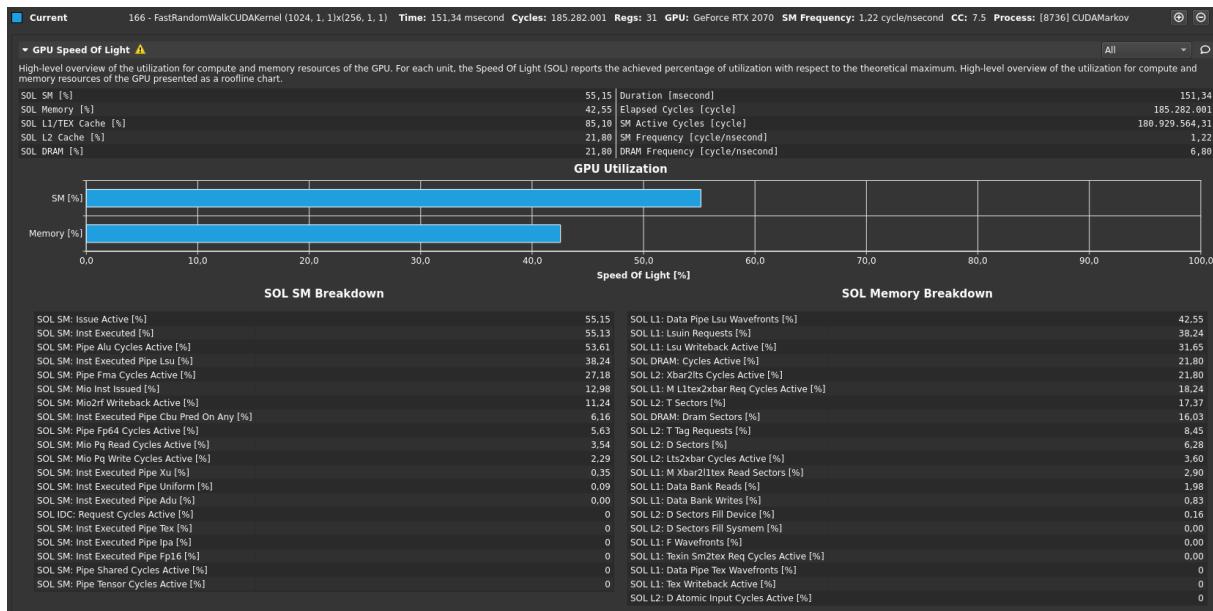


Figure 2.1 SOL SM Breakdown

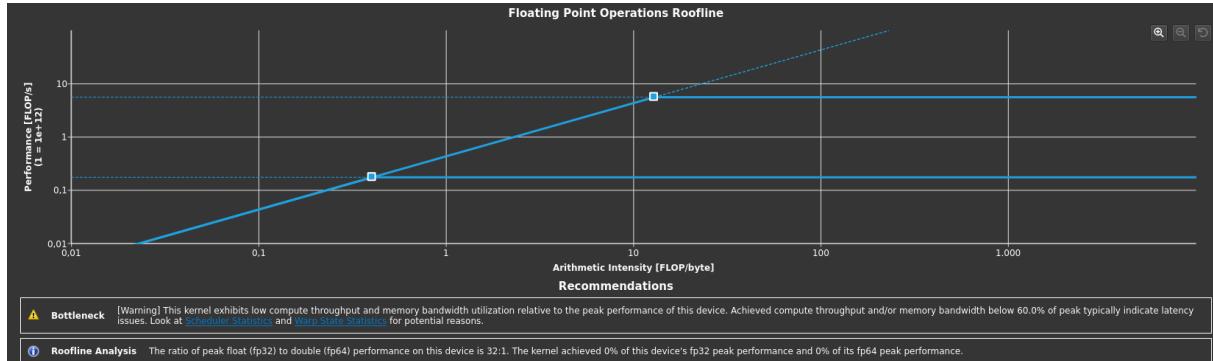


Figure 2.2 Arithmetic Intensity

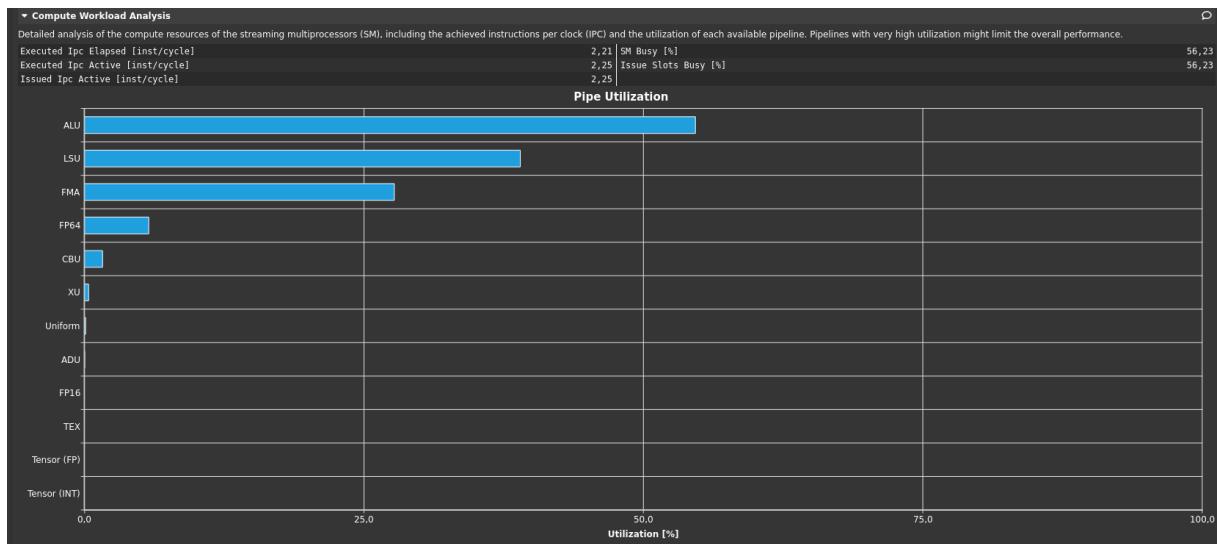


Figure 2.3 Pipe utilization

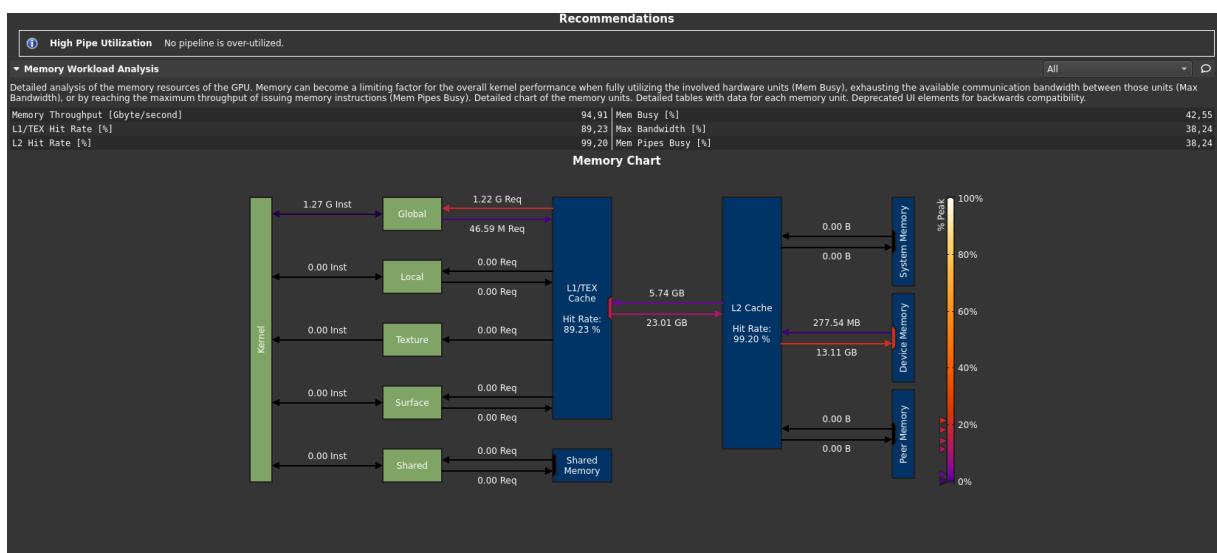


Figure 2.4 Memory Workload Analysis

Shared Memory												
	Instructions	Requests	Wavefronts	% Peak	Bank Conflicts							
Shared Load	0	0	0	0	0							
Shared Load Matrix	0	0	0	0	0							
Shared Store	0	0	0	0	0							
Shared Atomic	0	0	0	0	0							
Other	-	-	8.192	0.00	0							
Total	0	0	8.192	0.00	0							

L1/TEX Cache												
	Instructions	Requests	Wavefronts	% Peak	Sectors	Sectors/Req	Hit Rate	Bytes	Sector Misses to L2	% Peak to L2	Returns to SM	% Peak to SM
Local Load	0	0	0	0	0	0	0	0	192.724.146	2.90	2.101.570.675	31.65
Global Load	1.223.036.801	1.223.036.801	1.747.696.951	26.32	2.987.208.909	2.44	93.85	95.590.685.088				
Surface Load	0	0	0	0	0	0	0	0	0	0	0	0
Texture Load	0	0	0	0	0	0	0	0	0	0	0	0
Global Store	46.586.822	46.586.822	163.741.228	2.47	920.981.177	19.77	74.23	29.471.397.664	772.161.127	11.63	-	-
Local Store	0	0	0	0	0	0	0	0	0	0	-	-
Surface Store	0	0	0	0	0	0	0	0	0	0	-	-
Global Reduction	0	0	0	0	0	0	0	0	0	0	-	-
Surface Reduction	0	0	0	0	0	0	0	0	0	0	-	-
Global Atomic ALU	0	0	0	0	0	0	0	0	0	0	see above	see above
Global Atomic CAS	0	0	0	0	0	0	0	0	0	0	see above	see above
Surface Atomic ALU	0	0	0	0	0	0	0	0	0	0	see above	see above
Surface Atomic CAS	0	0	0	0	0	0	0	0	0	0	see above	see above
Loads	1.223.036.801	1.223.036.801	1.747.696.951	26.32	2.987.208.909	2.44	93.85	95.590.685.088	192.724.146	2.90	2.101.570.675	31.65
Stores	46.586.822	46.586.822	163.741.228	2.47	920.981.177	19.77	74.23	29.471.397.664	772.161.127	11.63	-	-
Total	1.269.623.623	1.269.623.623	1.911.438.179	28.78	3.908.190.086	3.08	89.23	125.062.082.752	964.885.273	14.53	2.101.570.675	31.65

L2 Cache												
	Requests	Sectors	Sectors/Req	% Peak	Hit Rate	Bytes	Throughput	Sector Misses to Device	Sector Misses to System	Sector Misses to Peer		
L1/TEX Load	63.680.170	193.047.854	3.03	3.45	99.46	6.177.531.328	40.817.641.959.44	1.028.090	0	0	0	0
L1/TEX Store	405.095.227	772.063.559	1.91	13.79	100	24.706.033.888	163.243.534.016.14	0	0	0	0	0
L1/TEX Atomic ALU	0	0	0	0	0	0	0	0	0	0	0	0
L1/TEX Atomic CAS	0	0	0	0	0	0	0	0	0	0	0	0
L1/TEX Reduction	0	0	0	0	0	0	0	0	0	0	0	0
L1/TEX Total	468.490.666	963.888.704	2.06	17.22	99.89	30.844.438.528	203.802.648.894.60	1.013.223	0	0	0	0
GPU Total	472.620.531	972.615.552	2.06	17.37	99.07	31.123.697.664	205.847.835.721.17	9.036.807	180	0	0	0

Device Memory												
	Sectors	% Peak	Bytes	Throughput								
Load	9.094.353	0.44	291.019.296	1.922.891.329.35								
Store	439.786.885	21.36	14.073.180.320	92.987.636.166.04								
Total	448.881.238	21.80	14.364.199.616	94.910.527.495.39								

Figure 2.5 Shared memory and L1/L2 cache

L1/TEX Cache (Deprecated)												
	Instructions	L1/TEX Requests	% Peak	Hit Rate	L2 Requests	% Peak	L2 Returns	% Peak	L1/TEX Returns	% Peak		
Global Load Cached	1.223.036.801	1.223.036.801	18.42	93.85	-	-	192.724.146	2.90	2.101.570.675	31.65		
Global Load Uncached	-	-	-	-	-	-	-	-	-	-		
Local Load Cached	0	0	0	0	-	-	-	-	-	-		
Local Load Uncached	-	-	-	-	-	-	-	-	-	-		
Surface Load	0	0	0	0	-	-	0	0	0	0		
Texture Load	0	0	0	0	-	-	0	0	0	0		
Global Store	46.586.822	46.586.822	0.70	74.23	772.161.127	11.63	-	-	-	-		
Local Store	0	0	0	0	-	-	-	-	-	-		
Surface Store	0	0	0	0	0	0	-	-	-	-		
Global Reduction	0	0	0	0	0	0	-	-	-	-		
Surface Reduction	0	0	0	0	0	0	-	-	-	-		
Global Atomic ALU	0	0	0	0	0	0	0	0	0	0	see above	see above
Global Atomic CAS	0	0	0	0	0	0	0	0	0	0	see above	see above
Surface Atomic ALU	0	0	0	0	0	0	0	0	0	0	see above	see above
Surface Atomic CAS	0	0	0	0	0	0	0	0	0	0	see above	see above
Loads	1.223.036.801	1.223.036.801	18.42	93.85	-	-	192.724.146	2.90	2.101.570.675	31.65		
Stores	46.586.822	46.586.822	0.70	74.23	772.161.127	11.63	-	-	-	-		
Total	1.269.623.623	1.269.623.623	19.12	89.23	772.161.127	11.63	192.724.146	2.90	2.101.570.675	31.65		

L2 Cache (Deprecated)												
	L2 Requests	% Peak	L2 Returns	% Peak	Total Bytes	Total Throughput						
Global Load Cached	-	-	-	-	6.167.172.672	40.749.197.804.43						
Global Load Uncached	-	-	192.724.146	2.90	-	-						
Local Load Cached	-	-	-	-	-	-						
Local Load Uncached	-	-	-	-	-	-						
Surface Load	-	-	0	0	0	0						
Texture Load	-	-	0	0	0	0						
Global Store	772.161.127	11.63	-	-	24.709.156.064	163.264.163.593.77						
Local Store	-	-	-	-	-	-						
Surface Store	0	0	-	-	0	0						
Global Reduction	0	0	-	-	0	0						
Surface Reduction	0	0	-	-	0	0						
Global Atomic ALU	0	0	0	0	0	0						
Global Atomic CAS	0	0	0	0	0	0						
Surface Atomic ALU	0	0	0	0	0	0						
Surface Atomic CAS	0	0	0	0	0	0						
Loads	-	-	192.724.146	2.90	6.167.172.672	40.749.197.804.43						
Stores	772.161.127	11.63	-	-	24.709.156.064	163.264.163.593.77						
Total	772.161.127	11.63	192.724.146	2.90	30.876.328.736	204.013.361.398.20						

Figure 2.6 L1/L2 Cache

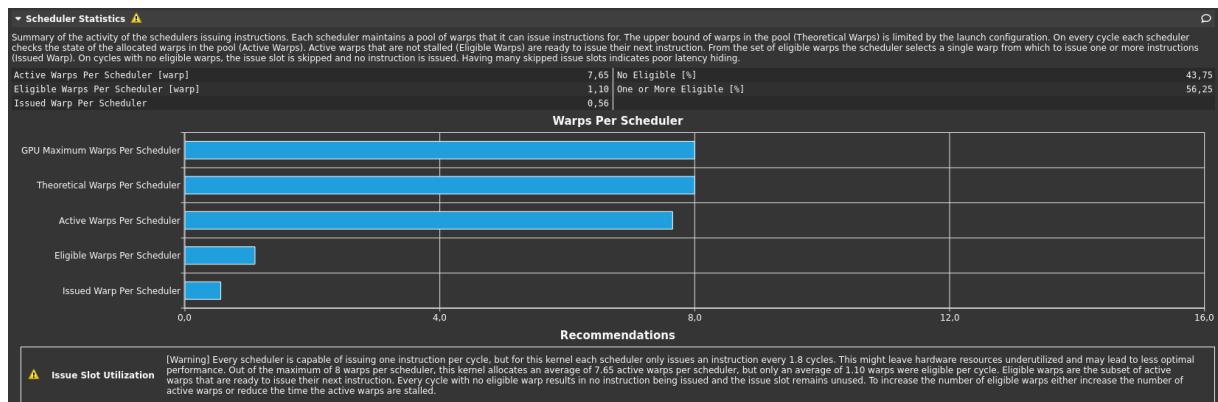


Figure 2.7 Warp Scheduler

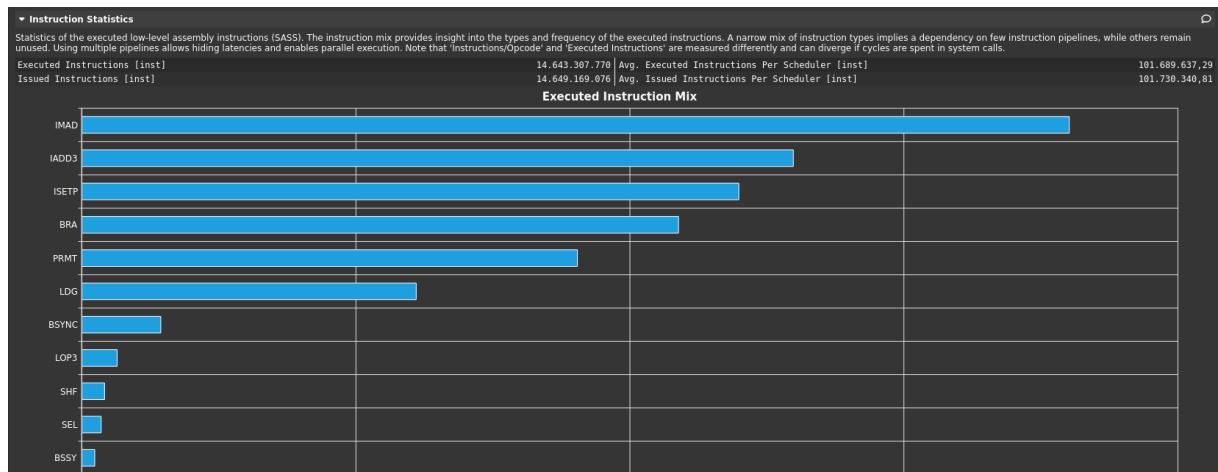


Figure 2.8 Instruction statistics



Figure 2.9 Instruction statistics continued

Launch Statistics	
Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.	
Grid Size	1,024 Registers Per Thread [register/thread]
Block Size	1,056 Static Shared Memory Per Block [byte/block]
Threads [thread]	262,144 Dynamic Shared Memory Per Block [byte/block]
Waves Per SM	7,111 Driver Shared Memory Per Block [byte/block]
Function Cache Configuration	cudaFuncCacheNone Shared Memory Configuration Size [kbyte] 32,777
Occupancy	
Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.	
Theoretical Occupancy [%]	100 Block Limit Registers [block]
Theoretical Active Warps per SM [warp]	32 Block Limit Shared Mem [block]
Achieved Occupancy [%]	95.67 Block Limit Warps [block]
Achieved Active Warps Per SM [warp]	30,621 Block Limit SM [block]

Figure 2.10 Launch statistics and occupancy

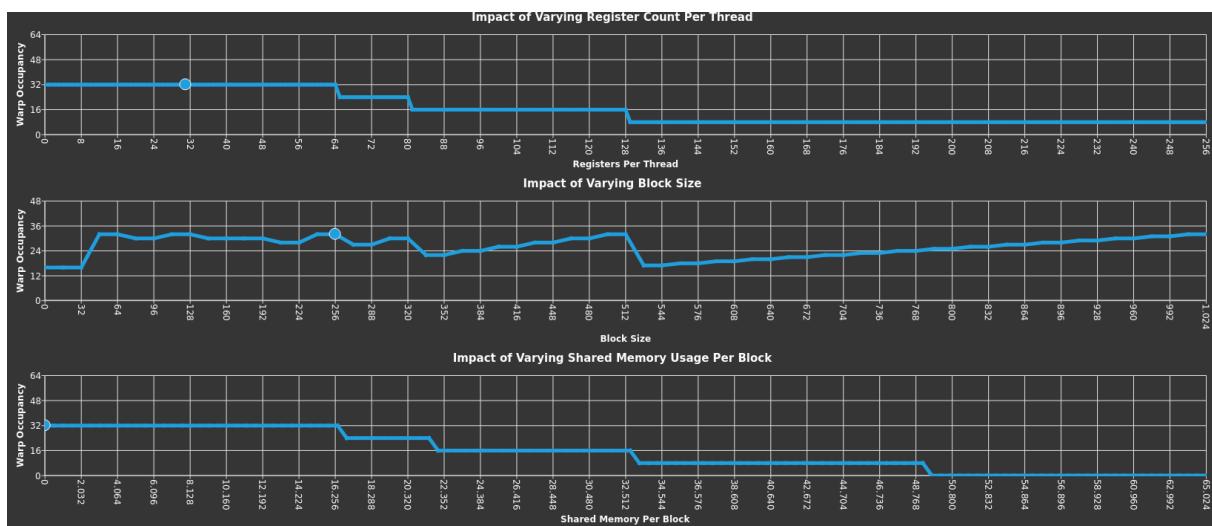


Figure 2.11 Warp occupancy analysis

Recommendations	
Occupancy	This kernel's theoretical occupancy is not impacted by any block limit.
Source Counters	
Branch Instructions (inst)	2,503,193,367 Branch Efficiency (%) 88.83
Branch Instructions Ratio (%)	0.17 Avg. Divergent Branches 1,707,456,31
Most Instructions Executed	
Location	Value
0x7f3856f66d40 in FastRandomWalkCUDAKernel	876,282,222
0x7f3856f66e20 in FastRandomWalkCUDAKernel	876,282,222
0x7f3856f66e10 in FastRandomWalkCUDAKernel	876,282,222
0x7f3856f66e00 in FastRandomWalkCUDAKernel	876,282,222
0x7f3856f66d00 in FastRandomWalkCUDAKernel	876,282,222
Recommendations	
Uncoalesced Global Accesses	[Warning] Uncoalesced global access, expected 495729524 sectors, got 1129463138 (2.28x) at PC 0x7f3856f972e0
Uncoalesced Global Accesses	[Warning] Uncoalesced global access, expected 10706038 sectors, got 235929600 (22.04x) at PC 0x7f3856f97500
Uncoalesced Global Accesses	[Warning] Uncoalesced global access, expected 73523408 sectors, got 219526454 (2.99x) at PC 0x7f3856f96e50
Uncoalesced Global Accesses	[Warning] Uncoalesced global access, expected 73523408 sectors, got 219526454 (2.99x) at PC 0x7f3856f96e00
Uncoalesced Global Accesses	[Warning] Uncoalesced global access, expected 73523408 sectors, got 219524734 (2.99x) at PC 0x7f3856f96e60
Uncoalesced Global Accesses	[Warning] Uncoalesced global access, expected 73523408 sectors, got 219524734 (2.99x) at PC 0x7f3856f96e40
Uncoalesced Global Accesses	[Warning] Uncoalesced global access, expected 73523408 sectors, got 219523845 (2.99x) at PC 0x7f3856f96e70
Uncoalesced Global Accesses	[Warning] Uncoalesced global access, expected 73523408 sectors, got 219523845 (2.99x) at PC 0x7f3856f96e50
Uncoalesced Global Accesses	[Warning] Uncoalesced global access, expected 112931494 sectors, got 242418510 (2.15x) at PC 0x7f3856f931a0
Uncoalesced Global Accesses	[Warning] Uncoalesced global access, expected 819200 sectors, got 26214400 (32.00x) at PC 0x7f3856f97530

Figure 2.12 Instructional statistics

## 2.0.2 Session

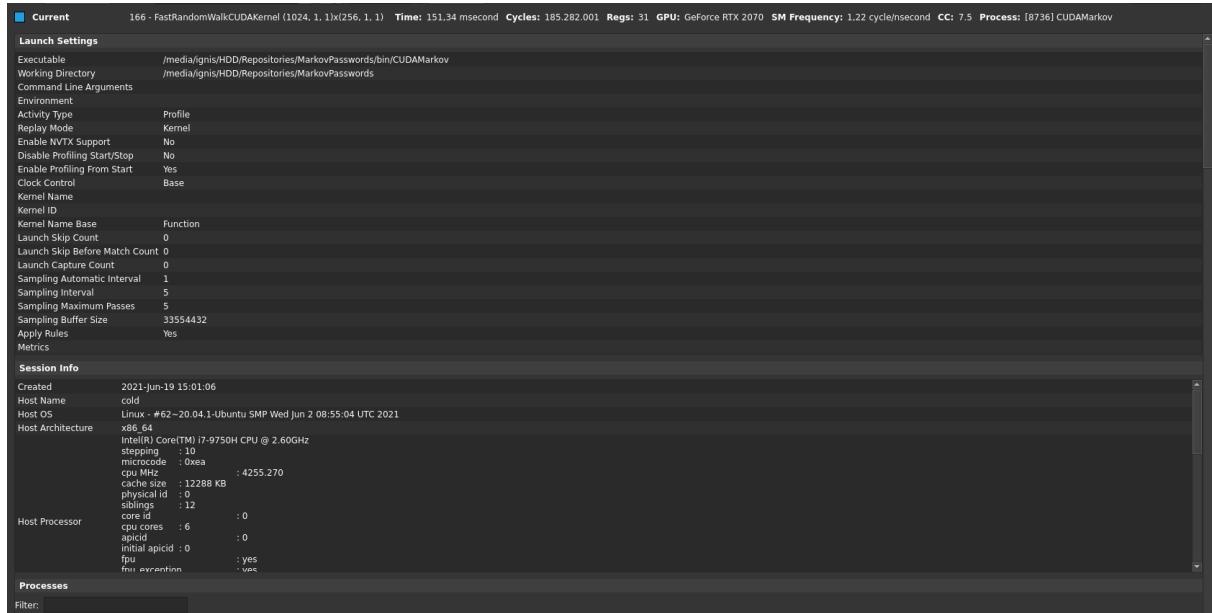


Figure 2.13 Session overview

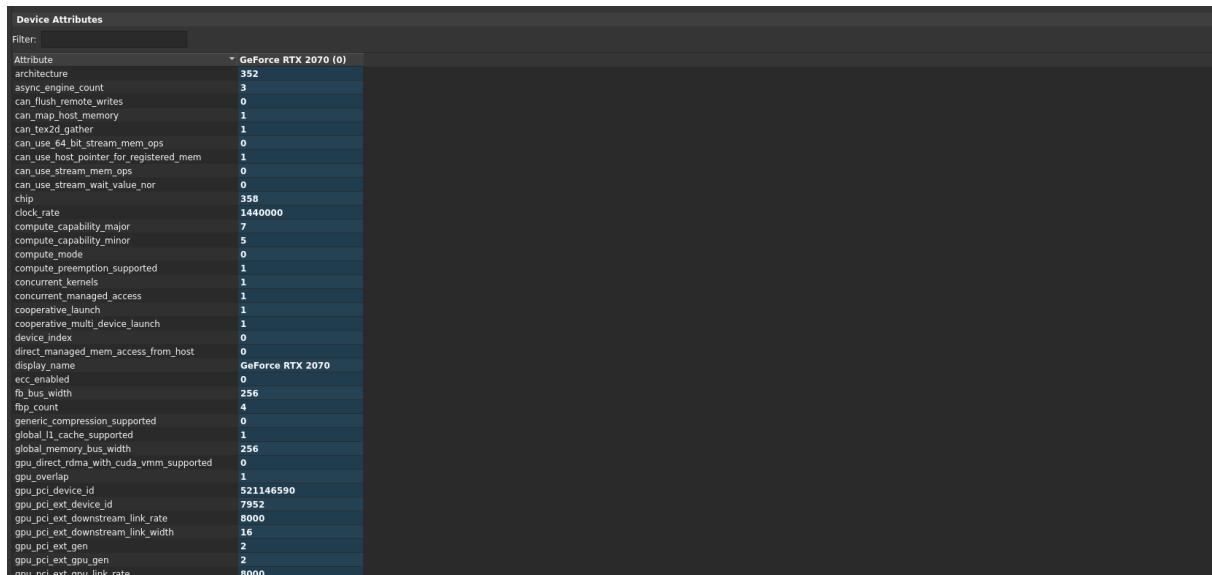


Figure 2.14 Device details

gpu_pci_ext_gpu_link_width	16
gpu_pci_revision_id	161
gpu_pci_sub_system_id	274537733
handle_type_posix_file_descriptor_supported	1
handle_type_win32_handle_supported	0
handle_type_win32_kmt_handle_supported	0
host_native_atomic_supported	0
host_register_supported	1
implementation	358
integrated	0
kernel_exec_timeout	0
l2_cache_size	4194304
l2s_count	32
limits_max_cta_per_sm	16
local_l1t_cache_supported	1
managed_memory	1
max_access_policy_window_size	0
max_block_dim_x	1024
max_block_dim_y	1024
max_block_dim_z	64
max_blocks_per_multiprocessor	16
max_gpu_frequency_khz	1440000
max_grid_dim_x	2147483647
max_grid_dim_y	65535
max_grid_dim_z	65535
max_ipc_per_multiprocessor	4
max_ipc_per_scheduler	1
max_mem_frequency_khz	7001000
max_persisting_l2_cache_size	0
max_pitch	2147483647
max_registers_per_block	65536
max_registers_per_multiprocessor	65536
max_registers_per_thread	255
max_shared_memory_per_block	49152
max_shared_memory_per_block_optin	65536
max_shared_memory_per_multiprocessor	65536
max_threads_per_block	1024
max_threads_per_multiprocessor	1024
max_warp_per_multiprocessor	32
max_warp_per_scheduler	8
maximum_surface_id_layered_layers	2048

Figure 2.15 Device details cont

maximum_surface_id_layered_width	32768
maximum_surface_id_width	32768
maximum_surface_id_height	65536
maximum_surface_id_layered_height	32768
maximum_surface_id_layered_layers	2048
maximum_surface_id_layered_width	32768
maximum_surface_id_width	131072
maximum_surface_id_depth	16384
maximum_surface_id_height	16384
maximum_surface_id_width	16384
maximum_surfacecubemap_layered_layers	2046
maximum_surfacecubemap_layered_width	32768
maximum_surfacecubemap_width	32768
maximum_texture_id_layered_layers	2048
maximum_texture_id_layered_width	32768
maximum_texture_id_linear_width	268435456
maximum_texture_id_mipmapped_width	32768
maximum_texture_id_width	131072
maximum_texture2d_gather_height	32768
maximum_texture2d_gather_width	32768
maximum_texture2d_height	65536
maximum_texture2d_layered_height	32768
maximum_texture2d_layered_layers	2048
maximum_texture2d_layered_width	32768
maximum_texture2d_linear_height	65000
maximum_texture2d_linear_pitch	2097120
maximum_texture2d_linear_width	131072
maximum_texture2d_mipmapped_height	32768
maximum_texture2d_mipmapped_width	32768
maximum_texture2d_width	131072
maximum_texture3d_depth	16384
maximum_texture3d_depth_alternate	32768
maximum_texture3d_height	16384
maximum_texture3d_height_alternate	8192
maximum_texture3d_width	16384
maximum_texture3d_width_alternate	8192
maximum_texturecubemap_layered_layers	2046
maximum_texturecubemap_layered_width	32768
maximum_texturecubemap_width	32768
memory_clock_rate	7001000
multi_gpu_board	0

Figure 2.16 Device details cont

multi_gpu_board_group_id	0
multiprocessor_count	36
num_128_per_ipb	8
numSchedulers_per_multiprocessor	4
numTex_per_multiprocessor	1
pageable_memory_access	0
pageable_memory_access_uses_host_page_tables	0
pci_bus_id	1
pci_device_id	0
pci_domain_id	0
ram_location	1
ram_type	17
reserved_shared_memory_per_block	0
sass_level	7
single_to_double_precision_perf_ratio	32
stream_priorities_supported	1
surface_alignment	512
tcc_driver	0
texture_alignment	512
texture_pitch_alignment	32
total_constant_memory	65536
total_memory	8370061312
unified_addressing	1
virtual_address_management_supported	1
warp_size	32

Figure 2.17 Device details cont

## 2.0.3 PTX

# Address	Source	Live Registers	Sampling Data (All)	Sampling Data (Not Issued)	Instructions Executed	at-on Thread	Avg Thread Executed	Divergent Branches	Address Space	Access Operation	Access Size
<b>FastRandomWalkCUDAKernel</b>											
1 00007f38 56f96b00	IMAD.MOV.U32 R1, RZ, RZ, c[0x0][0x28]	4	3	1	8.192	262.144	32	0	0	0	32
2 00007f38 56f96b10	ISETP.NE.U32,AND R0, PT, RZ, c[0x0][0x100], PT	4	4	0	8.192	262.144	32	0	0	0	32
3 00007f38 56f96b20	ISETP.NE.AND.EX PO, PT, RZ, c[0x0][0x164], PT, PO	4	0	0	8.192	262.144	32	0	0	0	32
4 00007f38 @1PO EXIT	S2R R7, SR_TID.X	4	0	0	8.192	0	0	0	0	0	0
5 00007f38 56f96b40	IMAD.MOV.U32 R8, RZ, RZ, c[0x0][0x198]	4	0	0	8.192	262.144	32	0	0	0	32
7 00007f38 56f96b60	ULDC UR4, c[0x0][0x178]	4	0	0	8.192	262.144	32	0	0	0	32
8 00007f38 56f96b70	IMAD.MOV.U32 R3, RZ, RZ, 0x8	5	0	0	8.192	262.144	32	0	0	0	32
9 00007f38 56f96b80	S2R R0, SR_CTAID.X	6	1	0	8.192	262.144	32	0	0	0	32
10 00007f38 56f96b90	UIADD3 UR4, UP0, UR4, 0x1, URZ	6	0	0	8.192	262.144	32	0	0	0	32
11 00007f38 56f96ba0	IADD3 R9, R8, -0x1, RZ	7	0	0	8.192	262.144	32	0	0	0	32
12 00007f38 56f96bb0	IMAD.MOV.U32 R14, RZ, RZ, RZ	8	0	0	8.192	262.144	32	0	0	0	32
13 00007f38 56f96bc0	SHF.R.S32.HI R8, RZ, 0x1f, R8	8	0	0	8.192	262.144	32	0	0	0	32
14 00007f38 56f96bd0	IMAD.MOV.U32 R12, RZ, RZ, RZ	9	0	0	8.192	262.144	32	0	0	0	32
15 00007f38 56f96be0	ULDC UR5, c[0x0][0x17c]	9	0	0	8.192	262.144	32	0	0	0	32
16 00007f38 56f96bf0	IMAD.MOV.U32 R10, RZ, RZ, RZ	10	0	0	8.192	262.144	32	0	0	0	32
17 00007f38 56f96c00	UIADD3 X UR5, URZ, UR5, URZ, UP0, IUPT	10	0	0	8.192	262.144	32	0	0	0	32
18 00007f38 56f96c10	IMAD R7, R0, c[0x0][0x01], R7	10	5	1	8.192	262.144	32	0	0	0	32
19 00007f38 56f96c20	IMAD R2, R7, 0x3, RZ	10	0	0	8.192	262.144	32	0	0	0	32
20 00007f38 56f96c30	IMM R7, R7, c[0x0][0x19c], RZ	10	1	0	8.192	262.144	32	0	0	0	32
21 00007f38 56f96c40	IMAD.WIDE R2, R2, R3, c[0x0][0x1a0]	10	0	0	8.192	262.144	32	0	0	0	32
22 00007f38 56f96c50	IADD3 R6, PO, R7, R14, RZ	11	153	62	819.200	26.214.400	32	0	0	0	32
23 00007f38 56f96c60	BMOV.32_CLEAR RZ, B0	11	17	0	819.200	26.214.400	32	0	0	0	32
24 00007f38 56f96c70	BSSY B0, 0x7f3856f97559	11	16	0	819.200	26.214.400	32	0	0	0	32
25 00007f38 56f96c80	IMAD.MOV.U32 R4, RZ, RZ, 0x7	12	20	1	819.200	26.214.400	32	0	0	0	32
26 00007f38 56f96c90	LEA.HI.X.S32.R3, R7, R12, 0x1, PO	13	0	0	819.200	26.214.400	32	0	0	0	32
27 00007f38 56f96ca0	IMAD.MOV.U32 R5, RZ, RZ, RZ	14	188	85	819.200	26.214.400	32	0	0	0	32
28 00007f38 56f96cb0	ULDC UR4, c[0x0][0x178]	14	1.584	570	11.684.464	287.649.750	24	0	0	0	32
29 00007f38 56f96cc0	LDS.E.US.SYS R15, [UR6]	15	8.108	5.743	11.684.464	287.649.750	24	0	0	0	32
30 00007f38 56f96cd0	LOP3.LUT R13, R4, 0xffff, RZ, 0xc0, !PT	16	300	69	11.684.464	287.649.750	24	0	0	0	32
31 00007f38 56f96ce0	RMD.32_CLEAR R7, B1	16	196	2	11.684.464	287.649.750	24	0	0	0	32
32 00007f38 56f96cf0	BSSY B1, 0x7f3856f96e50	16	196	0	11.684.464	287.649.750	24	0	0	0	32
PRMT_R10_R11_R12_R13_R14_R15_R16_R17_R18_R19_R20_R21											

# Address	Source	Live Registers	Sampling Data (All)	Sampling Data (Not Issued)	Instructions Executed	at-on Thread	Avg Thread Executed	Divergent Branches	Address Space	Access Operation	Access Size
<b>FastRandomWalkCUDAKernel</b>											
33 00007f38 56f96d00	PRMT R18, R13, 0x9910, RZ	17	687	82	11.684.464	287.649.750	24	0	0	0	32
34 00007f38 56f96d10	PRMT R13, R15, 0x9910, RZ	17	20.311	12.991	11.684.464	287.649.750	24	0	0	0	32
35 00007f38 56f96d20	ISETP.NE.AND.PO, PT, R13, R18, PT	17	1.328	385	11.684.464	287.649.750	24	0	0	0	32
36 00007f38 56f96d30	IADD3 R12, R13, RZ, RZ, c[0x0][0x178]	17	171	0	11.684.464	287.649.750	24	0	0	0	32
37 00007f38 56f96d40	@1PO BRA 0x7f3856f96e40	17	2.254	778	11.684.464	0	0	0	0	0	32
38 00007f38 56f96d50	IMAD.U32 R16, RZ, RZ, UR4	17	1.086	372	11.684.464	287.649.750	24	0	0	0	32
39 00007f38 56f96d60	IMAD.U32 R17, RZ, RZ, UR5	18	434	70	11.684.464	287.649.750	24	0	0	0	32
40 00007f38 56f96d70	IMAD.MOV.U32 R19, RZ, RZ, c[0x0][0x178]	19	457	65	11.684.464	287.649.750	24	0	0	0	32
41 00007f38 56f96d80	PRMT R19, R15, 0x9910, RZ	20	99.446	36.576	876.282.222	5.995.224.188	18	0	0	0	32
42 00007f38 56f96d90	ISETP.NE.AND.PO, PT, R13, RZ, PT	20	109.188	29.140	876.282.222	5.995.224.188	18	0	0	0	32
43 00007f38 56f96da0	IMAD.MOV.U32 R13, RZ, RZ, RZ	20	14.340	0	876.282.222	5.995.224.188	18	0	0	0	32
44 00007f38 56f96db0	@1PO BRA 0x7f3856f96e40	17	1.459	54.075	876.282.222	0	0	0	0	0	32
45 00007f38 56f96dc0	LDS.E.US.SYS R15, [R16]	20	356.113	213.111	876.282.222	5.995.224.188	18	0	0	0	32
46 00007f38 56f96dd0	IADD3 R19, R19, 0x1, RZ	20	14.661	0	876.282.222	5.995.224.188	18	0	0	0	32
47 00007f38 56f96de0	IADD3 R16, P1, R16, 0x1, R2	20	284.787	109.807	876.282.222	5.995.224.188	18	0	0	0	32
48 00007f38 56f96df0	IMAD.X.R17, R17, R17, R17	20	92.970	25.954	876.282.222	5.995.224.188	18	0	0	0	32
49 00007f38 56f96e00	PRM.R13, R15, 0x9910, RZ	20	506.019	233.027	876.282.222	5.995.224.188	18	0	0	0	32
50 00007f38 56f96e10	ISETP.NE.AND.PO, PT, R13, R18, PT	20	104.477	26.116	876.282.222	5.995.224.188	18	0	0	0	32
51 00007f38 56f96e20	@1PO BRA 0x7f3856f96d80	20	189.644	63.081	876.282.222	5.707.574.438	17	130.300.867	0	0	0
52 00007f38 56f96e30	IMAD.MOV.U32 R13, RZ, RZ, RZ	16	66.243	17.746	141.490.440	287.649.750	1	0	0	0	32
53 00007f38 56f96e40	BSYNC B1	15	21.174	6.041	141.490.440	287.649.750	1	0	0	0	32
54 00007f38 56f96e50	LDS.E.64.SYS R16, [R2]	17	29.214	12.084	11.684.464	287.649.750	24	0	0	0	64
55 00007f38 56f96e60	LDS.E.64.SYS R20, [R2+0x8]	19	3.706	1.995	11.684.464	287.649.750	24	0	0	0	64
56 00007f38 56f96e70	LDS.E.64.SYS R22, [R2+0x10]	21	4.206	2.322	11.684.464	287.649.750	24	0	0	0	64
57 00007f38 56f96e80	IADD3 R28, R13, -c[0x0][0x178], RZ	22	205	0	11.684.464	287.649.750	24	0	0	0	32
58 00007f38 56f96e90	LEA.R18, P0, R28, c[0x0][0x1801], 0x3	23	1.678	368	11.684.464	287.649.750	24	0	0	0	32
59 00007f38 56f96e90	IMAD.U32 R15, R16, 0x10000, RZ	24	126.520	56.245	11.684.464	287.649.750	24	0	0	0	32
60 00007f38 56f96e90	SHF.L.64.HI R19, R16, 0x10, R17	25	214	0	11.684.464	287.649.750	24	0	0	0	32
61 00007f38 56f96e90	STO.E.64.SYS [R2], R20	25	10.863	6.815	11.684.464	287.649.750	24	0	0	0	64
62 00007f38 56f96e90	LOP3.LUT R16, R17, R17, R17, R17, R17, R17, R17, R17	25	787	202	11.684.464	287.649.750	24	0	0	0	32
63 00007f38 56f96e90	LOP3.LUT R17, R17, R17, R17, R17, R17, R17, R17, R17	24	437	108	11.684.464	287.649.750	24	0	0	0	32
64 00007f38 56f96e90	STG.E.64.SYS [R2+0x8], R22	23	7.406	4.429	11.684.464	287.649.750	24	0	0	0	64
65 00007f38 56f96f00	SHF.R.S32.HI R15, RZ, 0x1f, R28	24	196	0	11.684.464	287.649.750	24	0	0	0	32

# Address	Source	Live Registers	Sampling Data (All)	Sampling Data (Not Issued)	Instructions Executed	at-on Thread	Avg Thread Executed	Divergent Branches	Address Space	Access Operation	Access Size
66 00007f38 56f96f10	SHF.R.U64 R25, R16, 0x5, R17	29	1.483	557	11.684.464	287.649.750	24				
67 00007f38 56f96f20	SHF.R.U32.HI R26, R2, 0x3, R17	26	485	153	11.684.464	287.649.750	24				
68 00007f38 56f96f30	IOP3.LUT R25, R25, R16, R2, 0x3c, IPT	26	471	120	11.684.464	287.649.750	24				
69 00007f38 56f96f40	IOP3.LUT R26, R26, R17, R2, 0x3c, IPT	25	461	134	11.684.464	287.649.750	24				
70 00007f38 56f96f50	IOP3.LUT R16, R20, R2, RZ, 0x3c, IPT	25	477	156	11.684.464	287.649.750	24				
71 00007f38 56f96f60	IMAD.SHL.U32 R19, R25, 0x2, RZ	24	202	0	11.684.464	287.649.750	24				
72 00007f38 56f96f70	IOP3.LUT R17, R21, R23, RZ, 0x3c, IPT	25	201	0	11.684.464	287.649.750	24				
73 00007f38 56f96f80	SHF.L.U64.HI R24, R25, 0x1, R26	24	1.889	481	11.684.464	287.649.750	24				
74 00007f38 56f96f90	IOP3.LUT R16, R16, R19, R25, 0x06, IPT	24	427	121	11.684.464	287.649.750	24				
75 00007f38 56f96fb0	LEA.H.IX R19, R28, C[0x0][0x184], R15, 0x3, P0	21	445	142	11.684.464	287.649.750	24				
77 00007f38 56f96fc0	STG.E.64.SYS [R2+0x10], R16	19	10.643	6.778	11.684.464	287.649.750	24				64
78 00007f38 56f96fd0	LDC.E.64.SYS R18, [R18]	19	5.739	3.464	11.684.464	287.649.750	24				64
79 00007f38 56f96fe0	BMOV.32.CLEAR RZ, B1	19	201	0	11.684.464	287.649.750	24				
80 00007f38 56f96ff0	BSSY B1, 0x7f3856f97190	19	233	1	11.684.464	287.649.750	24				
81 00007f38 56f97000	IOP3.LUT R15, R17, R19, R2, 0x0c, IPT	20	21.874	11.248	11.684.464	287.649.750	24				
82 00007f38 56f97010	IOP3.LUT R17, R17, R24, R6, 0x06, IPT	19	1.267	375	11.684.464	287.649.750	24				
83 00007f38 56f97020	@P0 BRA 0x7f3856f97060	18	2.539	1.016	11.684.464	6	0	6			
84 00007f38 56f97030	MOV R20, 0x550	18	1.118	404	11.684.464	287.649.744	24				
85 00007f38 56f97040	CALL.REL.NOIND 0x7f3856f97610	17	1.763	702	11.684.464	287.649.744	24				
86 00007f38 56f97050	BRA 0x7f3856f97180	17	2.548	1.016	11.684.464	287.649.744	24				
87 00007f38 56f97060	I2F.U32.RP R15, R18	18	767	218	6	6	1				
88 00007f38 56f97070	IMAD.MOV R17, RZ, RZ, -R18	19	0	0	6	6	1				
89 00007f38 56f97080	ISETP.NE.U32.AND P1, PT, R18, RZ, PT	19	0	0	6	6	1				
90 00007f38 56f97090	MUFU.RCP R15, R15	19	0	0	6	6	1				
91 00007f38 56f97090	IADD3 R20, R15, 0xfffffe, RZ	20	0	0	6	6	1				
92 00007f38 56f97080	F2I.FTZ.U32.TRUNC.NTZ R21, R20	20	0	0	6	6	1				
93 00007f38 56f97080	IMAD.MOV.U32 R20, RZ, RZ, RZ	20	0	0	6	6	1				
94 00007f38 56f97060	IMAD R17, R17, R21, R2	20	0	0	6	6	1				
95 00007f38 56f970e0	IMAD.HI.U32 R21, R21, R17, R20	20	0	0	6	6	1				
96 00007f38 56f970f0	IMAD.MOV.U32 R17, RZ, RZ	19	0	0	6	6	1				
97 00007f38 56f97100	IMAD.HI.U32 R21, R21, R16, R2	19	0	0	6	6	1				
98 00007f38 56f97110	IMAD.MOV R21, RZ, RZ, -R21	19	0	0	6	6	1				

# Address	Source	Live Registers	Sampling Data (All)	Sampling Data (Not Issued)	Instructions Executed	at-on Thread	Avg Thread Executed	Divergent Branches	Address Space	Access Operation	Access Size
99 00007f38 56f97120	IMAD.R16, R18, R21, R16	19	0	0	6	6	1				
100 00007f38 56f97130	ISETP.GE.U32.AND P0, PT, R16, R18, PT	18	0	0	6	6	1				
101 00007f38 56f97140	@P0 IMAD.IADD R16, R16, 0x1, -R18	18	0	0	6	0	0				
102 00007f38 56f97150	ISETP.GE.AND P0, PT, R16, R18, PT	18	0	0	6	6	1				
103 00007f38 56f97160	@P0 IMAD.IADD R16, R16, 0x1, -R18	18	0	0	6	0	0				
104 00007f38 56f97170	@P1 IOP3.LUT R16, R2, R18, RZ, 0x33, IPT	18	0	0	6	0	0				
105 00007f38 56f97180	BSYNC B1	17	2.039	892	11.684.470	287.649.750	24				
106 00007f38 56f97190	ISETP.GE.AND P0, PT, R0, PT, R0, 0x1, PT	17	2.707	1.008	11.684.464	287.649.750	24				
107 00007f38 56f971a0	@P0 BRA 0x7f3856f97400	17	2.528	922	11.684.464	0	0	0			
108 00007f38 56f971b0	IADD3 R15, R13, -c[0x0][0x178], RZ	18	1.068	355	11.684.464	287.649.750	24				
109 00007f38 56f971c0	IMAD.MOV.U32 R20, R18, RZ, RZ, 0x190	18	184	0	11.684.464	287.649.750	24				
110 00007f38 56f971d0	BMOV.32.CLEAR RZ, B1	18	195	0	11.684.464	287.649.750	24				
111 00007f38 56f971e0	IMAD.MOV.U32 R21, R18, RZ, 0xfffffe, RZ	19	167	0	11.684.464	287.649.750	24				
112 00007f38 56f971f0	SHF.R.S32.HI R22, RZ, RZ, 0x1f, R15	20	194	0	11.684.464	287.649.750	24				
113 00007f38 56f97200	IMAD.MOV.U32 R19, R19, RZ, RZ, 0x8	21	175	2	11.684.464	287.649.750	24				
114 00007f38 56f97210	BSSY B1, 0x7f3856f97400	21	159	0	11.684.464	287.649.750	24				
115 00007f38 56f97220	IMAD.R18, R15, c[0x0][0x198], RZ	22	192	1	11.684.464	287.649.750	24				
116 00007f38 56f97230	IMAD.R19, R8, R15, R15, R15	23	507	33	11.684.464	287.649.750	24				
117 00007f38 56f97240	IMAD.WIDE.U32 R20, R15, c[0x0][0x198], R20	23	400	41	11.684.464	287.649.750	24				
118 00007f38 56f97250	IMAD.WIDE.R18, R19, R19, c[0x0][0x188]	22	1.434	192	11.684.464	287.649.750	24				
119 00007f38 56f97260	IMAD.R13, R22, c[0x0][0x198], R13	22	1.471	239	11.684.464	287.649.750	24				
120 00007f38 56f97270	IMAD.MOV.U32 R15, RZ, RZ, R20	22	735	75	11.684.464	287.649.750	24				
121 00007f38 56f97280	IMAD.IADD R24, R24, 0x1, R13	22	481	49	11.684.464	287.649.750	24				
122 00007f38 56f97290	IMAD.MOV.U32 R13, R13, RZ, R18	21	433	58	11.684.464	287.649.750	24				
123 00007f38 56f972a0	IMAD.MOV.U32 R18, R18, R18, R18	21	480	81	11.684.464	287.649.750	24				
124 00007f38 56f972b0	IMAD.MOV.U32 R20, RZ, RZ, RZ	21	474	66	11.684.464	287.649.750	24				
125 00007f38 56f972c0	IMAD.MOV.U32 R18, RZ, RZ, R13	22	18.654	6.950	17.2403.954	1.902.672.000	11				
126 00007f38 56f972d0	IMAD.MOV.U32 R19, RZ, RZ, R22	23	6.550	1.097	17.2403.954	1.902.672.000	11				
127 00007f38 56f972e0	LDC.E.64.SYS R18, [R18]	23	114.75	66.549	17.2403.954	1.902.672.000	11				
128 00007f38 56f972f0	IADD3 R16, R0, -R18, R18, R16	23	311.537	152.671	17.2403.954	1.902.672.000	11				
129 00007f38 56f97300	IMAD.X R17, R17, 0x1, -R19, R0	22	17.804	5.596	17.2403.954	1.902.672.000	11				
130 00007f38 56f97310	ISETP.NE.U32.AND P0, PT, R16, RZ, PT	21	2.876	0	17.2403.954	1.902.672.000	11				
131 00007f38 56f97320	ISETP.GE.AND P0, PT, R17, RZ, PT, P0	21	22.174	5.962	17.2403.954	1.902.672.000	11				

# Address	Source	Live Registers	Sampling Data (All)	Sampling Data (Not Issued)	Instructions Executed	at-on Thread	Avg Thread Executed	Divergent Branches	Address Space	Access Operation	Access Size
132 00007f38 56f97300	@P0 BRA 0x7f3856f973c0	21	36.388	15.144	172.403.640	287.649.750	24	104.391.323			
133 00007f38 56f97340	IADD3 R20, R20, 0x1, RZ	21	24.536	9.068	160.719.550	1.615.022.310	10				
134 00007f38 56f97350	IADD3 R15, P1, R15, 0x1, RZ	21	6.096	1.530	160.719.550	1.615.022.310	10				
135 00007f38 56f97360	ISETP.GE.AND P0, PT, R0, PT, R0, R9, PT	21	6.332	1.587	160.719.550	1.615.022.310	10				
136 00007f38 56f97370	IADD3 R13, P2, R13, 0x8, RZ	21	6.376	1.805	160.719.550	1.615.022.310	10				
137 00007f38 56f97380	IMAD.X R24, R2, RZ, R24, P1	21	2.671	30	160.719.550	1.615.022.310	10				
138 00007f38 56f97390	IMAD.X R22, R22, RZ, R2, R2, P2	21	14.497	2.222	160.719.550	1.615.022.310	10				
139 00007f38 56f973a0	@P0 BRA 0x7f3856f972c0	21	16.604	4.259	160.719.550	1.615.022.310	10				
140 00007f38 56f973d0	IMAD.MOV.U32 R16, RZ, RZ, R15	14	10.624	2.949	0	0	0	0			
141 00007f38 56f973e0	IMAD.MOV.U32 R16, R1, PT, R16, R16	16	12.297	5.673	112.931.450	287.649.750	24				
142 00007f38 56f973f0	IMAD.MOV.U32 R17, RZ, R24, R24	16	4.465	883	112.931.450	287.649.750	24				
143 00007f38 56f973f0	LDC.E.US.B.SYS R11, [R16]	16	7.527	44.12	112.931.450	287.					

# Address	Source	Live Registers	Sampling Data (All)	Sampling Data (Not Issued)	Instructions Executed	at-On Thread	Avg Thread Executed	Divergent Branches	Address Space	Access Operation	Access Size
165 00007f38 56f97540	B SYNC_B0	13	1.160	332	2,747,966	26,214,400	9				
166 00007f38 56f97550	IMAD.MOV.U32 R4, RZ, RZ, 0xa	14	1.079	455	819,200	26,214,400	32				
167 00007f38 56f97560	IADD R10, R10, 0x1, RZ	14	14	0	819,200	26,214,400	32				
168 00007f38 56f97570	IADD R5, R5, 0x1, RZ	14	31	6	819,200	26,214,400	32				
169 00007f38 56f97580	ISETP.GE.U32.AND P0, PT, R10, c[0x0][0x1601], PT	14	33	9	819,200	26,214,400	32				
170 00007f38 56f97590	SHF.R.S32.NI R0, RZ, 0x1f, R10	15	29	8	819,200	26,214,400	32				
171 00007f38 56f975a0	STG.E.UB.SYS [R16], R4	15	330	212	819,200	26,214,400	32			Global	Store
172 00007f38 56f975b0	IADD R14, P1, R5, R14, RZ	14	14	0	819,200	26,214,400	32				
173 00007f38 56f975c0	ISETP.GE.U32.AND EX P0, PT, R0, c[0x0][0x164], PT	14	157	36	819,200	26,214,400	32				
174 00007f38 56f975d0	LEA.HI,X.SX32 R12, R5, R12, 0x1, P1	13	35	8	819,200	26,214,400	32				
175 00007f38 56f975e0	EIP# BRA 0x7f3856f96c50	12	190	82	819,200	25,952,256	31	0			
176 00007f38 56f975f0	STG.E.UB.SYS [R16+0x1], RZ	5	52	14	8,192	262,144	32			Global	Store
177 00007f38 56f97600	EXTT	1	1	0	8,192	262,144	32				
178 00007f38 56f97610	I2F.U64.RP R15, R18	1	1,641	767	11,684,464	287,649,744	24				
179 00007f38 56f97620	MVFU.RCP R15, R16	1	8,356	3,739	11,684,464	287,649,744	24				
180 00007f38 56f97630	IADD R22, R15, 0x1fffffe, RZ	4	3,209	1,254	11,684,464	287,649,744	24				
181 00007f38 56f97640	F2I.U64.TRUNC R12, R22	1	1,386	431	11,684,464	287,649,744	24				
182 00007f38 56f97650	IMAD.WIDE.U32 R24, R22, R18, RZ	4	8,611	3,881	11,684,464	287,649,744	24				
183 00007f38 56f97660	IMAD.R21, R22, R19, R25	5	1,956	606	11,684,464	287,649,744	24				
184 00007f38 56f97670	IADD R27, P0, RZ, -R24, RZ	5	181	0	11,684,464	287,649,744	24				
185 00007f38 56f97680	IMAD.MOV.U32 R25, RZ, RZ, R22	5	203	0	11,684,464	287,649,744	24				
186 00007f38 56f97690	IMAD.R21, R23, R18, R21	5	504	61	11,684,464	287,649,744	24				
187 00007f38 56f976a0	IMAD.HI.U32 R24, R22, R27, RZ	6	480	63	11,684,464	287,649,744	24				
188 00007f38 56f976b0	IMAD.X R21, R2, R23, P0	6	1,468	237	11,684,464	287,649,744	24				
189 00007f38 56f976c0	IMAD.WIDE.U32 R24, P0, R22, R21, R24	6	4,408	218	11,684,464	287,649,744	24				
190 00007f38 56f976d0	IMAD.HI.U32 R22, R23, R21, RZ	6	1,327	231	11,684,464	287,649,744	24				
191 00007f38 56f976e0	IMAD.HI.U32 R24, P1, R23, R27, R24	6	1,697	244	11,684,464	287,649,744	24				
192 00007f38 56f976f0	IMAD.R27, R23, R21, RZ	5	1,524	225	11,684,464	287,649,744	24				
193 00007f38 56f97700	IMAD.X R22, R23, R1x, R23, P0	4	771	61	11,684,464	287,649,744	24				
194 00007f38 56f97710	IMAD.MOV.U32 R23, RZ, RZ, RZ	4	479	69	11,684,464	287,649,744	24				
195 00007f38 56f97720	IADD R27, P2, R27, R24, RZ	4	198	0	11,684,464	287,649,744	24				
196 00007f38 56f97730	IADD.X R15, RZ, RZ, R22, P2, P1	4	1,833	381	11,684,464	287,649,744	24				
197 00007f38 56f97740	IMAD.WIDE.U32 R24, R27, R18, RZ	5	181	0	11,684,464	287,649,744	24				

# Address	Source	Live Registers	Sampling Data (All)	Sampling Data (Not Issued)	Instructions Executed	at-On Thread	Avg Thread Executed	Divergent Branches	Address Space	Access Operation	Access Size
198 00007f38 56f97750	IMAD.R21, R27, R19, R25	6	2,029	536	11,684,464	287,649,744	24				
199 00007f38 56f97760	IADD R25, P0, RZ, -R24, RZ	6	217	0	11,684,464	287,649,744	24				
200 00007f38 56f97770	IMAD.R11, R15, R18, R21	5	975	159	11,684,464	287,649,744	24				
201 00007f38 56f97780	IMAD.HI.U32 R26, R27, R25, RZ	6	540	58	11,684,464	287,649,744	24				
202 00007f38 56f97790	IMAD.X R22, R26, R27, R21, P0	7	1,444	222	11,684,464	287,649,744	24				
203 00007f38 56f977a0	IMAD.WIDE.U32 R26, P0, R27, R22, R26	6	1,383	241	11,684,464	287,649,744	24				
204 00007f38 56f977b0	IMAD.R24, R15, R22, RZ	7	1,388	230	11,684,464	287,649,744	24				
205 00007f38 56f977c0	IMAD.HI.U32 R21, P1, R15, R16, R26	8	1,483	213	11,684,464	287,649,744	24				
206 00007f38 56f977d0	IADD R21, P2, R24, R21, RZ	8	2,784	807	11,684,464	287,649,744	24				
207 00007f38 56f977e0	IMAD.HI.U32 R24, R22, R2, RZ	8	206	0	11,684,464	287,649,744	24				
208 00007f38 56f977f0	IMAD.HI.U32 R22, R21, R16, RZ	8	1,296	213	11,684,464	287,649,744	24				
209 00007f38 56f97800	IMAD.X R15, R24, R1x, R15, P0	8	1,577	243	11,684,464	287,649,744	24				
210 00007f38 56f97810	IMAD.WIDE.U32 R22, R21, R17, R22	8	1,313	250	11,684,464	287,649,744	24				
211 00007f38 56f97820	IADD.X R15, RZ, R2, R15, P2, P1	7	209	0	11,684,464	287,649,744	24				
212 00007f38 56f97830	IMAD.HI.U32 R20, P0, R15, R16, R16	8	2,156	659	11,684,464	287,649,744	24				
213 00007f38 56f97840	IMAD.R22, R15, R17, RZ	7	1,294	247	11,684,464	287,649,744	24				
214 00007f38 56f97850	IMAD.HI.U32 R15, R15, R17, RZ	7	727	82	11,684,464	287,649,744	24				
215 00007f38 56f97860	IADD.R21, P1, R22, R21, RZ	7	1,749	317	11,684,464	287,649,744	24				
216 00007f38 56f97870	IMAD.X R15, RZ, R2, R15, P0	6	1,179	280	11,684,464	287,649,744	24				
217 00007f38 56f97880	IMAD.WIDE.U32 R22, R21, R18, RZ	8	644	60	11,684,464	287,649,744	24				
218 00007f38 56f97890	IMAD.R21, R21, R19, R23	8	205	0	11,684,464	287,649,744	24				
219 00007f38 56f978a0	IMAD.R21, R21, R19, R23	8	729	57	11,684,464	287,649,744	24				
220 00007f38 56f978b0	IADD.R23, P1, R22, R16, RZ	8	934	205	11,684,464	287,649,744	24				
221 00007f38 56f978c0	IMAD.R15, R15, R18, R21	8	934	139	11,684,464	287,649,744	24				
222 00007f38 56f978d0	ISETP.GE.U32.AND P0, PT, R23, R18, PT	7	216	0	11,684,464	287,649,744	24				
223 00007f38 56f978e0	IMAD.X R17, R17, 0x1, -R15, P1	8	922	149	11,684,464	287,649,744	24				
224 00007f38 56f978f0	IADD.R15, P1, R17, 0x1, -R15, P1	8	284	0	11,684,464	287,649,744	24				
225 00007f38 56f97900	ISETP.GE.U32.AND EX P0, PT, R17, R19, PT, P0	9	1,874	449	11,684,464	287,649,744	24				
226 00007f38 56f97910	IMAD.X R21, R17, 0x1, -R19, P1	9	197	0	11,684,464	287,649,744	24				
227 00007f38 56f97920	ISETP.NE.U32.AND P0, PT, R17, R19, PT, P0	9	217	0	11,684,464	287,649,744	24				
228 00007f38 56f97930	SEL.R15, R15, R13, P0	9	528	95	11,684,464	287,649,744	24				
229 00007f38 56f97940	SEL.R21, R21, R17, P0	9	479	135	11,684,464	287,649,744	24				
230 00007f38 56f97950	IADD.R16, P2, -R18, R15, RZ	9	508	136	11,684,464	287,649,744	24				
231 00007f38 56f97960	ISETP.GE.U32.AND EX P0, PT, R15, R18, PT	9	472	145	11,684,464	287,649,744	24				
232 00007f38 56f97970	ISETP.NE.AND EX P1, PT, R19, RZ, PT, P1	9	1,790	467	11,684,464	287,649,744	24				
233 00007f38 56f97980	IMAD.X R17, R21, 0x1, -R19, P2	10	199	0	11,684,464	287,649,744	24				
234 00007f38 56f97990	ISETP.GE.U32.AND EX P0, PT, R21, R19, PT, P0	10	185	0	11,684,464	287,649,744	24				
235 00007f38 56f979a0	SEL.R17, R17, R21, P0	10	1,358	367	11,684,464	287,649,744	24				
236 00007f38 56f979b0	IMAD.MOV.U32 R21, R2, RZ, 0x0	10	192	0	11,684,464	287,649,744	24				
237 00007f38 56f979c0	SEL.R16, R16, R15, P0	10	201	0	11,684,464	287,649,744	24				
238 00007f38 56f979d0	SEL.R17, R17, 0xffffffff, P1	9	484	115	11,684,464	287,649,744	24				
239 00007f38 56f979e0	SEL.R16, R16, 0xffffffff, P1	9	422	123	11,684,464	287,649,744	24				
240 00007f38 56f979f0	RET.REL.NODEC R20 0x7f3856f96b00	9	192	0	11,684,464	287,649,744	24	0			
241 00007f38 56f97a00	BRA 0x7f3856f97a00	0	7,516	3,626							
242 00007f38 56f97a10	NOP		0	0	0	0	0				
243 00007f38 56f97a20	NOP		0	0	0	0	0				
244 00007f38 56f97a30	NOP		0	0	0	0	0				
245 0											

## 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:

markopy	.....	27
Markov	Namespace for the markov-model related classes. Contains <a href="#">Model</a> , <a href="#">Node</a> and <a href="#">Edge</a> classes	31
Markov::API	Namespace for the <a href="#">MarkovPasswords API</a>	31
Markov::API::CLI	Structure to hold parsed cli arguements	32
Markov::API::Concurrency	Namespace for <a href="#">Concurrency</a> related classes	33
Markov::API::CUDA	Namespace for objects requiring <a href="#">CUDA</a> libraries	33
Markov::API::CUDA::Random	Namespace for <a href="#">Random</a> engines operable under <a href="#">device</a> space	35
Markov::GUI	Namespace for <a href="#">MarkovPasswords API</a> GUI wrapper	36
Markov::Markopy	.....	36
Markov::Random	Objects related to RNG	37
model_2gram	.....	38
random	.....	38
random-model	.....	38
Testing	Namespace for Microsoft Native Unit <a href="#">Testing</a> Classes	39
Testing::MarkovModel	Testing namespace for <a href="#">MarkovModel</a>	39
Testing::MarkovPasswords	Testing namespace for <a href="#">MarkovPasswords</a>	42
Testing::MVP	Testing Namespace for Minimal Viable Product	43
Testing::MVP::MarkovModel	Testing Namespace for <a href="#">MVP MarkovModel</a>	43
Testing::MVP::MarkovPasswords	Testing namespace for <a href="#">MVP MarkovPasswords</a>	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 . . . . .	55
Markov::API::CUDA::CUDADeviceController . . . . .	64
Markov::API::CUDA::CUDAModelMatrix . . . . .	71
Markov::API::CUDA::Random::Marsaglia . . . . .	148
Markov::Edge< NodeStorageType > . . . . .	113
Markov::Edge< char > . . . . .	113
Markov::Edge< storageType > . . . . .	113
Markov::Model< NodeStorageType > . . . . .	165
Markov::Model< char > . . . . .	165
Markov::API::MarkovPasswords . . . . .	122
Markov::API::ModelMatrix . . . . .	173
Markov::API::CUDA::CUDAModelMatrix . . . . .	71
Markov::Node< storageType > . . . . .	199
Markov::Node< char > . . . . .	199
Markov::Node< NodeStorageType > . . . . .	199
QMainWindow . . . . .	207
Markov::GUI::about . . . . .	54
Markov::GUI::CLI . . . . .	61
Markov::GUI::Generate . . . . .	118
Markov::GUI::MarkovPasswordsGUI . . . . .	139
Markov::GUI::menu . . . . .	158
Markov::GUI::Train . . . . .	216
Markov::Random::RandomEngine . . . . .	207
Markov::Random::DefaultRandomEngine . . . . .	108
Markov::Random::Marsaglia . . . . .	142
Markov::API::CUDA::Random::Marsaglia . . . . .	148
Markov::Random::Mersenne . . . . .	160
Markov::API::CLI::Terminal . . . . .	209
Markov::API::Concurrency::ThreadSharedListHandler . . . . .	212
char . . . . .	??
long int . . . . .	??



# 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
Markov::API::CLI::Argparse	Parse command line arguements . . . . .	55
Markov::GUI::CLI	QT CLI Class . . . . .	61
Markov::API::CUDA::CUDADeviceController	Controller class for CUDA device . . . . .	64
Markov::API::CUDA::CUDAModelMatrix	Extension of Markov::API::ModelMatrix which is modified to run on GPU devices . . . . .	71
Markov::Random::DefaultRandomEngine	Implementation using Random.h default random engine . . . . .	108
Markov::Edge< NodeStorageType >	Edge class used to link nodes in the model together . . . . .	113
Markov::GUI::Generate	QT Generation page class . . . . .	118
Markov::API::MarkovPasswords	Markov::Model with char represented nodes . . . . .	122
Markov::GUI::MarkovPasswordsGUI	Reporting UI . . . . .	139
Markov::Random::Marsaglia	Implementation of Marsaglia Random Engine . . . . .	142
Markov::API::CUDA::Random::Marsaglia	Extension of Markov::Random::Marsaglia which is capable o working on <b>device</b> space . . . . .	148
Markov::GUI::menu	QT Menu class . . . . .	158
Markov::Random::Mersenne	Implementation of Mersenne Twister Engine . . . . .	160
Markov::Model< NodeStorageType >	Class for the final Markov Model, constructed from nodes and edges . . . . .	165
Markov::API::ModelMatrix	Class to flatten and reduce Markov::Model to a Matrix . . . . .	173
Markov::Node< storageType >	A node class that for the vertices of model. Connected with eachother using Edge . . . . .	199
QMainWindow		207
Markov::Random::RandomEngine	An abstract class for Random Engine . . . . .	207
Markov::API::CLI::Terminal	Pretty colors for Terminal. Windows Only . . . . .	209
Markov::API::Concurrency::ThreadSharedListHandler	Simple class for managing shared access to file . . . . .	212
Markov::GUI::Train	QT Training page class . . . . .	216



# CHAPTER7

---

## File Index

---

### 7.1 File List

Here is a list of all files with brief descriptions:

<a href="#">about.cpp</a>	About page	221
<a href="#">about.h</a>	About page	221
<a href="#">argparse.cpp</a>	Arguement handler class for native CPP cli	223
<a href="#">argparse.h</a>	Arguement handler class for native CPP cli	224
<a href="#">CLI.cpp</a>	CLI page	229
<a href="#">CLI.h</a>	CLI page	230
<a href="#">cudaDeviceController.cu</a>	Simple static class for basic CUDA device controls	231
<a href="#">cudaDeviceController.h</a>	Simple static class for basic CUDA device controls	233
<a href="#">cudaModelMatrix.cu</a>	CUDA accelerated extension of <a href="#">Markov::API::ModelMatrix</a>	236
<a href="#">cudaModelMatrix.h</a>	CUDA accelerated extension of <a href="#">Markov::API::ModelMatrix</a>	240
<a href="#">cudarandom.h</a>	Extension of <a href="#">Markov::Random::Marsaglia</a> for CUDA	244
<a href="#">dllmain.cpp</a>	DLLMain for dynamic windows library	246
<a href="#">edge.h</a>	Edge class template	248
<a href="#">framework.h</a>	For windows dynamic library	251
<a href="#">Generate.cpp</a>	Generation Page	252
<a href="#">Generate.h</a>	Generation Page	254
<a href="#">MarkovAPICLI/src/main.cpp</a>	Test cases for <a href="#">Markov::API::ModelMatrix</a>	256
<a href="#">MarkovPasswordsGUI/src/main.cpp</a>	Entry point for GUI	258
<a href="#">main.cu</a>	Simple test file to check libcudamarkov	260
<a href="#">markopy.cpp</a>	CPython wrapper for libmarkov utils	261
<a href="#">markopy.py</a>		264
<a href="#">markovPasswords.cpp</a>	Wrapper for <a href="#">Markov::Model</a> to use with char represented models	266
<a href="#">markovPasswords.h</a>	Wrapper for <a href="#">Markov::Model</a> to use with char represented models	270
<a href="#">MarkovPasswordsGUI.cpp</a>	Main activity page for GUI	273

MarkovPasswordsGUI.h	
Main activity page for GUI . . . . .	274
menu.cpp	
Menu page . . . . .	276
menu.h	
Menu page . . . . .	277
model.h	
Model class template . . . . .	278
model_2gram.py . . . . .	284
modelMatrix.cpp	
An extension of <a href="#">Markov::API::MarkovPasswords</a> . . . . .	284
modelMatrix.h	
An extension of <a href="#">Markov::API::MarkovPasswords</a> . . . . .	288
node.h	
Node class template . . . . .	292
MarkovModel/src/pch.cpp	
For windows dynamic library . . . . .	297
UnitTests/pch.cpp	
For windows dynamic library . . . . .	297
MarkovModel/src/pch.h	
For windows dynamic library . . . . .	298
UnitTests/pch.h	
For windows dynamic library . . . . .	299
random-model.py . . . . .	300
random.h	
Random engine implementations for <a href="#">Markov</a> . . . . .	301
term.cpp	
Terminal handler for pretty stuff like colors . . . . .	305
term.h	
Terminal handler for pretty stuff like colors . . . . .	307
threadSharedListHandler.cpp	
Thread-safe wrapper for std::ifstream . . . . .	310
threadSharedListHandler.h	
Thread-safe wrapper for std::ifstream . . . . .	311
Train.cpp	
Training page for GUI . . . . .	314
Train.h	
Training page for GUI . . . . .	315
UnitTests.cpp	
Unit tests with Microsoft::VisualStudio::CppUnitTestFramework . . . . .	317

# CHAPTER8

---

## Namespace Documentation

---

### 8.1 markopy Namespace Reference

#### Functions

- def `cli_init` (str input\_model)
- def `cli_train` (markopy.ModelMatrix model, str dataset, str seperator, str output, bool output\_forced=False, bool bulk=False)
- def `cli_generate` (markopy.ModelMatrix model, str wordlist, bool bulk=False)

#### Variables

- `parser`
- `help`
- `default`
- `action`
- `args = parser.parse_args()`

#### 8.1.1 Detailed Description

```
@package markopy
@file markopy.py
@namespace Python::Markopy
@brief Command line
@authors Ata Hakçıl
```

#### 8.1.2 Function Documentation

##### 8.1.2.1 `cli_generate()`

```
def markopy.cli_generate (
    markopy.ModelMatrix model,
    str wordlist,
    bool bulk = False )

Generate strings from the model
@param model: model instance
@param wordlist: wordlist filename
@param bulk: marks bulk operation with directories
```

Definition at line 106 of file `markopy.py`.

```
00106 def cli_generate(model : markopy.ModelMatrix, wordlist : str, bulk : bool=False):
00107     """ Generate strings from the model
00108     @param model: model instance
00109     @param wordlist: wordlist filename
00110     @param bulk: marks bulk operation with directories
00111     """
00112     if not (wordlist or args.count):
00113         logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters. Exiting.")
00114         exit(2)
00115
00116     if(bulk and os.path.isfile(wordlist)):
00117         logging pprint(f"{wordlist} exists and will be overwritten.", 1)
00118     model.Generate(int(args.count), wordlist, int(args.min), int(args.max), int(args.threads))
00119
00120 if(args.bulk):
00121     logging pprint(f"\"Bulk mode operation chosen.\", 4)
```

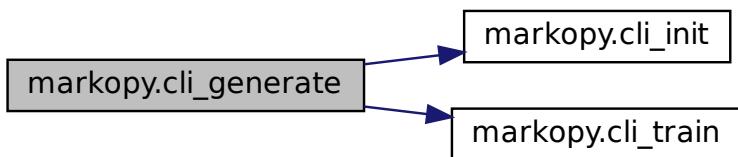
```

00123
00124     if (args.mode.lower() == "train"):
00125         if (os.path.isdir(args.output) and not os.path.isfile(args.output)) and
00126             (os.path.isdir(args.dataset) and not os.path.isfile(args.dataset)):
00127             corpus_list = os.listdir(args.dataset)
00128             for corpus in corpus_list:
00129                 model = cli_init(args.input)
00130                 logging pprint(f"Training {args.input} with {corpus}", 2)
00131                 output_file_name = corpus
00132                 model_extension = ""
00133                 if "." in args.input:
00134                     model_extension = args.input.split(".")[-1]
00135                     cli_train(model, f"{args.dataset}/{corpus}", args.separator,
00136                               f"{args.output}/{corpus}.{model_extension}", output_forced=True, bulk=True)
00137             else:
00138                 logging pprint("In bulk training, output and dataset should be a directory.")
00139                 exit(1)
00140
00141         elif (args.mode.lower() == "generate"):
00142             if (os.path.isdir(args.wordlist) and not os.path.isfile(args.wordlist)) and
00143                 (os.path.isdir(args.input) and not os.path.isfile(args.input)):
00144                 model_list = os.listdir(args.input)
00145                 print(model_list)
00146                 for input in model_list:
00147                     logging pprint(f"Generating from {args.input}/{input} to {args.wordlist}/{input}.txt",
00148                                  2)
00149
00150                     model = cli_init(f"{args.input}/{input}")
00151                     model_base = input
00152                     if "." in args.input:
00153                         model_base = input.split(".")[1]
00154                     cli_generate(model, f"{args.wordlist}/{model_base}.txt", bulk=True)
00155             else:
00156                 logging pprint("In bulk generation, input and wordlist should be directory.")
00157
00158
00159
00160         elif (args.mode.lower() == "train"):
00161             cli_train(model, args.dataset, args.separator, args.output, output_forced=True)
00162
00163
00164         elif(args.mode.lower() == "combine"):
00165             cli_train(model, args.dataset, args.separator, args.output)
00166             cli_generate(model, args.wordlist)
00167
00168
00169     else:
00170         logging pprint("Invalid mode arguement given.")
00171         logging pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00172         exit(5)

```

References [cli\\_init\(\)](#), and [cli\\_train\(\)](#).

Here is the call graph for this function:



### 8.1.2.2 cli\_init()

```

def markopy.cli_init (
    str input_model )

```

```
Initialize the model for CLI interation
@param input_model Model filename
```

Definition at line 45 of file [markopy.py](#).

```
00045 def cli_init(input_model : str):
00046     """ Initialize the model for CLI interation
00047     @param input_model Model filename
00048     """
00049     logging.VERBOSITY = 0
00050     if args.verbosity:
00051         logging.VERBOSITY = args.verbosity
00052         logging pprint(f"Verbosity set to {args.verbosity}.", 2)
00053
00054     logging pprint("Initializing model.", 1)
00055     model = markopy.ModelMatrix()
00056     logging pprint("Model initialized.", 2)
00057
00058     logging pprint("Importing model file.", 1)
00059
00060     if(not os.path.isfile(input_model)):
00061         logging pprint(f"Model file at {input_model} not found. Check the file path, or working
directory")
00062         exit(1)
00063
00064     model.Import(input_model)
00065     logging pprint("Model imported successfully.", 2)
00066     return model
00067
```

Referenced by [cli\\_generate\(\)](#).

Here is the caller graph for this function:



### 8.1.2.3 cli\_train()

```
def markopy.cli_train (
    markopy.ModelMatrix model,
    str dataset,
    str seperator,
    str output,
    bool output_forced = False,
    bool bulk = False )

Train a model via CLI parameters
@param model Model instance
@param dataset filename for the dataset
@param seperator seperator used with the dataset
@param output output filename
@param output_forced force overwrite
@param bulk marks bulk operation with directories
```

Definition at line 68 of file [markopy.py](#).

```
00068 def cli_train(model : markopy.ModelMatrix, dataset : str, seperator : str, output : str, output_forced
: bool=False, bulk : bool=False):
00069     """ Train a model via CLI parameters
00070     @param model Model instance
00071     @param dataset filename for the dataset
00072     @param seperator seperator used with the dataset
00073     @param output output filename
00074     @param output_forced force overwrite
00075     @param bulk marks bulk operation with directories
00076     """
```

```

00077     if not (dataset and separator and (output or not output_forced)):
00078         logging pprint(f"Training mode requires -d/--dataset{', -o/--output' if output_forced else''}
00079         and -s/--separator parameters. Exiting.")
00080         exit(2)
00080
00081     if(not bulk and not os.path.isfile(dataset)):
00082         logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00083         exit(3)
00084
00085     if(output and os.path.isfile(output)):
00086         logging pprint(f"{output} exists and will be overwritten.", 1 )
00087
00088     if(seperator == '\\t'):
00089         logging pprint("Escaping seperator.", 3)
00090         seperator = '\t'
00091
00092     if(len(seperator)!=1):
00093         logging pprint(f'Delimiter must be a single character, and "{seperator}" is not accepted.')
00094         exit(4)
00095
00096     logging pprint(f'Starting training.', 3)
00097     model.Train(dataset,seperator, int(args.threads))
00098     logging pprint(f'Training completed.', 2)
00099
00100    if(output):
00101        logging pprint(f'Exporting model to {output}', 2)
00102        model.Export(output)
00103    else:
00104        logging pprint(f'Model will not be exported.', 1)
00105

```

Referenced by [cli\\_generate\(\)](#).

Here is the caller graph for this function:



### 8.1.3 Variable Documentation

#### 8.1.3.1 action

`markopy.action`

Definition at line 38 of file [markopy.py](#).

#### 8.1.3.2 args

`markopy.args = parser.parse_args()`

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

#### 8.1.3.3 default

`markopy.default`

Definition at line 34 of file [markopy.py](#).

#### 8.1.3.4 help

`markopy.help`

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

### 8.1.3.5 parser

```
markopy.parser
Initial value:
00001 = argparse.ArgumentParser(description="Python wrapper for MarkovPasswords.",
00002 epilog=f, formatter_class=argparse.RawTextHelpFormatter)
Definition at line 14 of file markopy.py.
```

## 8.2 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](#)
- [Random](#)  
*Objects related to RNG.*

### Classes

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

### 8.2.1 Detailed Description

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

## 8.3 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.*

### 8.3.1 Detailed Description

Namespace for the [MarkovPasswords API](#).

## 8.4 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)

### 8.4.1 Detailed Description

Structure to hold parsed cli arguements.

Namespace for the [CLI](#) objects

## 8.4.2 Typedef Documentation

### 8.4.2.1 ProgramOptions

```
typedef struct Markov::API::CLI::\_programOptions Markov::API::CLI::ProgramOptions
Structure to hold parsed cli arguements.
```

## 8.4.3 Function Documentation

### 8.4.3.1 operator<<()

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

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[1;" << buf << "m";
00070     return os;
00071 }
```

References [Markov::API::CLI::Terminal::colormap](#).

## 8.5 Markov::API::Concurrency Namespace Reference

Namespace for [Concurrency](#) related classes.

### Classes

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

#### 8.5.1 Detailed Description

Namespace for [Concurrency](#) related classes.

## 8.6 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*

#### 8.6.1 Detailed Description

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

#### 8.6.2 Function Documentation

##### 8.6.2.1 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

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

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

```

00195
00196
00197     int kernelWorkerIndex = threadIdx.x + blockIdx.x * blockDim.x;
00198
00199     if(n==0) return;
00200
00201     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;

```

```

00245         }
00246         res[bufferctr + len++] = '\n';
00247         bufferctr+=len;
00248     }
00249     res[bufferctr] = '\0';
00250 }
```

### 8.6.2.2 strchr()

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

srtchr implementation on **device** space  
Find 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
00253     for ( ; ; ++p, s_len--) {
00254         if (*p == c)
00255             return ((char *)p);
00256         if (!*p)
00257             return ((char *)NULL);
00258     }
00259 }
```

## 8.7 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 o 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.*

### 8.7.1 Detailed Description

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

### 8.7.2 Function Documentation

### 8.7.2.1 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

x	marsaglia internal x. Not constant, (ref)
y	marsaglia internal y. Not constant, (ref)
z	marsaglia internal z. Not constant, (ref)

#### 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 }
```

## 8.8 Markov::GUI Namespace Reference

namespace for MarkovPasswords [API GUI](#) wrapper

### Classes

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

### 8.8.1 Detailed Description

namespace for MarkovPasswords [API GUI](#) wrapper

## 8.9 Markov::Markopy Namespace Reference

### Functions

- [BOOST\\_PYTHON\\_MODULE](#) (markopy)

## 8.9.1 Function Documentation

### 8.9.1.1 BOOST\_PYTHON\_MODULE()

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

Definition at line 34 of file [markopy.cpp](#).

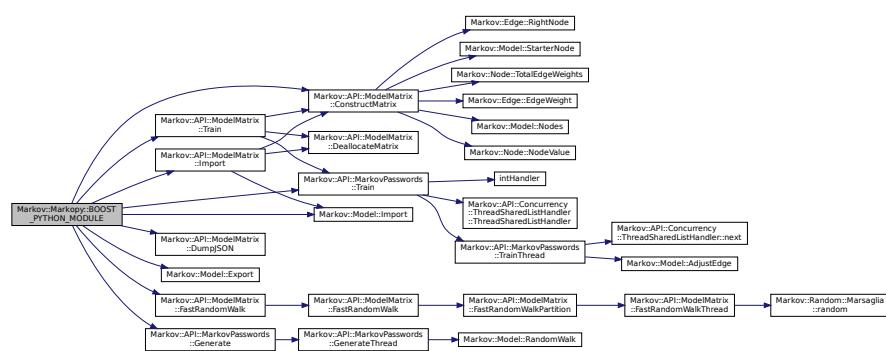
```
00035 {
00036     bool (Markov::API::MarkovPasswords::*Import) (const char*) = &Markov::Model<char>::Import;
00037     bool (Markov::API::MarkovPasswords::*Export) (const char*) = &Markov::Model<char>::Export;
00038     class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
00039         .def(init<>())
00040         .def("Train", &Markov::API::MarkovPasswords::Train)
00041         .def("Generate", &Markov::API::MarkovPasswords::Generate)
00042         .def("Import", Import, "Import a model file.")
00043         .def("Export", Export, "Export a model to file.")
00044     ;
00045
00046     int (Markov::API::ModelMatrix::*FastRandomWalk) (unsigned long int, const char*, int, int, int,
00047         bool) = &Markov::API::ModelMatrix::FastRandomWalk;
00048     class_<Markov::API::ModelMatrix>("ModelMatrix", init<>())
00049         .def(init<>())
00050         .def("Train", &Markov::API::ModelMatrix::Train)
00051         .def("Import", &Markov::API::ModelMatrix::Import, "Import a model file.")
00052         .def("Export", Export, "Export a model to file.")
00053         .def("ConstructMatrix", &Markov::API::ModelMatrix::ConstructMatrix)
00054         .def("DumpJSON", &Markov::API::ModelMatrix::DumpJSON)
00055         .def("FastRandomWalk", FastRandomWalk)
00056         ;
00057     };
00058 }
```

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:



## 8.10 Markov::Random Namespace Reference

Objects related to RNG.

### Classes

- class [DefaultRandomEngine](#)  
*Implementation using [Random.h](#) default random engine.*
- class [Marsaglia](#)  
*Implementation of [Marsaglia Random Engine](#).*
- class [Mersenne](#)  
*Implementation of [Mersenne Twister Engine](#).*

- class [RandomEngine](#)  
*An abstract class for Random Engine.*

### 8.10.1 Detailed Description

Objects related to RNG.

## 8.11 model\_2gram Namespace Reference

### Variables

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

### 8.11.1 Detailed Description

python script for generating a 2gram model

### 8.11.2 Variable Documentation

#### 8.11.2.1 alphabet

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

#### 8.11.2.2 f

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

## 8.12 random Namespace Reference

### 8.12.1 Detailed Description

-model

python script for generating a 2gram model

## 8.13 random-model Namespace Reference

### Variables

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

### 8.13.1 Variable Documentation

#### 8.13.1.1 alphabet

random-model.alphabet = string.printable

password alphabet

Definition at line 10 of file [random-model.py](#).

#### 8.13.1.2 f

random-model.f = open('.../.../models/random.mdl', "wb")

output file handle

Definition at line 16 of file [random-model.py](#).

## 8.14 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.

### 8.14.1 Detailed Description

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

## 8.15 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.*

### 8.15.1 Detailed Description

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

### 8.15.2 Function Documentation

### 8.15.2.1 TEST\_CLASS() [1/3]

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

Test class for rest of Edge cases.

send exception on integer underflow

test integer overflows

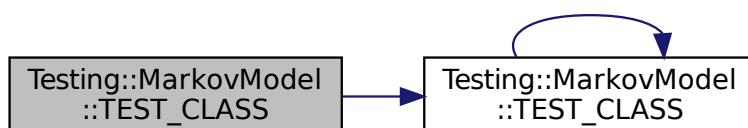
Definition at line 500 of file [UnitTests.cpp](#).

```
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:



### 8.15.2.2 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(functional_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 
```

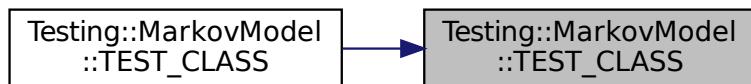
References [TEST\\_CLASS\(\)](#).

Referenced by [TEST\\_CLASS\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.15.2.3 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](#).

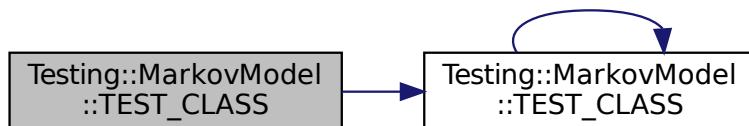
```
00539 {
```

```

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
00557     TEST_METHOD(rand_next_u64_max) {
00558         Markov::Random::Marsaglia MarsagliaRandomEngine;
00559         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00560         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00561         Markov::Edge<unsigned char>* e = src->Link(target1);
00562         e->AdjustEdge((0xffffffff));
00563         Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00564         Assert::IsTrue(res == target1);
00565         delete src;
00566         delete target1;
00567         delete e;
00568
00569     }
00570
00571     TEST_METHOD(uninitialized_rand_next) {
00572
00573         auto _invalid_next = [] {
00574             Markov::Random::Marsaglia MarsagliaRandomEngine;
00575             Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00576             Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00577             Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(src, target1);
00578             Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00579
00580             delete src;
00581             delete target1;
00582             delete e;
00583         };
00584
00585         Assert::ExpectException<std::logic_error>(_invalid_next);
00586     }
00587
00588 }
00589
00590 }
```

References [TEST\\_CLASS\(\)](#).

Here is the call graph for this function:



## 8.16 Testing::MarkovPasswords Namespace Reference

Testing namespace for [MarkovPasswords](#).

### 8.16.1 Detailed Description

Testing namespace for [MarkovPasswords](#).

## 8.17 Testing::MVP Namespace Reference

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

### Namespaces

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

#### 8.17.1 Detailed Description

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

## 8.18 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.*

#### 8.18.1 Detailed Description

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

#### 8.18.2 Function Documentation

##### 8.18.2.1 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');
```

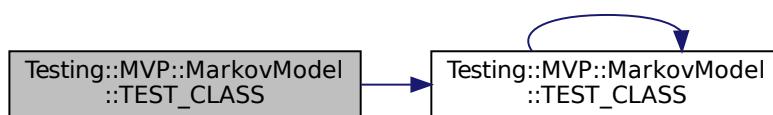
```

00045     RightNode);
00046     Assert::IsTrue(LeftNode == e->LeftNode());
00047     Assert::IsTrue(RightNode == e->RightNode());
00048     delete LeftNode;
00049     delete RightNode;
00050     delete e;
00051 }
00052
00053 TEST_METHOD(AdjustEdge) {
00054     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00055     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00056     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00057     RightNode);
00058     e->AdjustEdge(15);
00059     Assert::AreEqual(15ull, e->EdgeWeight());
00060     e->AdjustEdge(15);
00061     Assert::AreEqual(30ull, e->EdgeWeight());
00062     delete LeftNode;
00063     delete RightNode;
00064     delete e;
00065 }
00066
00067 TEST_METHOD(TraverseNode) {
00068     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00069     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00070     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00071     RightNode);
00072     Assert::IsTrue(RightNode == e->TraverseNode());
00073     delete LeftNode;
00074     delete RightNode;
00075     delete e;
00076 }
00077
00078 TEST_METHOD(set_left_and_right) {
00079     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00080     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00081     Markov::Edge<unsigned char>* el = new Markov::Edge<unsigned char>(LeftNode,
00082     RightNode);
00083
00084     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>;
00085     e2->SetLeftEdge(LeftNode);
00086     e2->SetRightEdge(RightNode);
00087
00088     Assert::IsTrue(el->LeftNode() == e2->LeftNode());
00089     Assert::IsTrue(el->RightNode() == e2->RightNode());
00090     delete LeftNode;
00091     delete RightNode;
00092     delete el;
00093     delete e2;
00094 }
00095
00096 TEST_METHOD(negative_adjust) {
00097     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00098     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00099     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00100     RightNode);
00101     e->AdjustEdge(15);
00102     Assert::AreEqual(15ull, e->EdgeWeight());
00103     e->AdjustEdge(-15);
00104     Assert::AreEqual(0ull, e->EdgeWeight());
00105     delete LeftNode;
00106     delete RightNode;
00107     delete e;
00108 }
00109
00110
00111
00112 }
00113

```

References [TEST\\_CLASS\(\)](#).

Here is the call graph for this function:



### 8.18.2.2 TEST\_CLASS() [2/3]

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

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         public:
00356             TEST_METHOD(model_constructor) {
00357                 Markov::Model<unsigned char> m;
00358                 Assert::AreEqual((unsigned char)'\\0', m.StarterNode()->nodeValue());
00359             }
00360
00361             TEST_METHOD(import_filename) {
00362                 Markov::Model<unsigned char> m;
00363                 Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
00364             }
00365
00366             TEST_METHOD(export_filename) {
00367                 Markov::Model<unsigned char> m;
00368                 Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
00369             }
00370
00371             TEST_METHOD(random_walk) {
00372                 unsigned char* res = new unsigned char[12 + 5];
00373                 Markov::Random::Marsaglia MarsagliaRandomEngine;
00374                 Markov::Model<unsigned char> m;
00375                 Assert::IsTrue(m.Import("../Models/finished2.mdl"));
00376                 Assert::IsNotNull(m.RandomWalk(&MarsagliaRandomEngine, 1, 12, res));
00377             }
00378     };
```

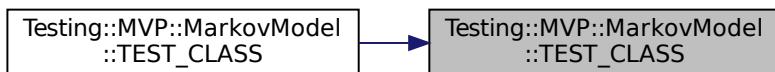
References [TEST\\_CLASS\(\)](#).

Referenced by [TEST\\_CLASS\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.18.2.3 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
00123             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
00131             TEST_METHOD(uchar_constructor) {
00132                 Markov::Node<unsigned char>* n = NULL;
00133                 unsigned char test_cases[] = { 'c', 0x00, 0xff, -32 };
00134                 for (unsigned char tcase : test_cases) {
00135                     n = new Markov::Node<unsigned char>(tcase);
00136                     Assert::AreEqual(tcase, n->NodeValue());
00137                     delete n;
00138                 }
00139             }
00140
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
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
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             TEST_METHOD(rand_next_u32) {
00184                 Markov::Random::Marsaglia MarsagliaRandomEngine;
00185                 Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00186                 Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00187                 Markov::Edge<unsigned char>* e = src->Link(target1);
00188                 e->AdjustEdge(1 << 31);
00189                 Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00190                 Assert::IsTrue(res == target1);
00191
00192
00193

```

```

00194         delete src;
00195         delete target1;
00196         delete e;
00197     }
00198
00199
00200     TEST_METHOD(rand_next_choice_1) {
00201         Markov::Random::Marsaglia MarsagliaRandomEngine;
00202         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00203         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00204         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00205         Markov::Edge<unsigned char>* el = src->Link(target1);
00206         Markov::Edge<unsigned char>* e2 = src->Link(target2);
00207         el->AdjustEdge(1);
00208         e2->AdjustEdge((unsigned long)(ull << 31));
00209         Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00210         Assert::IsNotNull(res);
00211         Assert::IsTrue(res == target2);
00212         delete src;
00213         delete target1;
00214         delete el;
00215         delete e2;
00216     }
00217
00218
00219
00220     TEST_METHOD(rand_next_choice_2) {
00221         Markov::Random::Marsaglia MarsagliaRandomEngine;
00222         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00223         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00224         Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00225         Markov::Edge<unsigned char>* el = src->Link(target1);
00226         Markov::Edge<unsigned char>* e2 = src->Link(target2);
00227         el->AdjustEdge(1);
00228         e2->AdjustEdge((unsigned long)(ull << 31));
00229         Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00230         Assert::IsNotNull(res);
00231         Assert::IsTrue(res == target1);
00232         delete src;
00233         delete target1;
00234         delete e1;
00235         delete e2;
00236     }
00237
00238
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>* el = new Markov::Edge<unsigned char>(src, target1);
00247         Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00248         el->AdjustEdge(25);
00249         src->UpdateEdges(el);
00250         e2->AdjustEdge(30);
00251         src->UpdateEdges(e2);
00252
00253         Assert::AreEqual((size_t)2, src->Edges()->size());
00254
00255
00256         delete src;
00257         delete target1;
00258         delete e1;
00259         delete e2;
00260     }
00261
00262
00263
00264     TEST_METHOD(update_edges_total) {
00265
00266         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00267         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00268         Markov::Edge<unsigned char>* el = new Markov::Edge<unsigned char>(src, target1);
00269         Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target1);
00270         el->AdjustEdge(25);
00271         src->UpdateEdges(el);
00272         e2->AdjustEdge(30);
00273         src->UpdateEdges(e2);
00274
00275         //Assert::AreEqual(55ull, src->TotalEdgeWeights());
00276
00277
00278         delete src;
00279         delete target1;
00280         delete e1;
00281         delete e2;
00282     }
00283
00284
00285
00286     TEST_METHOD(find_vertice) {
00287
00288         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00289

```

```

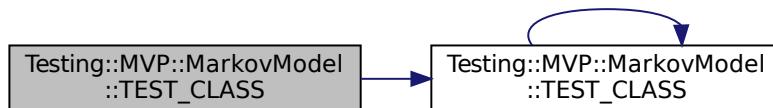
00291     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00292     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00293     Markov::Edge<unsigned char>* res = NULL;
00294     src->Link(target1);
00295     src->Link(target2);

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 TEST_METHOD(find_vertice_without_any) {
00315
00316     auto _invalid_next = [] {
00317         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00318         Markov::Edge<unsigned char>* res = NULL;
00319
00320         res = src->FindEdge('b');
00321         Assert::IsNull(res);
00322
00323         delete src;
00324     };
00325
00326     //Assert::ExpectException<std::logic_error>(_invalid_next);
00327 }
00328
00329
00330 TEST_METHOD(find_vertice_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 }
00347
00348
00349 };

```

References [TEST\\_CLASS\(\)](#).

Here is the call graph for this function:



## 8.19 Testing::MVP::MarkovPasswords Namespace Reference

Testing namespace for [MVP MarkovPasswords](#).

### Functions

- [TEST\\_CLASS](#) (ArgParser)

*Test Class for Argparse class.*

### 8.19.1 Detailed Description

Testing namespace for [MVP MarkovPasswords](#).

### 8.19.2 Function Documentation

#### 8.19.2.1 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](#).

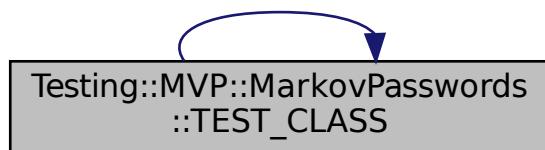
```
00396         {
00397             public:
00400                 TEST_METHOD(generate_basic) {
00401                     int argc = 8;
00402                     char *argv[] = {"markov.exe", "generate", "-if", "model.mdl", "-of",
00403 "passwords.txt", "-n", "100"};
00404                     /*ProgramOptions *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
00457     TEST_METHOD(train_basic) {
00458         int argc = 4;
00459         char *argv[] = { "markov.exe", "train", "-ef", "model.mdl" };
00460
00461         /*ProgramOptions* p = Argparse:::parse(argc, argv);
00462         Assert:::IsNotNull(p);
00463
00464         Assert:::AreEqual(p->bImport, false);
00465         Assert:::AreEqual(p->bExport, true);
00466         Assert:::AreEqual(p->exportname, "model.mdl"); */
00467
00468     }
00469
00470     TEST_METHOD(train_basic_longname) {
00471         int argc = 4;
00472         char *argv[] = { "markov.exe", "train", "--exportfilename", "model.mdl" };
00473
00474         /*ProgramOptions* p = Argparse:::parse(argc, argv);
00475         Assert:::IsNotNull(p);
00476
00477         Assert:::AreEqual(p->bImport, false);
00478         Assert:::AreEqual(p->bExport, true);
00479         Assert:::AreEqual(p->exportname, "model.mdl"); */
00480
00481     }
00482
00483     }
00484
00485
00486
00487
00488 };
```

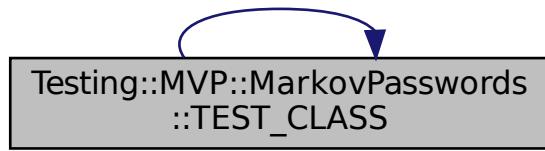
References [TEST\\_CLASS\(\)](#).

Referenced by [TEST\\_CLASS\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



# CHAPTER9

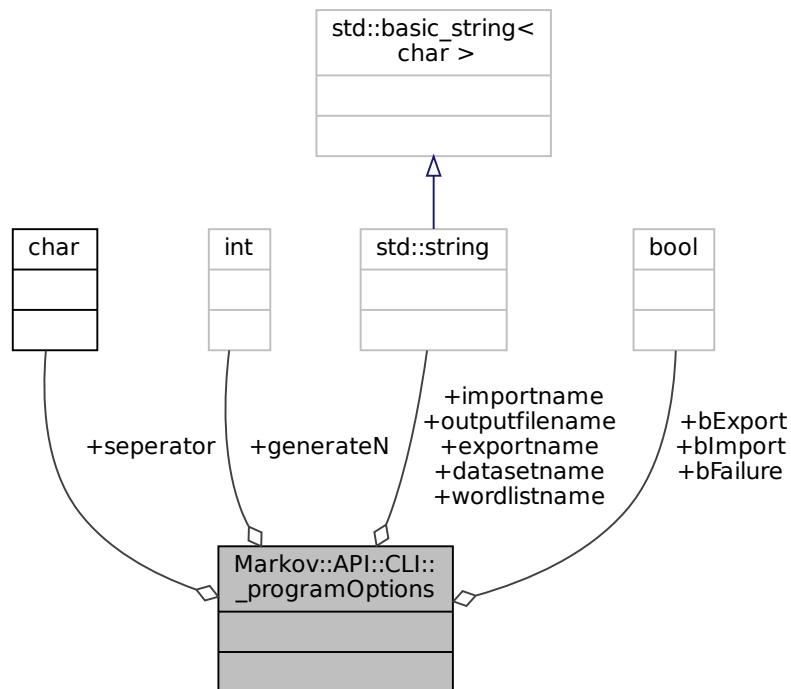
## Class Documentation

### 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 separator`  
*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:

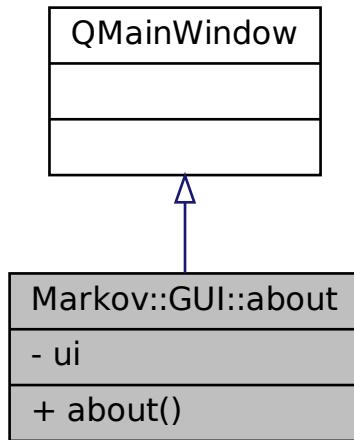
- [argparse.h](#)

## 9.2 Markov::GUI::about Class Reference

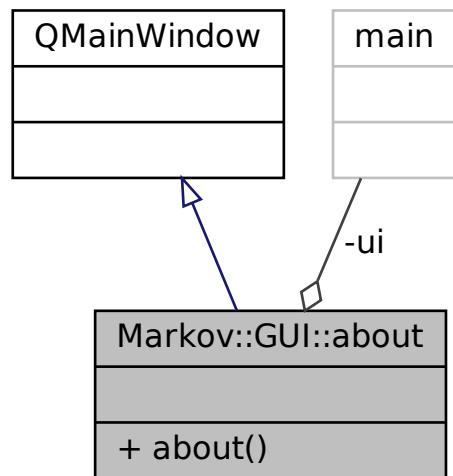
QT Class for about page.

```
#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:

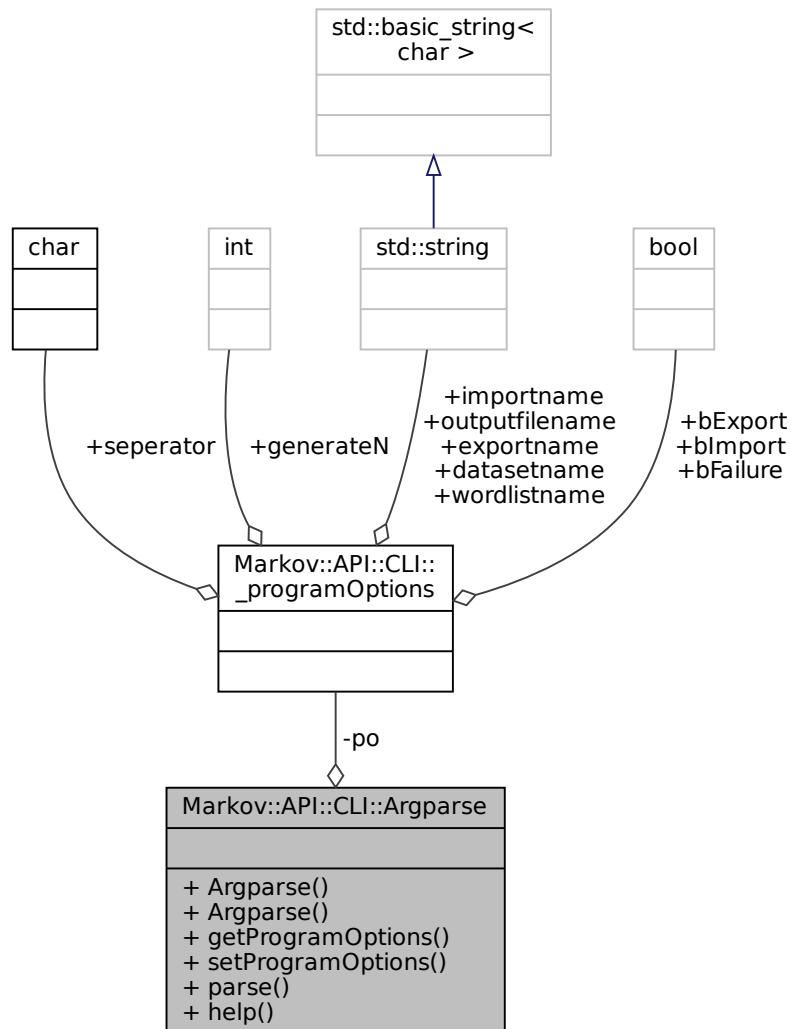
- [about.h](#)
- [about.cpp](#)

## 9.3 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 ()`

*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.*

### 9.3.1 Detailed Description

Parse command line arguments.

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

### 9.3.2 Constructor & Destructor Documentation

#### 9.3.2.1 Argparse() [1/2]

```
Markov::API::CLI::Argparse::Argparse ( )
```

#### 9.3.2.2 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

<code>argc</code>	Number of command line arguments
<code>argv</code>	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     desc.add_options()
00109         ("generate", "Generate strings with given parameters")
00110         ("train", "Train model with given parameters")
00111         ("combine", "Combine")
00112         ("import", opt::value<std::string>(), "Import model file")
00113         ("output", opt::value<std::string>(), "Output model file. This model will be exported
when done. Will be ignored for generation mode")
00114         ("dataset", opt::value<std::string>(), "Dataset file to read input from training. Will
be ignored for generation mode")
00115         ("seperator", opt::value<char>(), "Seperator character to use with training data.
(character between occurrence and value)")
00116         ("wordlist", opt::value<std::string>(), "Wordlist file path to export generation
results to. Will be ignored for training mode")
00117         ("count", opt::value<int>(), "Number of lines to generate. Ignored in training mode")
00118         ("verbosity", "Output verbosity")
00119         ("help", "Option definitions");
00120
00121     opt::variables_map vm;
00122
```

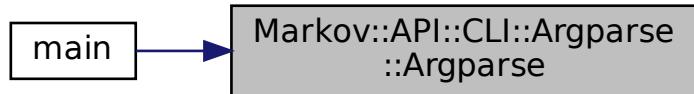
```

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
00195     //else if (vm.count("train")) std::cout << "train oldu" << std::endl;
00196 }
```

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



### 9.3.3 Member Function Documentation

#### 9.3.3.1 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](#).

#### 9.3.3.2 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 }
```

### 9.3.3.3 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; }

### 9.3.3.4 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.separator = 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 }

```

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::separator](#).

### 9.3.4 Member Data Documentation

#### 9.3.4.1 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:

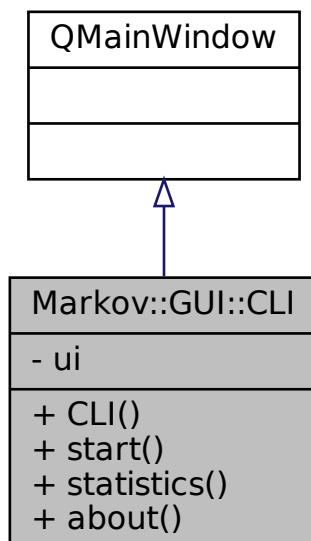
- [argparse.h](#)
- [argparse.cpp](#)

## 9.4 Markov::GUI::CLI Class Reference

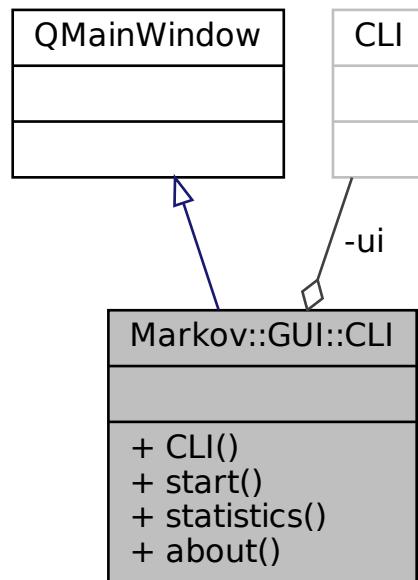
QT [CLI](#) Class.

```
#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`

### 9.4.1 Detailed Description

QT `CLI` Class.

Definition at line 14 of file `CLI.h`.

### 9.4.2 Constructor & Destructor Documentation

#### 9.4.2.1 `CLI()`

```

Markov::GUI::CLI::CLI (
    QWidget * parent = Q_NULLPTR )
Definition at line 15 of file CLI.cpp.
00016     : QMainWidget(parent)
00017 {
  
```

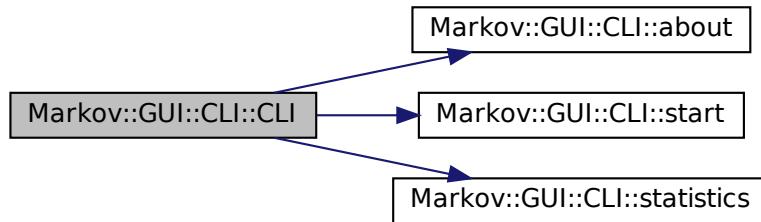
```

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:



### 9.4.3 Member Function Documentation

#### 9.4.3.1 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:



#### 9.4.3.2 start

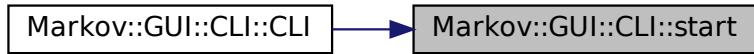
```

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:

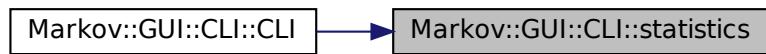


#### 9.4.3.3 statistics

```
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:



### 9.4.4 Member Data Documentation

#### 9.4.4.1 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:

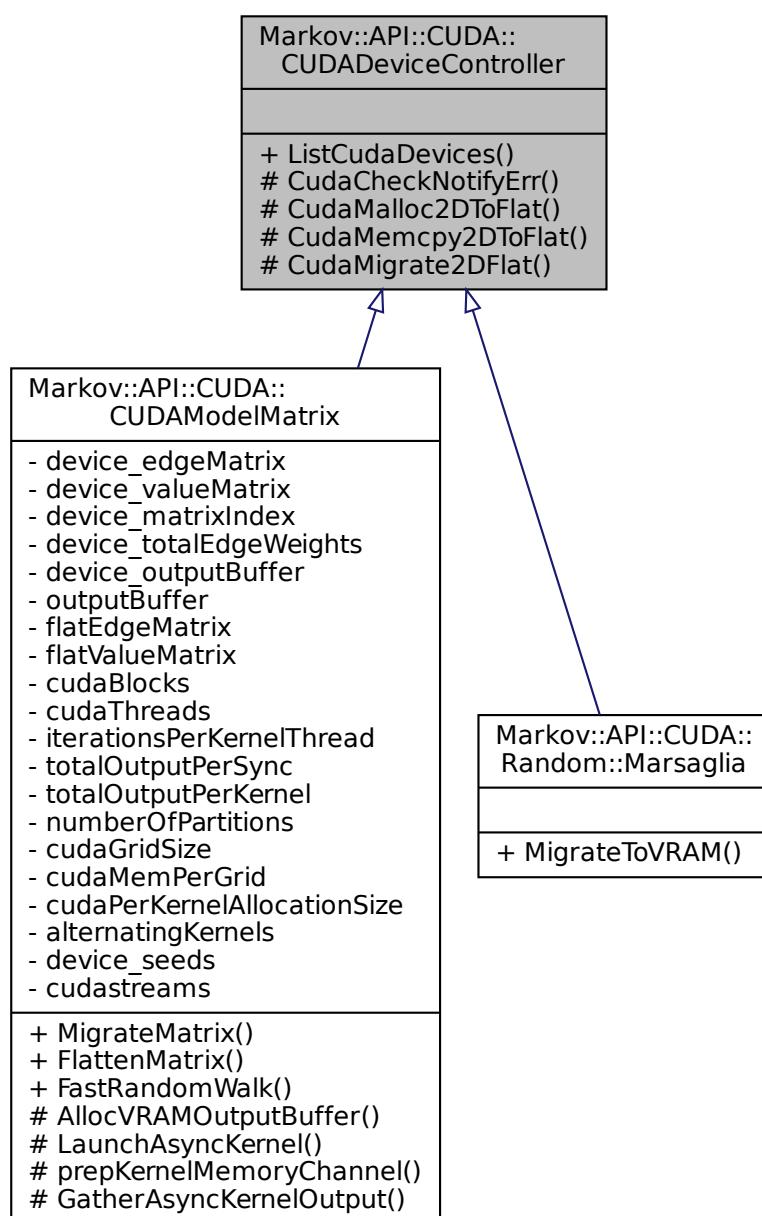
- [CLI.h](#)
- [CLI.cpp](#)

## 9.5 Markov::API::CUDA::CUDADeviceController Class Reference

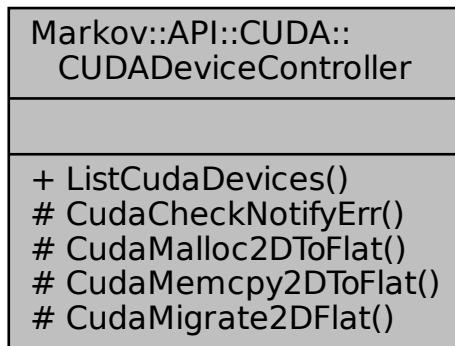
Controller class for [CUDA](#) device.

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

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



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



## 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.*

### 9.5.1 Detailed Description

Controller class for [CUDA](#) device.

This implementation only supports Nvidia devices.

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

### 9.5.2 Member Function Documentation

#### 9.5.2.1 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 }
```

### 9.5.2.2 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:**

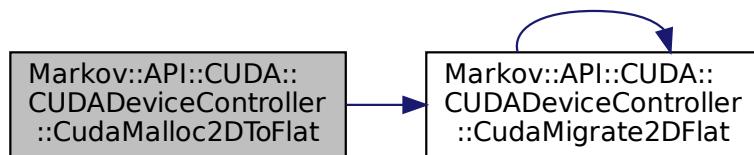
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
}
```

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

```
00075
00076     cudaError_t cudastatus = cudaMalloc((T **)dst, row*col*sizeof(T));
00077     CudaCheckNotifyErr(cudastatus, "cudaMalloc Failed.", false);
00078     return cudastatus;
00079 }
```

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

Here is the call graph for this function:



### 9.5.2.3 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);
  
```

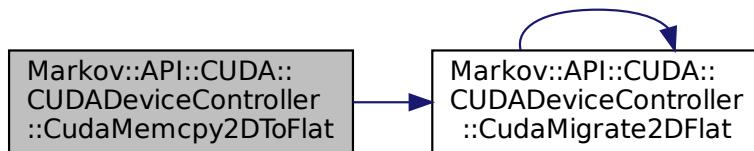
Definition at line 103 of file [cudaDeviceController.h](#).

```

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:



#### 9.5.2.4 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

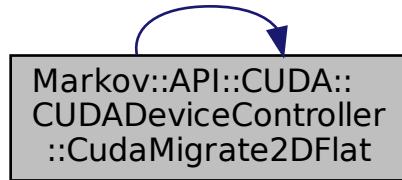
##### Example output:

```
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 }
```

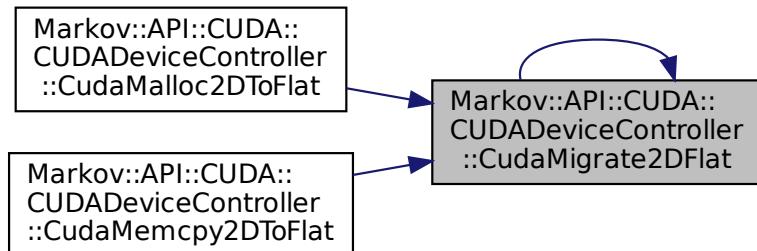
References [CudaMigrate2DFlat\(\)](#).

Referenced by [CudaMalloc2DToFlat\(\)](#), [CudaMemcpy2DToFlat\(\)](#), and [CudaMigrate2DFlat\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.5.2.5 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
00017     host.                                     //list cuda Capable devices on
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     }
  
```

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

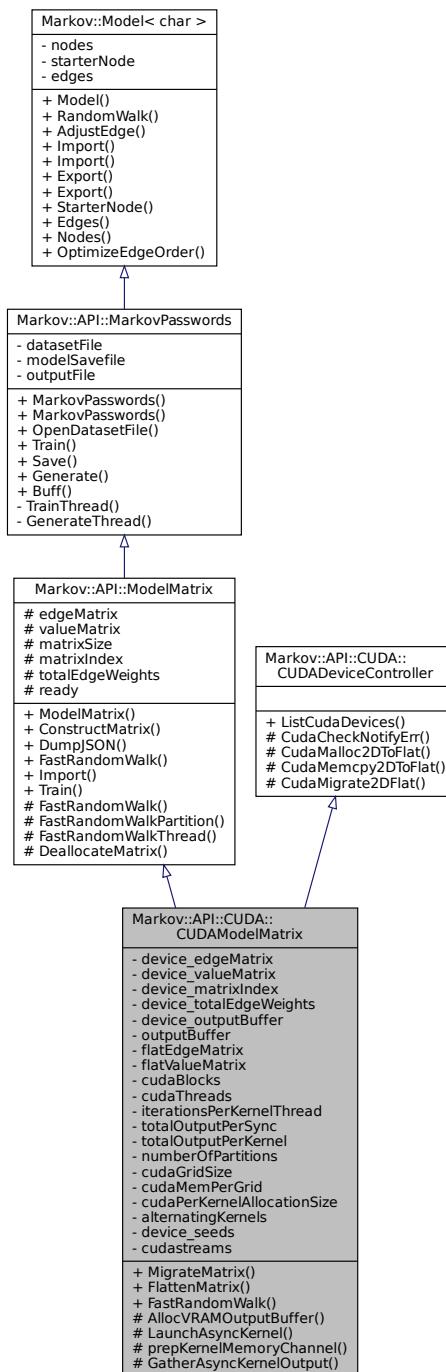
- [cudaDeviceController.h](#)
- [cudaDeviceController.cu](#)

## 9.6 Markov::API::CUDA::CUDAModelMatrix Class Reference

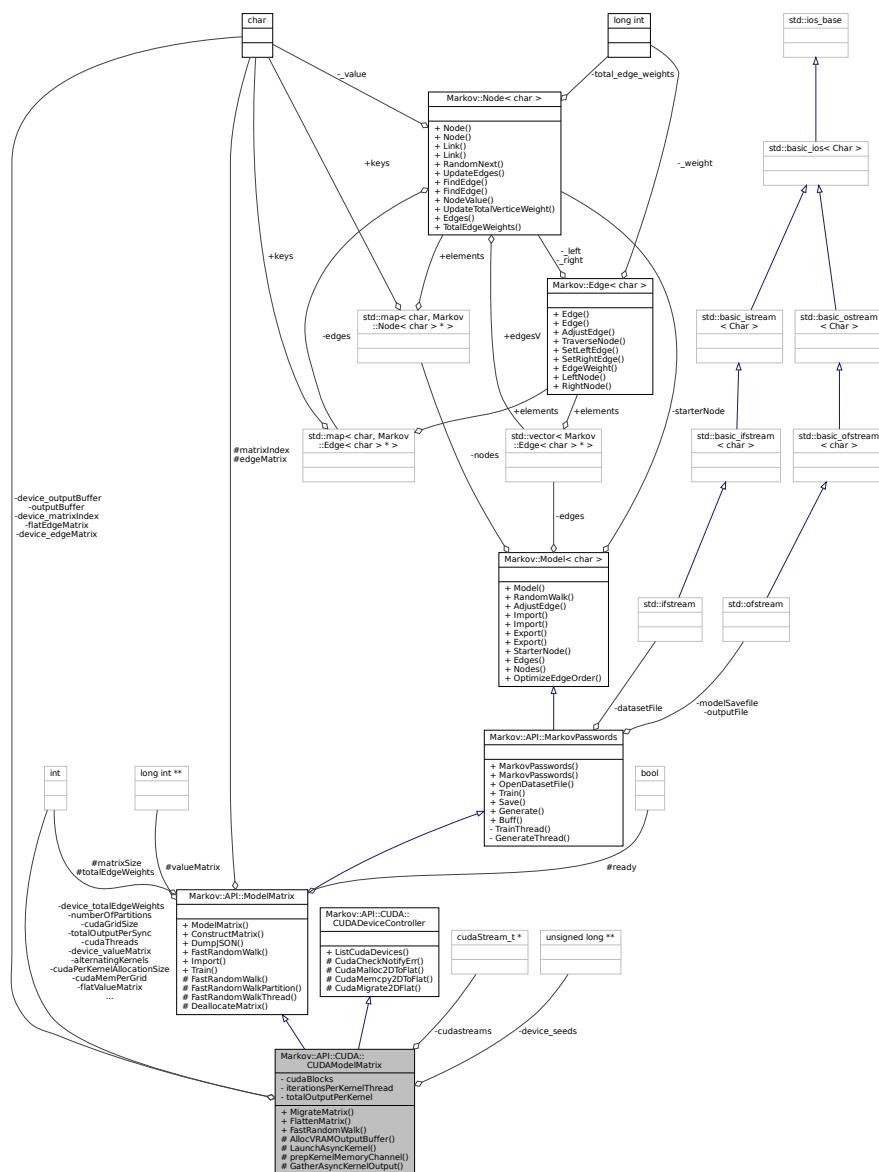
Extension of [Markov::API::ModelMatrix](#) which is modified to run on GPU devices.

```
#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 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.*

## 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 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)
- 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.*
  - 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.

## 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.*

### 9.6.1 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`.

### 9.6.2 Member Function Documentation

### 9.6.2.1 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 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 }
```

### 9.6.2.2 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 <a href="#">CUDA</a> kernel
sizePerGrid	- Size to allocate per grid member

#### Returns

pointer to the allocation on VRAM

### 9.6.2.3 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 buffered
<i>multiplier</i>	A constant value to buffer 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

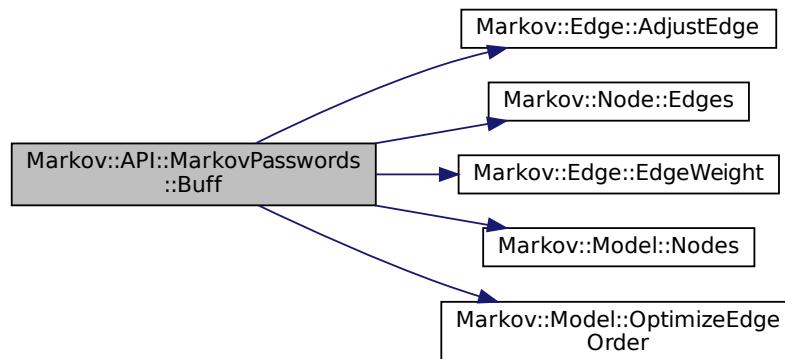
Definition at line 153 of file [markovPasswords.cpp](#).

```
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 }
```

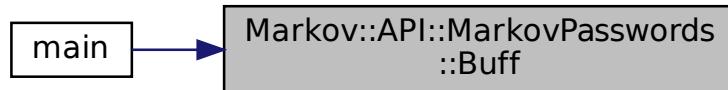
References [Markov::Edge< NodeStorageType >::AdjustEdge\(\)](#), [Markov::Node< storageType >::Edges\(\)](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), and [Markov::Model< NodeStorageType >::OptimizeEdgeOrder\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.6.2.4 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     }
00074     i++;
00075 }

```

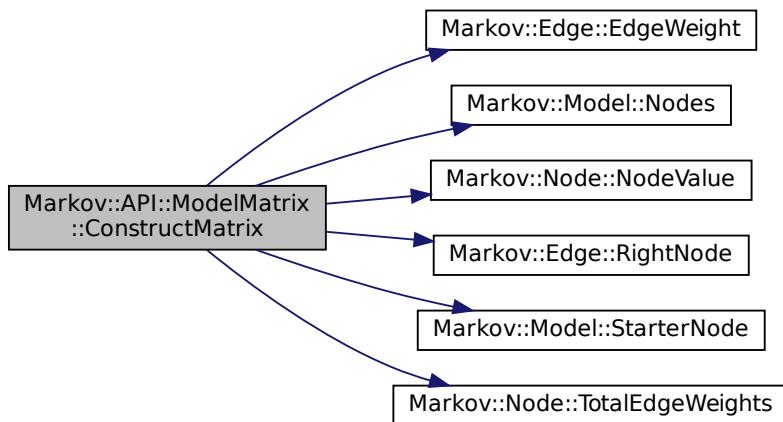
```

00076     this->ready = true;
00077     return true;
00078     //this->DumpJSON();
00079 }
```

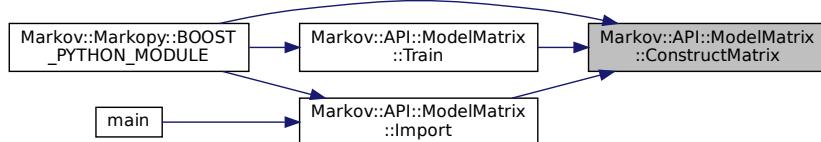
References [Markov::API::ModelMatrix::edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [Markov::API::ModelMatrix::Markov::API::ModelMatrix::matrixSize](#), [Markov::Model< NodeStorageType >::Nodes\(\)](#), [Markov::Node< storageType >::NodeValue\(\)](#), [Markov::API::ModelMatrix::ready](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::StarterNode\(\)](#), [Markov::Node< storageType >::TotalEdgeWeights\(\)](#), [Markov::API::ModelMatrix::totalEdgeWeights](#), and [Markov::API::ModelMatrix::value\(\)](#).

Referenced by [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:



### 9.6.2.5 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

<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 }
```

**9.6.2.6 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:**

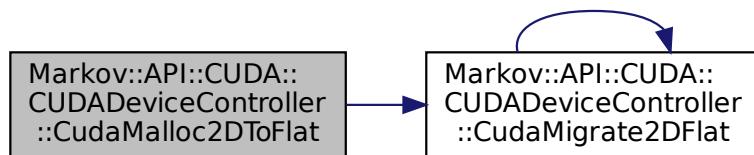
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
}
```

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:



### 9.6.2.7 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:

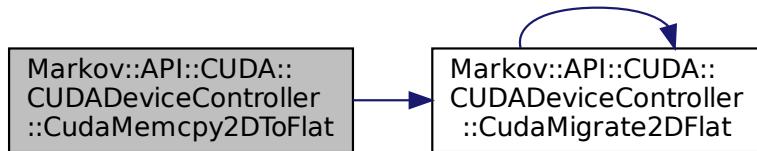
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
CudaCheckNotifyErr(cudastatus, " CudaMemcpy2DToFlat Failed.", false);
```

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:



### 9.6.2.8 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.");
```

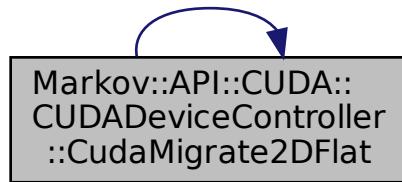
Definition at line 132 of file [cudaDeviceController.h](#).

```
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 }
```

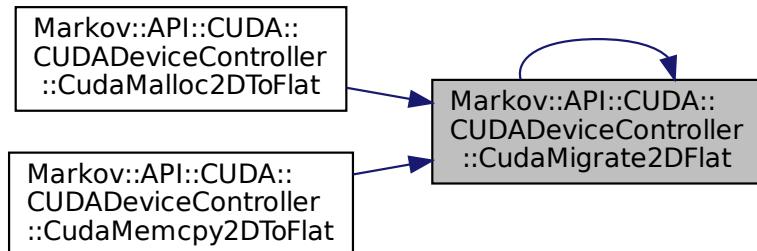
References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

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:



### 9.6.2.9 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

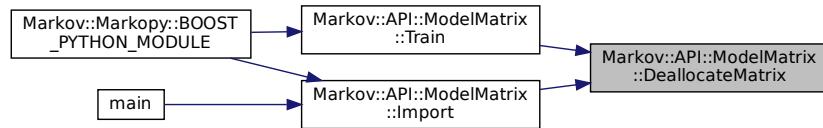
Definition at line 81 of file [modelMatrix.cpp](#).

```

00081
00082     if(!this->ready) return false;
00083     delete[] this->matrixIndex;
00084     delete[] this->totalEdgeWeights;
00085
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:



### 9.6.2.10 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](#).

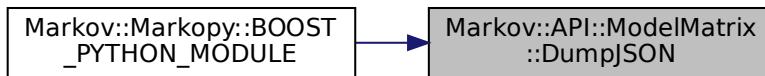
```

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"
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::BOOST\\_PYTHON\\_MODULE\(\)](#).

Here is the caller graph for this function:



### 9.6.2.11 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;`; }

### 9.6.2.12 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 300 of file `model.h`.

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

### 9.6.2.13 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 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     return true;
00298 }
```

### 9.6.2.14 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 = (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 }
```

```

00092     //start kernelID 1
00093     this->LaunchAsyncKernel(1, minLen, maxLen);
00094
00095     for(int i=1;i<numberOfPartitions;i++) {
00096         if(bInfinite) i=0;
00097
00098         //wait kernelID1 to finish, and start kernelID 0
00099         cudaStreamSynchronize(this->cudastreams[1]);
00100         this->LaunchAsyncKernel(0, minLen, maxLen);
00101
00102         //start memcpy from kernel 1 (block until done)
00103         this->GatherAsyncKernelOutput(1, bFileIO, wordlist);
00104
00105         //wait kernelID 0 to finish, then start kernelID1
00106         cudaStreamSynchronize(this->cudastreams[0]);
00107         this->LaunchAsyncKernel(1, minLen, maxLen);
00108
00109         //start memcpy from kernel 0 (block until done)
00110         this->GatherAsyncKernelOutput(0, bFileIO, wordlist);
00111
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 [alternatingKernels](#), [cudaBlocks](#), [cudaGridSize](#), [cudaMemPerGrid](#), [cudaPerKernelAllocationSize](#), [cudaThreads](#), [iterationsPerKernelThread](#), [numberOfPartitions](#), [totalOutputPerKernel](#), and [totalOutputPerSync](#).

### 9.6.2.15 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);
```

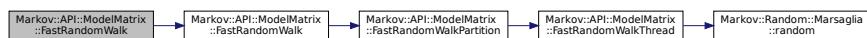
Definition at line 217 of file [modelMatrix.cpp](#).

```
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 }
```

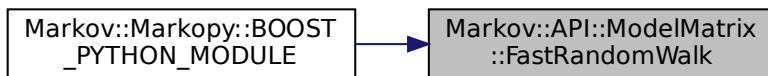
References [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Referenced by [Markov::Markopy::BOOST\\_PYTHON\\_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.6.2.16 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 %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:



### 9.6.2.17 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](#).

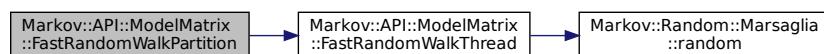
```

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 }
```

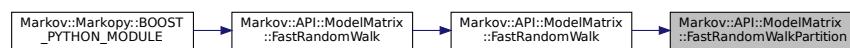
References [Markov::API::ModelMatrix::FastRandomWalkThread\(\)](#).

Referenced by [Markov::API::ModelMatrix::FastRandomWalk\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.6.2.18 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

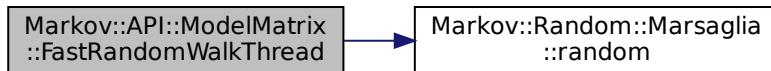
<i>mlock</i>	- mutex lock to distribute to child threads
<i>wordlist</i>	- Reference to the wordlist file to write to
<i>n</i>	- Number of passwords to generate.
<i>wordlistFileName</i>	- Filename to write to
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate
<i>id</i>	- <b>DEPRECATED</b> 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](#).

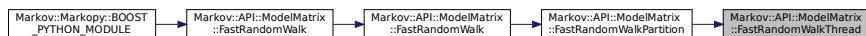
```
00153
00154     if(n==0) return;
00155
00156     Markov::Random::Marsaglia MarsagliaRandomEngine;
00157     char* e;
00158     char *res = new char[(maxLen+2)*n];
00159     int index = 0;
00160     char next;
00161     int len=0;
00162     long int selection;
00163     char cur;
00164     long int 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:



### 9.6.2.19 FlattenMatrix()

`__host__ void Markov::API::CUDA::CUDAModelMatrix::FlattenMatrix ( )`  
 Flatten migrated matrix from 2d to 1d.

Definition at line 261 of file [cudaModelMatrix.cu](#).

```

00261
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 }
```

References [flatEdgeMatrix](#), [flatValueMatrix](#), and [Markov::API::ModelMatrix::matrixSize](#).

### 9.6.2.20 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     }
```

References [cudaMemPerGrid](#), [cudaPerKernelAllocationSize](#), and [outputBuffer](#).

### 9.6.2.21 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 }
```

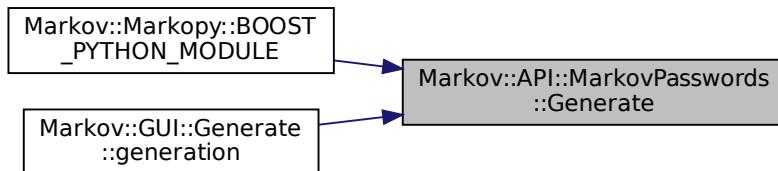
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

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:



### 9.6.2.22 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
<i>minLen</i>	- Minimum password length to generate
<i>maxLen</i>	- Maximum password length to generate

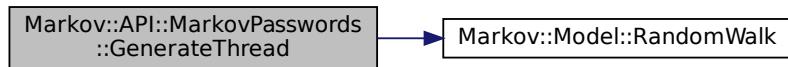
Definition at line 140 of file [markovPasswords.cpp](#).

```
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 }
```

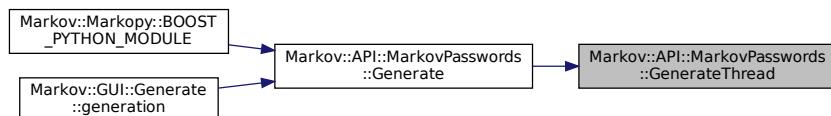
References [Markov::Model< NodeStorageType >::RandomWalk\(\)](#).

Referenced by [Markov::API::MarkovPasswords::Generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.6.2.23 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.

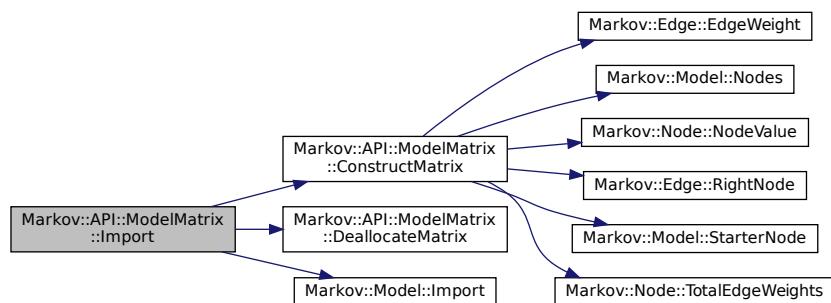
Definition at line 19 of file [modelMatrix.cpp](#).

```
00019
00020     this->DeallocateMatrix(); {
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

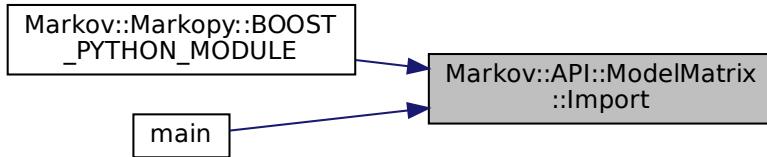
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::BOOST\\_PYTHON\\_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.6.2.24 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 216 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 }
```

### 9.6.2.25 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->cuadstreams[2]);
00174     //else cudaStreamSynchronize(this->cuadstreams[kernelID-1]);
00175     FastRandomWalkCUDAKernel<<(cudaBlocks,cudaThreads,0,
00176     this->cuadstreams[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 }
```

### 9.6.2.26 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);
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 }
```

### 9.6.2.27 MigrateMatrix()

```
__host__ void Markov::API::CUDA::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
00021     cudaError_t 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     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 }

```

### 9.6.2.28 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; }
```

### 9.6.2.29 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:



### 9.6.2.30 OptimizeEdgeOrder()

```
void Markov::Model< char >::OptimizeEdgeOrder [inherited]
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() << ", ";
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     }
```

### 9.6.2.31 prepKernelMemoryChannel()

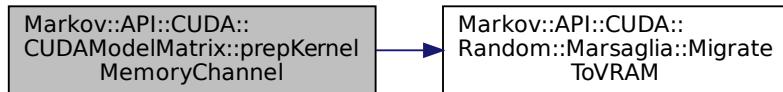
```
__host__ void Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel (
    int numberOfStreams) [protected]
```

Definition at line 145 of file [cudaModelMatrix.cu](#).

```
00145
00146
00147     this->cuadstreams = new cudaStream_t[numberOfStreams];
00148     for(int i=0;i<numberOfStreams;i++)
00149         cudaStreamCreate(&this->cuadstreams[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 }
```

References [cudaGridSize](#), [cudaPerKernelAllocationSize](#), [device\\_outputBuffer](#), [device\\_seeds](#), [Markov::API::CUDA::Random::Marsaglia](#) and [outputBuffer](#).

Here is the call graph for this function:



### 9.6.2.32 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 <a href="#">Markov::Random::Mersenne</a> and <a href="#">Markov::Random::Marsaglia</a>
<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     }
```

```

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 }
```

### 9.6.2.33 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.

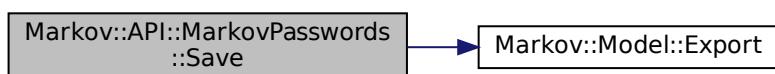
Definition at line 106 of file [markovPasswords.cpp](#).

```

00106
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:



### 9.6.2.34 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; }
```

**9.6.2.35 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.

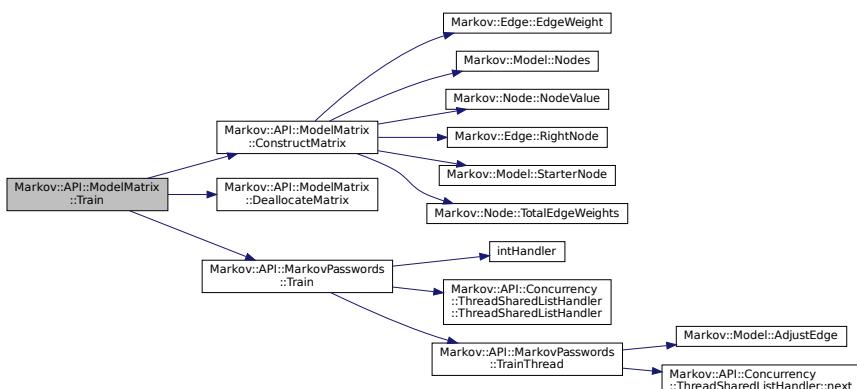
Definition at line 25 of file [modelMatrix.cpp](#).

```
00025
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

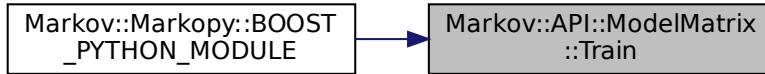
References [Markov::API::ModelMatrix::ConstructMatrix\(\)](#), [Markov::API::ModelMatrix::DeallocateMatrix\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::BOOST\\_PYTHON\\_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.6.2.36 TrainThread()

```
void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private], [inherited]
```

A single thread invoked by the Train function.

#### Parameters

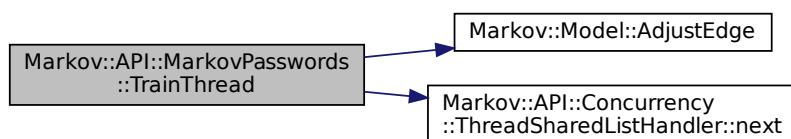
<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

Definition at line 85 of file [markovPasswords.cpp](#).

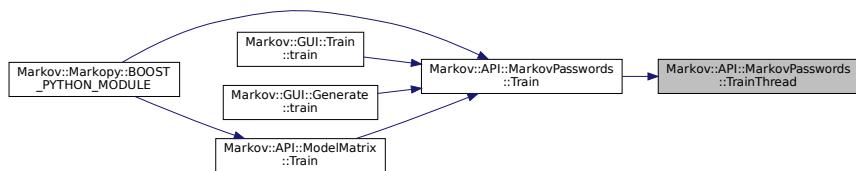
```
00085
00086     char format_str[] = "%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
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     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:



## 9.6.3 Member Data Documentation

### 9.6.3.1 alternatingKernels

`int Markov::API::CUDA::CUDAModelMatrix::alternatingKernels [private]`

Definition at line 135 of file [cudaModelMatrix.h](#).

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

### 9.6.3.2 cudaBlocks

`int Markov::API::CUDA::CUDAModelMatrix::cudaBlocks [private]`

Definition at line 125 of file [cudaModelMatrix.h](#).

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

### 9.6.3.3 cudaGridSize

`int Markov::API::CUDA::CUDAModelMatrix::cudaGridSize [private]`

Definition at line 131 of file [cudaModelMatrix.h](#).

Referenced by [FastRandomWalk\(\)](#), and [prepKernelMemoryChannel\(\)](#).

### 9.6.3.4 cudaMemPerGrid

`int Markov::API::CUDA::CUDAModelMatrix::cudaMemPerGrid [private]`

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

Referenced by [FastRandomWalk\(\)](#), and [GatherAsyncKernelOutput\(\)](#).

### 9.6.3.5 cudaPerKernelAllocationSize

`long int Markov::API::CUDA::CUDAModelMatrix::cudaPerKernelAllocationSize [private]`

Definition at line 133 of file [cudaModelMatrix.h](#).

Referenced by [FastRandomWalk\(\)](#), [GatherAsyncKernelOutput\(\)](#), and [prepKernelMemoryChannel\(\)](#).

### 9.6.3.6 cudastreams

`cudaStream_t* Markov::API::CUDA::CUDAModelMatrix::cudastreams [private]`

Definition at line 139 of file [cudaModelMatrix.h](#).

### 9.6.3.7 cudaThreads

int Markov::API::CUDA::CUDAModelMatrix::cudaThreads [private]

Definition at line 126 of file [cudaModelMatrix.h](#).

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

### 9.6.3.8 datasetFile

std::ifstream\* Markov::API::MarkovPasswords::datasetFile [private], [inherited]

Definition at line 123 of file [markovPasswords.h](#).

### 9.6.3.9 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](#).

### 9.6.3.10 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](#).

### 9.6.3.11 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\(\)](#).

### 9.6.3.12 device\_seeds

unsigned long\*\* Markov::API::CUDA::CUDAModelMatrix::device\_seeds [private]

Definition at line 137 of file [cudaModelMatrix.h](#).

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

### 9.6.3.13 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](#).

### 9.6.3.14 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](#).

### 9.6.3.15 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\(\)](#).

### 9.6.3.16 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](#).

### 9.6.3.17 flatEdgeMatrix

```
char* Markov::API::CUDA::CUDAModelMatrix::flatEdgeMatrix [private]
```

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 [FlattenMatrix\(\)](#).

### 9.6.3.18 flatValueMatrix

```
long int* Markov::API::CUDA::CUDAModelMatrix::flatValueMatrix [private]
```

Adding Value matrix end-to-end and resize to 1-D array for better performance on traversing.

Definition at line 123 of file [cudaModelMatrix.h](#).

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

### 9.6.3.19 iterationsPerKernelThread

```
int Markov::API::CUDA::CUDAModelMatrix::iterationsPerKernelThread [private]
```

Definition at line 127 of file [cudaModelMatrix.h](#).

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

### 9.6.3.20 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\(\)](#).

### 9.6.3.21 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\(\)](#).

### 9.6.3.22 modelSavefile

```
std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]
```

Dataset file input of our system

Definition at line 124 of file [markovPasswords.h](#).

### 9.6.3.23 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.
```

### 9.6.3.24 numberOfPartitions

```
int Markov::API::CUDA::CUDAModelMatrix::numberOfPartitions [private]
Definition at line 130 of file cudaModelMatrix.h.
Referenced by FastRandomWalk\(\).
```

### 9.6.3.25 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\(\).
```

### 9.6.3.26 outputFile

```
std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]
File to save model of our system
```

Definition at line 125 of file [markovPasswords.h](#).

### 9.6.3.27 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\(\).
```

### 9.6.3.28 starterNode

```
Node<char *>* Markov::Model< char >::starterNode [private], [inherited]
Starter Node of this model.
Definition at line 198 of file model.h.
```

### 9.6.3.29 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\(\).
```

### 9.6.3.30 totalOutputPerKernel

```
long int Markov::API::CUDA::CUDAModelMatrix::totalOutputPerKernel [private]
Definition at line 129 of file cudaModelMatrix.h.
Referenced by FastRandomWalk\(\).
```

---

### 9.6.3.31 totalOutputPerSync

```
long int Markov::API::CUDA::CUDAModelMatrix::totalOutputPerSync [private]
```

Definition at line 128 of file [cudaModelMatrix.h](#).

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

### 9.6.3.32 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:

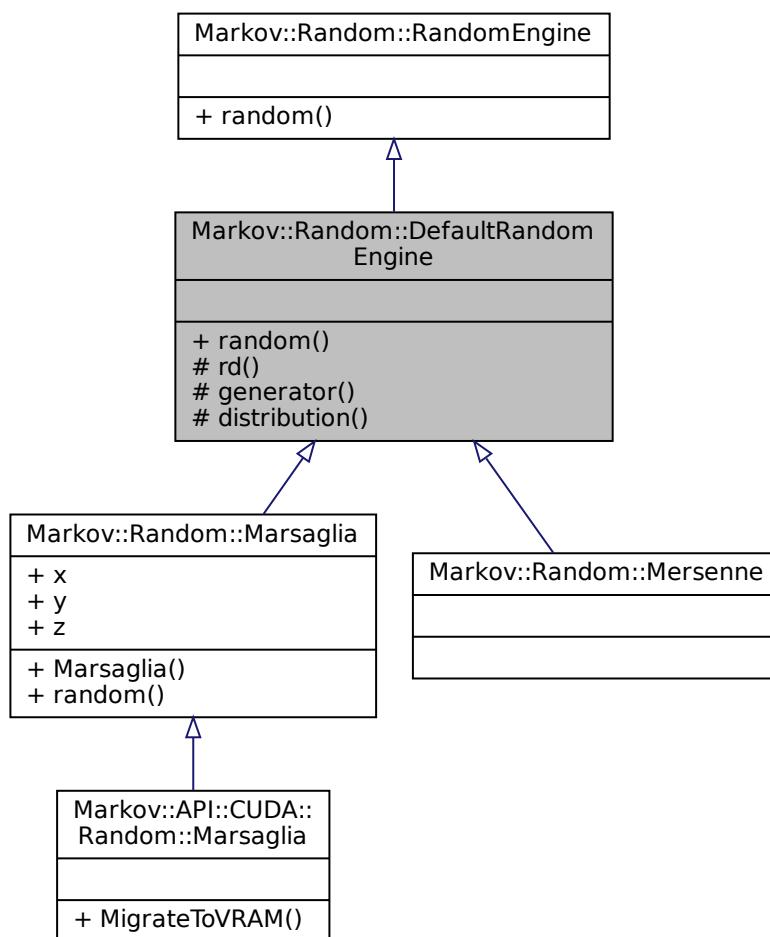
- [cudaModelMatrix.h](#)
  
- [cudaModelMatrix.cu](#)

## 9.7 Markov::Random::DefaultRandomEngine Class Reference

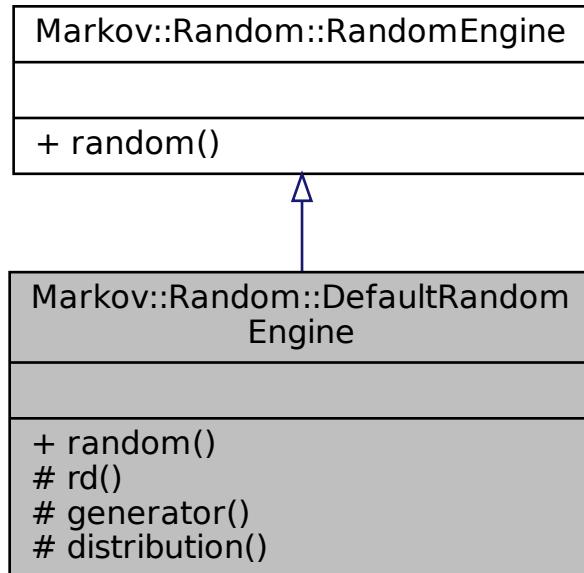
Implementation using [Random.h](#) default random engine.

```
#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.*

### 9.7.1 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](#).

## 9.7.2 Member Function Documentation

### 9.7.2.1 distribution()

```
std::uniform_int_distribution<long long unsigned>& Markov::Random::DefaultRandomEngine::distribution
( ) [inline], [protected]
```

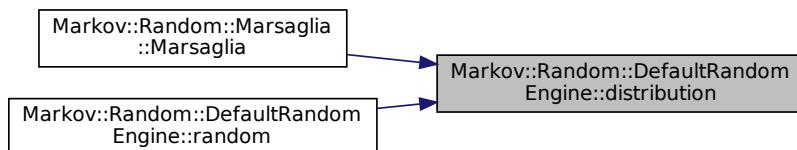
Distribution schema for seeding.

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:



### 9.7.2.2 generator()

```
std::default_random_engine& Markov::Random::DefaultRandomEngine::generator ( ) [inline],
[protected]
```

Default random engine for seeding.

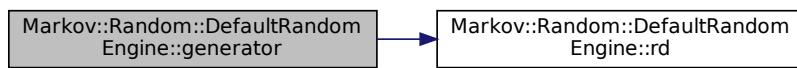
Definition at line 82 of file [random.h](#).

```
00082
00083     static std::default_random_engine _generator(rd()());
00084     return _generator;
00085 }
```

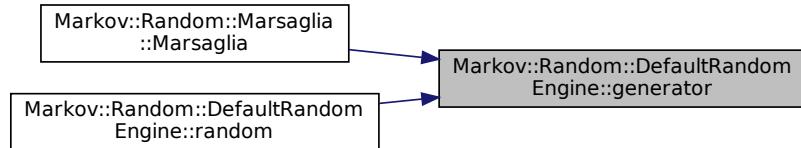
References [rd\(\)](#).

Referenced by [Markov::Random::Marsaglia::Marsaglia\(\)](#), and [random\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.7.2.3 random()

`unsigned long Markov::Random::DefaultRandomEngine::random () [inline], [virtual]`  
Generate Random Number.

Returns

random number in long range.

Implements [Markov::Random::RandomEngine](#).

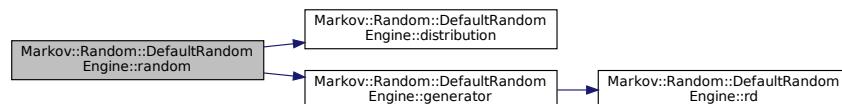
Reimplemented in [Markov::Random::Marsaglia](#).

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:



### 9.7.2.4 rd()

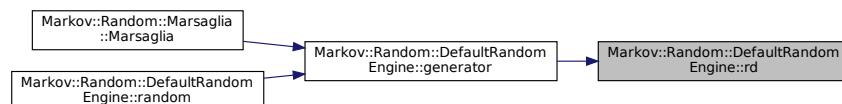
`std::random_device& Markov::Random::DefaultRandomEngine::rd () [inline], [protected]`  
Default random device for seeding.

Definition at line 74 of file [random.h](#).

```
00074     {
00075         static std::random_device _rd;
00076         return _rd;
00077     }
```

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

Here is the caller graph for this function:



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

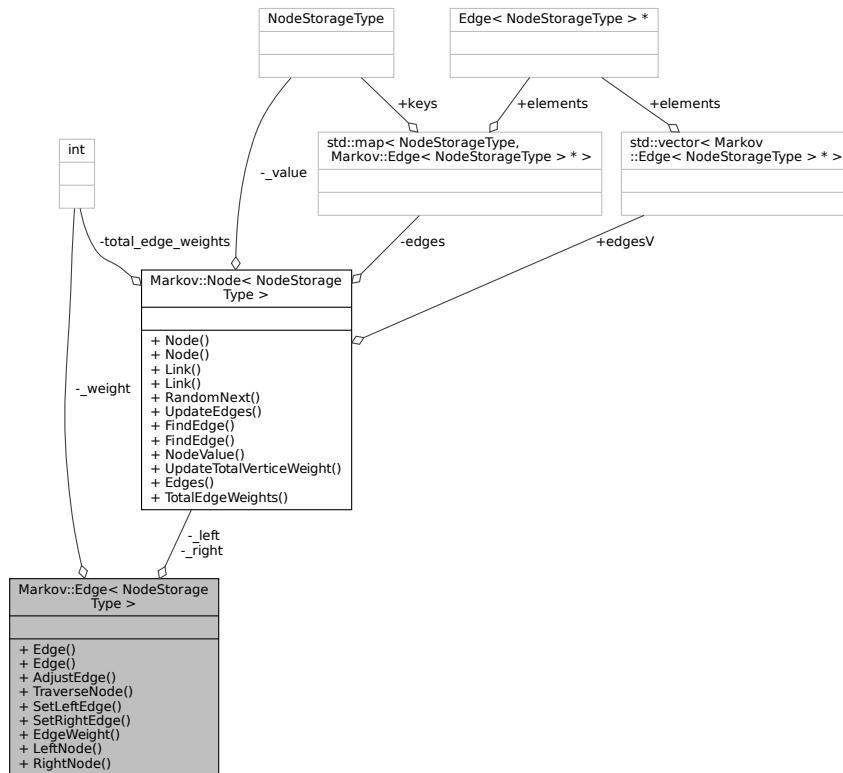
- [random.h](#)

## 9.8 Markov::Edge< NodeStorageType > Class Template Reference

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

```
#include <model.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.*

### 9.8.1 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 30 of file `model.h`.

### 9.8.2 Constructor & Destructor Documentation

#### 9.8.2.1 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 }
```

#### 9.8.2.2 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>* e1 = new Markov::Edge<unsigned char>(src, target1);
Definition at line 130 of file edge.h.
00130
00131   {
00132     this->_left = _left;
00133     this->_right = _right;
00134     this->_weight = 0;
```

```
00134 }
```

### 9.8.3 Member Function Documentation

#### 9.8.3.1 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

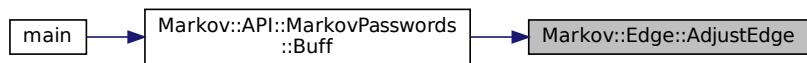
```
Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
e1->AdjustEdge(25);
```

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:



#### 9.8.3.2 EdgeWeight()

```
template<typename NodeStorageType >
uint64_t Markov::Edge< NodeStorageType >::EdgeWeight [inline]
return edge's EdgeWeight.
```

**Returns**

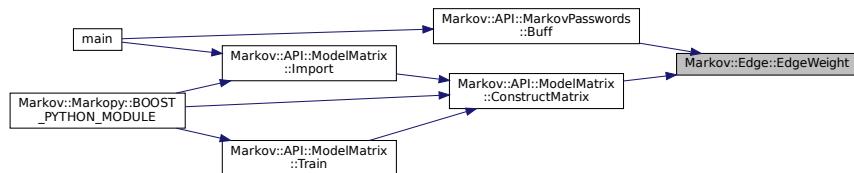
edge's EdgeWeight.

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:

**9.8.3.3 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 }
```

**9.8.3.4 RightNode()**

```
template<typename NodeStorageType >
Markov::Node< NodeStorageType > * Markov::Edge< NodeStorageType >::RightNode [inline]
return edge's RightNode
```

**Returns**

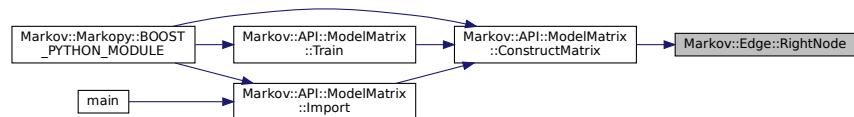
edge's RightNode.

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:



### 9.8.3.5 SetLeftEdge()

```
template<typename NodeStorageType >
void Markov::Edge< NodeStorageType >::SetLeftEdge (
    Markov::Node< NodeStorageType > * n )
Set LeftNode of this edge.
```

#### Parameters

<i>node</i>	- <a href="#">Node</a> to be linked with.
-------------	---

Definition at line 150 of file [edge.h](#).

```
00150
00151     this->left = n;
00152 }
```

### 9.8.3.6 SetRightEdge()

```
template<typename NodeStorageType >
void Markov::Edge< NodeStorageType >::SetRightEdge (
    Markov::Node< NodeStorageType > * n )
Set RightNode of this edge.
```

#### Parameters

<i>node</i>	- <a href="#">Node</a> to be linked with.
-------------	---

Definition at line 155 of file [edge.h](#).

```
00155
00156     this->right = n;
00157 }
```

### 9.8.3.7 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 }
```

## 9.8.4 Member Data Documentation

### 9.8.4.1 left

```
template<typename NodeStorageType >
Node<NodeStorageType>* Markov::Edge< NodeStorageType >::_left [private]
```

source node  
 Definition at line 105 of file [edge.h](#).

#### 9.8.4.2 \_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< char >::TraverseNode\(\).
```

#### 9.8.4.3 \_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:
```

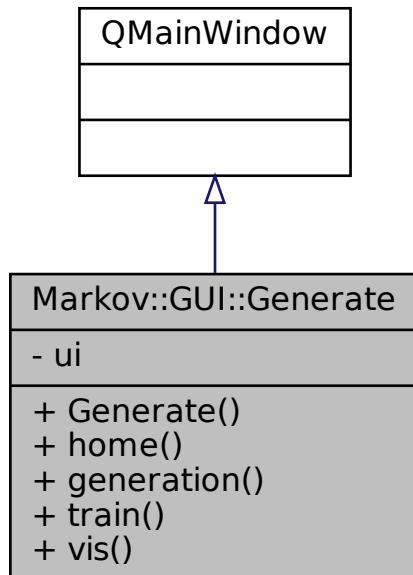
- [model.h](#)
- [edge.h](#)

## 9.9 Markov::GUI::Generate Class Reference

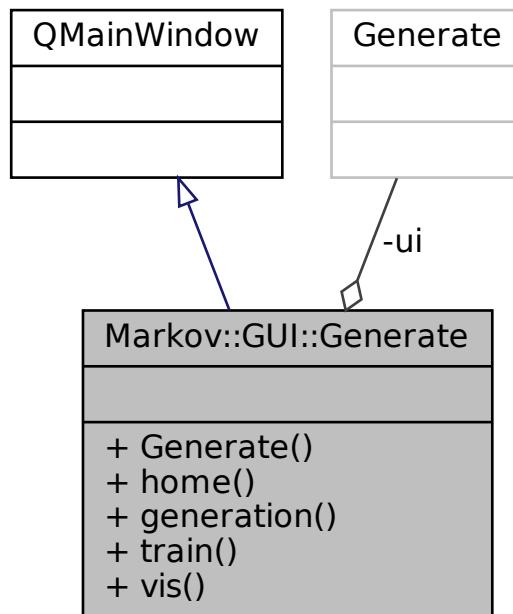
QT Generation page class.

```
#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`

### 9.9.1 Detailed Description

QT Generation page class.  
Definition at line 15 of file [Generate.h](#).

### 9.9.2 Constructor & Destructor Documentation

#### 9.9.2.1 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
00031     ui.pushButton->setVisible(false);
00032     ui.lineEdit->setVisible(false);
00033     ui.lineEdit_2->setVisible(false);
00034     ui.lineEdit_3->setVisible(false);
00035     ui.label_3->setVisible(false);
00036     ui.label_4->setVisible(false);
00037     ui.label_5->setVisible(false);
00038
00039
00040 }
```

References [ui](#).

### 9.9.3 Member Function Documentation

#### 9.9.3.1 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:



### 9.9.3.2 home

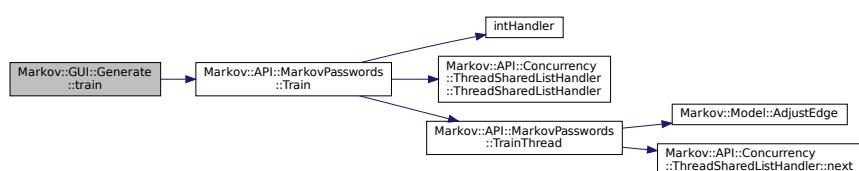
```
void Generate::home ( ) [slot]
Definition at line 121 of file Generate.cpp.
00121
00122     {
00123         CLI* w = new CLI;
00124         w->show();
00125         this->close();
00126 }
```

### 9.9.3.3 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](#).

Here is the call graph for this function:



### 9.9.3.4 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 }
```

## 9.9.4 Member Data Documentation

### 9.9.4.1 ui

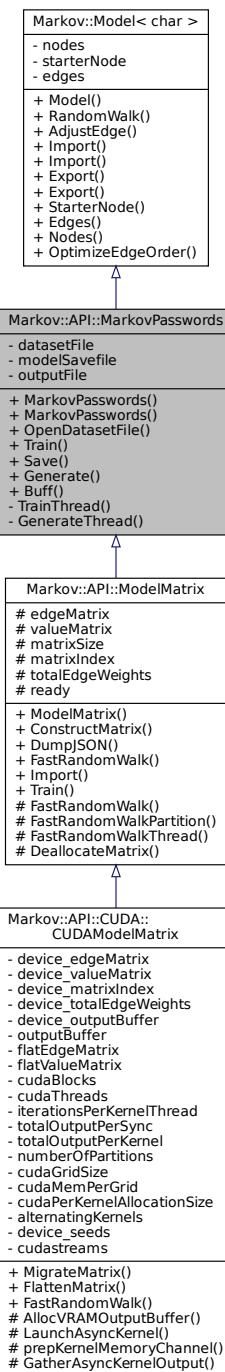
Ui::Generate Markov::GUI::Generate::ui [private]  
Definition at line 21 of file [Generate.h](#).  
Referenced by [Generate\(\)](#), [generation\(\)](#), and [train\(\)](#).  
The documentation for this class was generated from the following files:

- [Generate.h](#)
- [Generate.cpp](#)

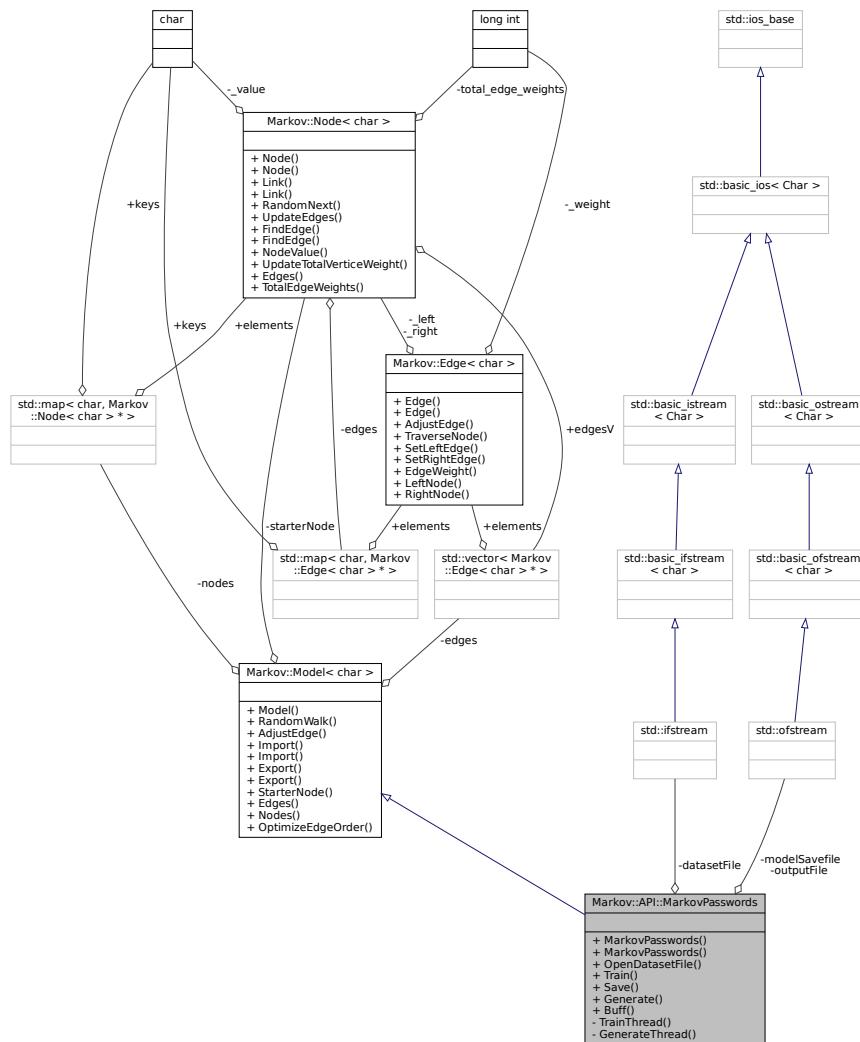
## 9.10 Markov::API::MarkovPasswords Class Reference

[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 * 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.*

### 9.10.1 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`.

## 9.10.2 Constructor & Destructor Documentation

### 9.10.2.1 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 }
```

### 9.10.2.2 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

<i>filename</i>	- 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:



## 9.10.3 Member Function Documentation

### 9.10.3.1 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

<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
00355 }
```

#### 9.10.3.2 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

<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 }
```

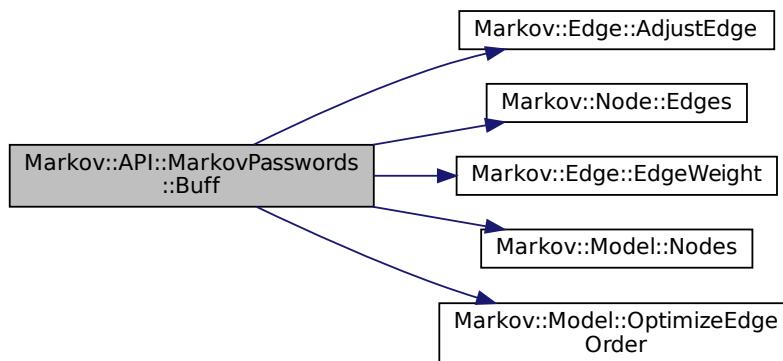
```

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
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:



### 9.10.3.3 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; }
```

### 9.10.3.4 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 300 of file [model.h](#).

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

### 9.10.3.5 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 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         *f << e->LeftNode() ->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode() ->NodeValue() <<
00294         "\n";
00295     }
00296     return true;
00297 }
```

### 9.10.3.6 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](#).

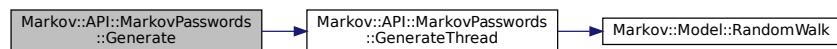
```

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
00138 }
```

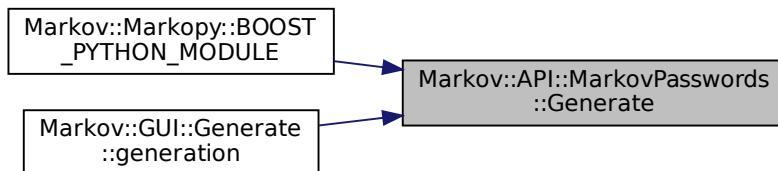
References [GenerateThread\(\)](#).

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:



### 9.10.3.7 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:



#### 9.10.3.8 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 280 of file model.h.**

```
00280
00281     std::ifstream importfile;
00282     importfile.open(filename);
00283     return this->Import(&importfile);
00284
00285 }
```

**9.10.3.9 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 216 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 }
```

### 9.10.3.10 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; }
```

### 9.10.3.11 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

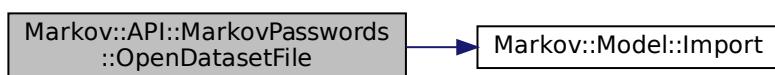
Definition at line 51 of file [markovPasswords.cpp](#).

```
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:



### 9.10.3.12 OptimizeEdgeOrder()

`void Markov::Model< char >::OptimizeEdgeOrder [inherited]`

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() << ", ";
00274     //std::cout << "\n";
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 }

```

### 9.10.3.13 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 <a href="#">Markov::Random::Mersenne</a> and <a href="#">Markov::Random::Marsaglia</a>
<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         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   //do something with the generated string
00329   return buffer; //for now
00330 }
00331
00332 }
```

### 9.10.3.14 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:



### 9.10.3.15 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; }
```

### 9.10.3.16 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");
```

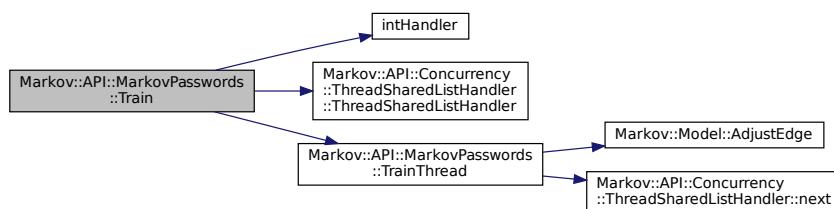
Definition at line 65 of file [markovPasswords.cpp](#).

```
00065     signal(SIGINT, intHandler);
00066     signal(SIGTERM, 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 }
```

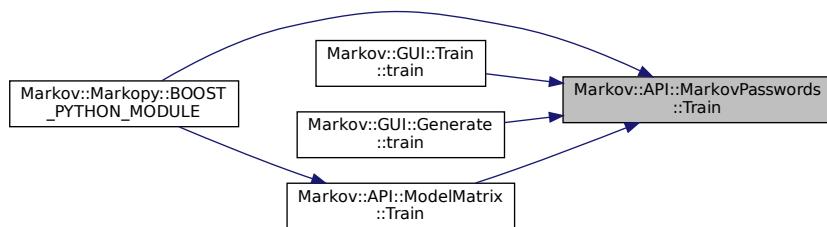
References [intHandler\(\)](#), [Markov::API::Concurrency::ThreadSharedListHandler::ThreadSharedListHandler\(\)](#), and [TrainThread\(\)](#).

Referenced by [Markov::Markopy::BOOST\\_PYTHON\\_MODULE\(\)](#), [Markov::GUI::Train::train\(\)](#), [Markov::GUI::Generate::train\(\)](#), and [Markov::API::ModelMatrix::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.10.3.17 TrainThread()

```
void Markov::API::MarkovPasswords::TrainThread (
    Markov::API::Concurrency::ThreadSharedListHandler * listhandler,
    char delimiter ) [private]
```

A single thread invoked by the Train function.

#### Parameters

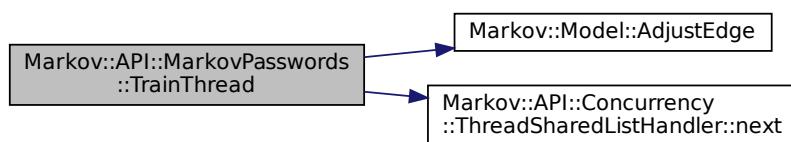
<i>listhandler</i>	- Listhandler class to read corpus from
<i>delimiter</i>	- a character, same as the delimiter in dataset content

Definition at line 85 of file markovPasswords.cpp.

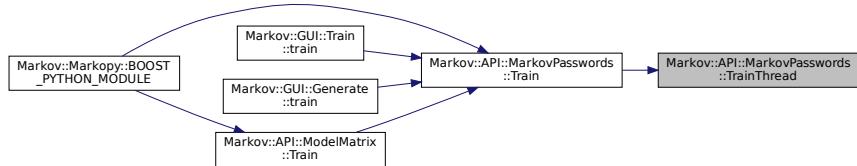
```
00085
00086     char format_str[] = "%ld,%s";
00087     format_str[3]=delimiter;
00088     std::string line;
00089     while (listhandler->next(&line) && keepRunning) {
00090         long int oc;
00091         if (line.size() > 100) {
00092             line = line.substr(0, 100);
00093         }
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 [Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 9.10.4 Member Data Documentation

### 9.10.4.1 datasetFile

```
std::ifstream* Markov::API::MarkovPasswords::datasetFile [private]
Definition at line 123 of file markovPasswords.h.
```

### 9.10.4.2 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.
```

### 9.10.4.3 modelSavefile

```
std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private]
Dataset file input of our system
```

Definition at line 124 of file markovPasswords.h.

### 9.10.4.4 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.
```

### 9.10.4.5 outputFile

```
std::ofstream* Markov::API::MarkovPasswords::outputFile [private]
File to save model of our system
```

Definition at line 125 of file markovPasswords.h.

### 9.10.4.6 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:

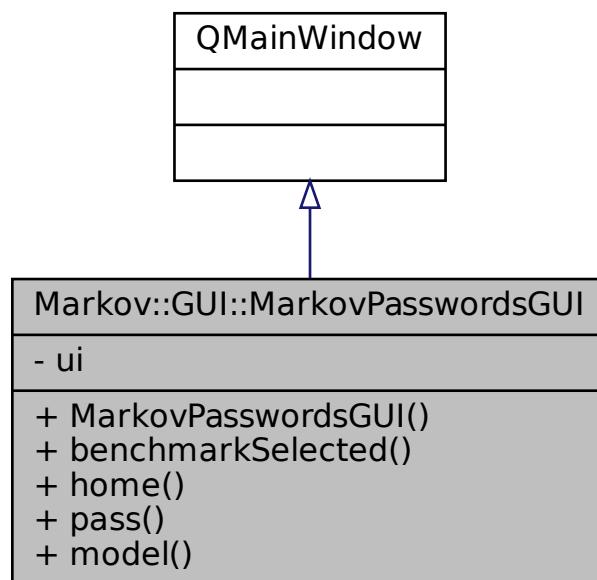
- markovPasswords.h
- markovPasswords.cpp

## 9.11 Markov::GUI::MarkovPasswordsGUI Class Reference

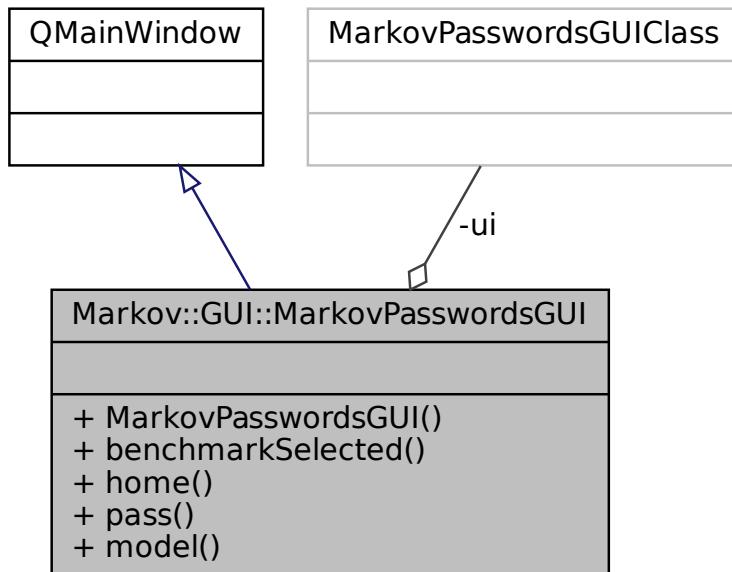
Reporting UI.

```
#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`

### 9.11.1 Detailed Description

Reporting UI.

UI for reporting and debugging tools for MarkovPassword

Definition at line 19 of file [MarkovPasswordsGUI.h](#).

### 9.11.2 Constructor & Destructor Documentation

#### 9.11.2.1 `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](#).

### 9.11.3 Member Function Documentation

#### 9.11.3.1 benchmarkSelected

```
void Markov::GUI::MarkovPasswordsGUI::benchmarkSelected () [slot]
```

#### 9.11.3.2 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 }

```

#### 9.11.3.3 model

```

void MarkovPasswordsGUI::model () [slot]
Definition at line 42 of file MarkovPasswordsGUI.cpp.
00042
00043     {
00044         QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00045
00046         //get working directory
00047         char path[255];
00048         GetCurrentDirectoryA(255, path);
00049
00050         //get absolute path to the layout html
00051         std::string layout = "file:///\" + std::string(path) + "\\views\\index.html";
00052         std::replace(layout.begin(), layout.end(), '\\', '/');
00053         webkit->setUrl(QUrl(layout.c_str()));
00054     }

```

#### 9.11.3.4 pass

```

void MarkovPasswordsGUI::pass () [slot]
Definition at line 29 of file MarkovPasswordsGUI.cpp.
00029
00030     {
00031         QWebEngineView* webkit = ui.centralWidget->findChild<QWebEngineView*>("chartArea");
00032
00033         //get working directory
00034         char path[255];
00035         GetCurrentDirectoryA(255, path);
00036
00037         //get absolute path to the layout html
00038         std::string layout = "file:///\" + std::string(path) + "\\views\\bar.html";
00039         std::replace(layout.begin(), layout.end(), '\\', '/');
00040         webkit->setUrl(QUrl(layout.c_str()));
00041     }

```

### 9.11.4 Member Data Documentation

### 9.11.4.1 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:

- [MarkovPasswordsGUI.h](#)

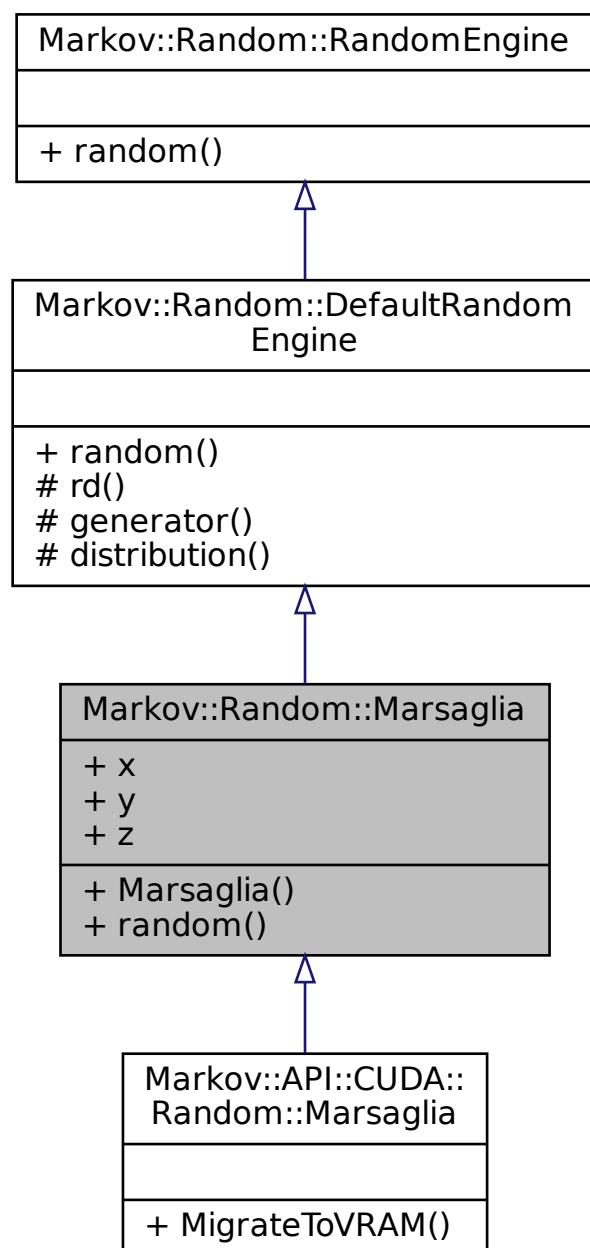
- [MarkovPasswordsGUI.cpp](#)

## 9.12 Markov::Random::Marsaglia Class Reference

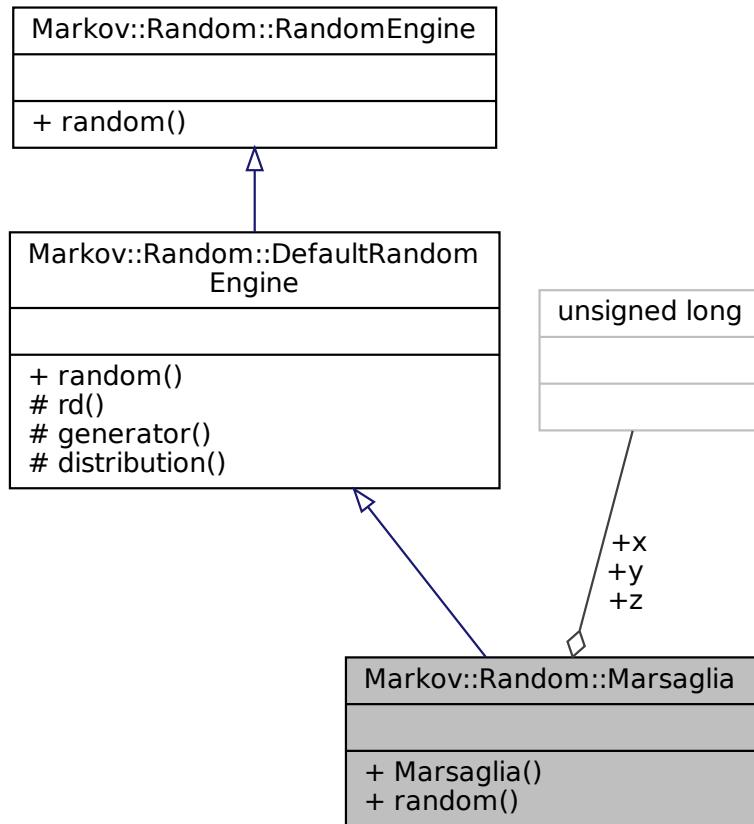
Implementation of [Marsaglia Random](#) Engine.

```
#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.*

## 9.12.1 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](#).

## 9.12.2 Constructor & Destructor Documentation

### 9.12.2.1 Marsaglia()

[Markov::Random::Marsaglia::Marsaglia](#) ( ) [inline]

Construct [Marsaglia](#) Engine.

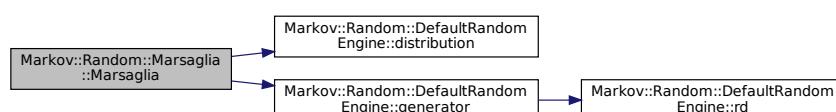
Initialize [x](#),[y](#) and [z](#) using the default random engine.

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:



## 9.12.3 Member Function Documentation

### 9.12.3.1 distribution()

[std::uniform\\_int\\_distribution<long long unsigned>](#)& [Markov::Random::DefaultRandomEngine::distribution](#) ( ) [inline], [protected], [inherited]

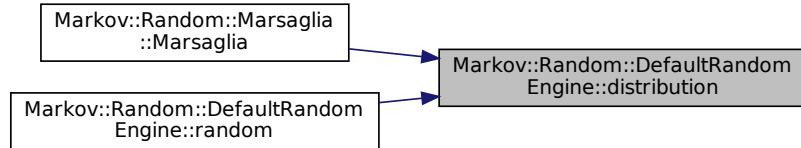
Distribution schema for seeding.

Definition at line [90](#) of file [random.h](#).

```
00090
00091     static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffffff);
00092     return _distribution;
00093 }
```

Referenced by [Marsaglia\(\)](#), and [Markov::Random::DefaultRandomEngine::random\(\)](#).

Here is the caller graph for this function:



### 9.12.3.2 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     static std::default_random_engine _generator(rd()());
00083     return _generator;
00084 }
00085 }
```

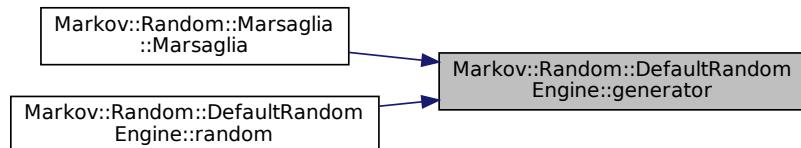
References [Markov::Random::DefaultRandomEngine::rd\(\)](#).

Referenced by [Marsaglia\(\)](#), and [Markov::Random::DefaultRandomEngine::random\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.12.3.3 random()

`unsigned long Markov::Random::Marsaglia::random ( ) [inline], [virtual]`

Generate Random Number.

**Returns**

random number in long range.

Reimplemented from [Markov::Random::DefaultRandomEngine](#).

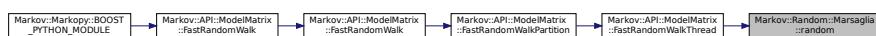
Definition at line 140 of file [random.h](#).

```
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:

**9.12.3.4 rd()**

```
std::random_device& Markov::Random::DefaultRandomEngine::rd () [inline], [protected], [inherited]
```

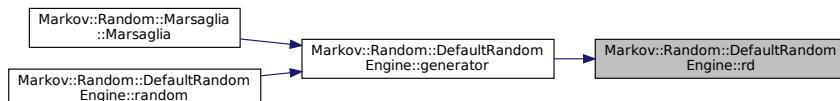
Default random device for seeding.

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:

**9.12.4 Member Data Documentation****9.12.4.1 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\(\)](#).

**9.12.4.2 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\(\)](#).

### 9.12.4.3 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:

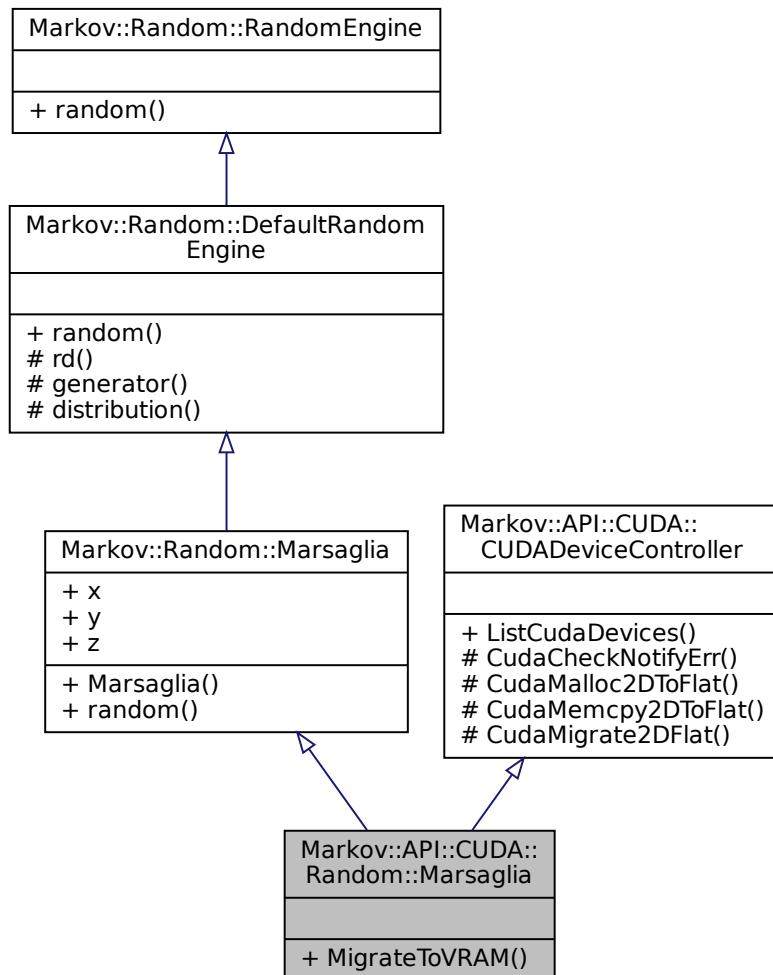
- [random.h](#)

## 9.13 Markov::API::CUDA::Random::Marsaglia Class Reference

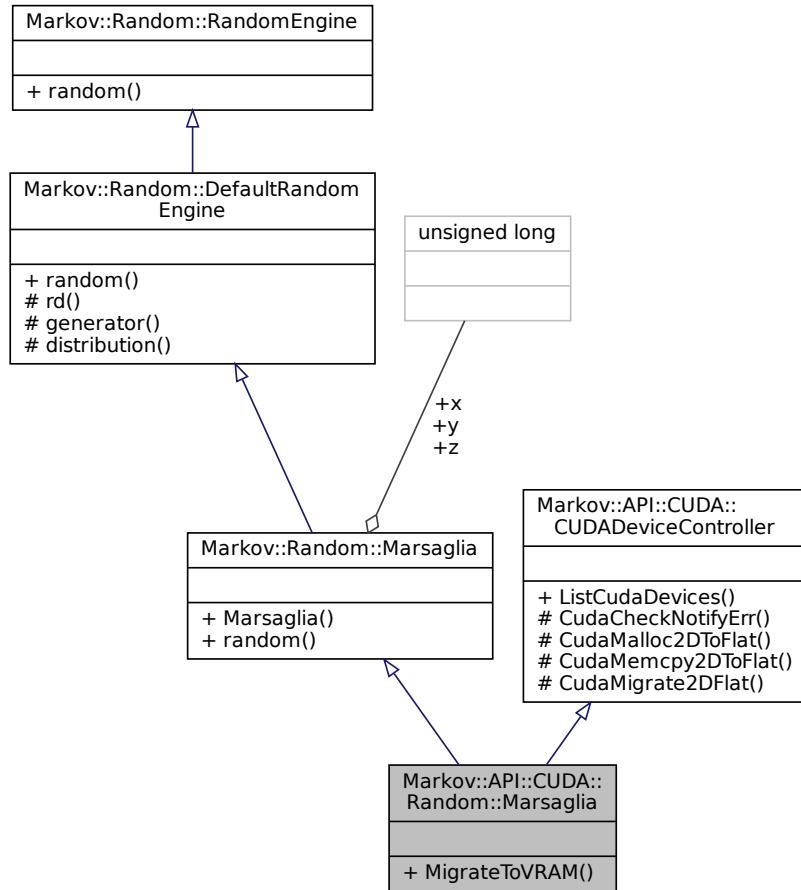
Extension of [Markov::Random::Marsaglia](#) which is capable of working on **device** space.

#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 ()`

- 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.*
  - 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.*

### 9.13.1 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](#).

### 9.13.2 Member Function Documentation

#### 9.13.2.1 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

<u>_status</u>	Cuda error status to check
<u>msg</u>	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     }
```

**9.13.2.2 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**

<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:**

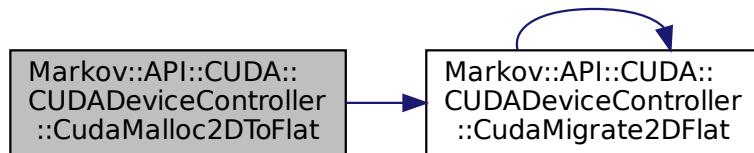
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
if(cudastatus!=cudaSuccess){
    CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
}
```

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:



### 9.13.2.3 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:

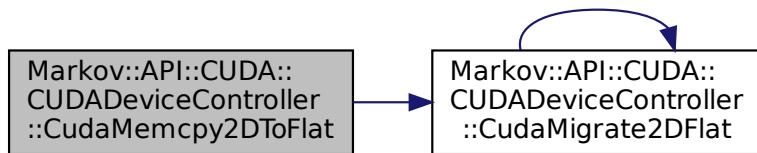
```
cudaError_t cudastatus;
char* dst;
cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
CudaCheckNotifyErr(cudastatus, " CudaMemcpy2DToFlat Failed.", false);
```

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:



#### 9.13.2.4 CudaMigrate2DFlat()

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

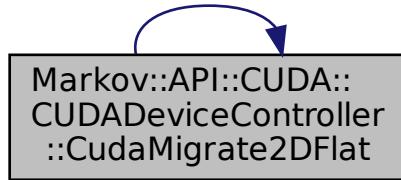
Definition at line 132 of file [cudaDeviceController.h](#).

```
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 }
```

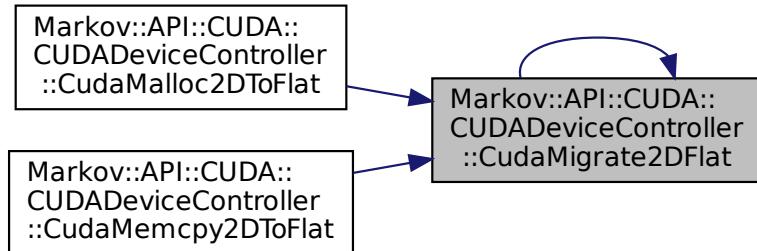
References [Markov::API::CUDA::CUDADeviceController::CudaMigrate2DFlat\(\)](#).

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:



### 9.13.2.5 distribution()

```
std::uniform_int_distribution<long long unsigned>& Markov::Random::DefaultRandomEngine::distribution()
  () [inline], [protected], [inherited]
```

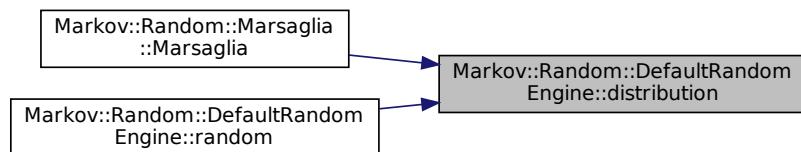
Distribution schema for seeding.

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:



### 9.13.2.6 generator()

```
std::default_random_engine& Markov::Random::DefaultRandomEngine::generator ( ) [inline],  
[protected], [inherited]
```

Default random engine for seeding.

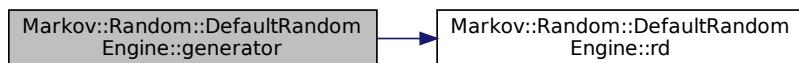
Definition at line 82 of file [random.h](#).

```
00082         static std::default_random_engine _generator(rd()());  
00083         return _generator;  
00084     }  
00085 }
```

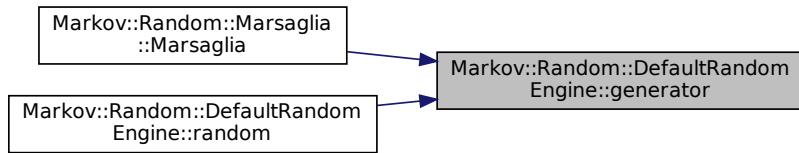
References [Markov::Random::DefaultRandomEngine::rd\(\)](#).

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:



### 9.13.2.7 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         }
```

### 9.13.2.8 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.

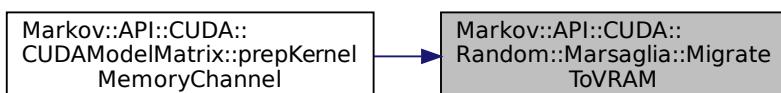
Definition at line 28 of file [cudarandom.h](#).

```
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:



### 9.13.2.9 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](#).

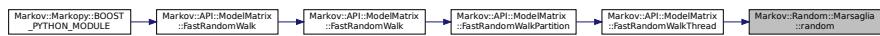
```
00140
00141     unsigned long t;
00142     x ^= x << 16;
00143     x ^= x >> 5;
00144     x ^= x << 1;
00145
00146     t = x;
```

```

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:



### 9.13.2.10 rd()

`std::random_device& Markov::Random::DefaultRandomEngine::rd () [inline], [protected], [inherited]`  
Default random device for seeding.

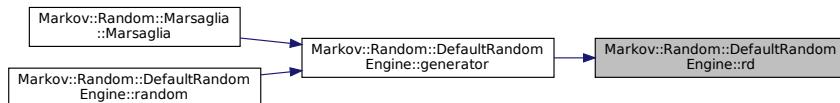
Definition at line 74 of file [random.h](#).

```

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:



## 9.13.3 Member Data Documentation

### 9.13.3.1 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\(\)](#).

### 9.13.3.2 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\(\)](#).

### 9.13.3.3 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:

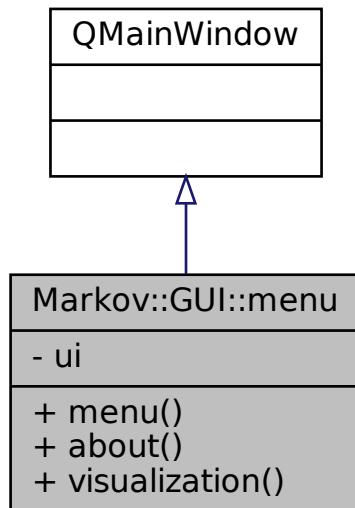
- [cudarandom.h](#)

## 9.14 Markov::GUI::menu Class Reference

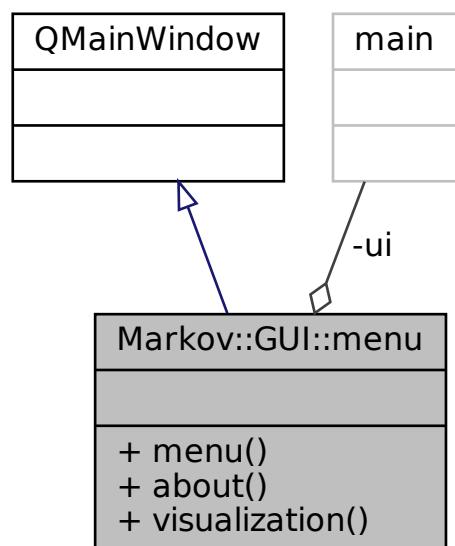
QT Menu class.

```
#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](#)

### 9.14.1 Detailed Description

QT Menu class.

Definition at line [15](#) of file [menu.h](#).

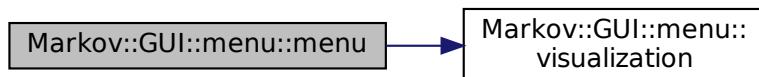
### 9.14.2 Constructor & Destructor Documentation

#### 9.14.2.1 menu()

```
menu::menu (
    QWidget * parent = Q_NULLPTR )
Definition at line 14 of file menu.cpp.
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 }
```

References [ui](#), and [visualization\(\)](#).

Here is the call graph for this function:



### 9.14.3 Member Function Documentation

#### 9.14.3.1 about

```
void menu::about ( ) [slot]
```

Definition at line [23](#) of file [menu.cpp](#).

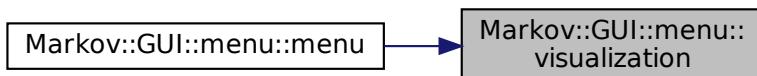
```
00023 {
00024
00025
00026 }
```

### 9.14.3.2 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:



### 9.14.4 Member Data Documentation

#### 9.14.4.1 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:

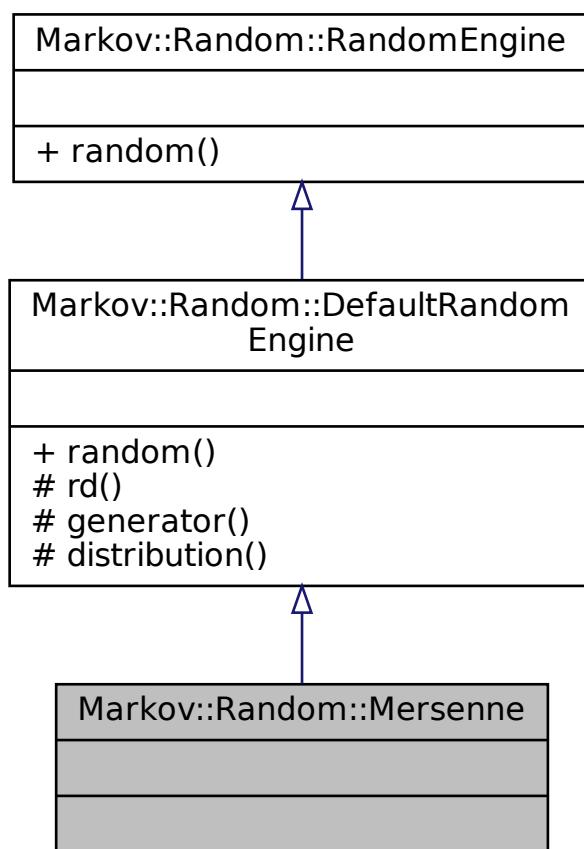
- [menu.h](#)
- [menu.cpp](#)

## 9.15 Markov::Random::Mersenne Class Reference

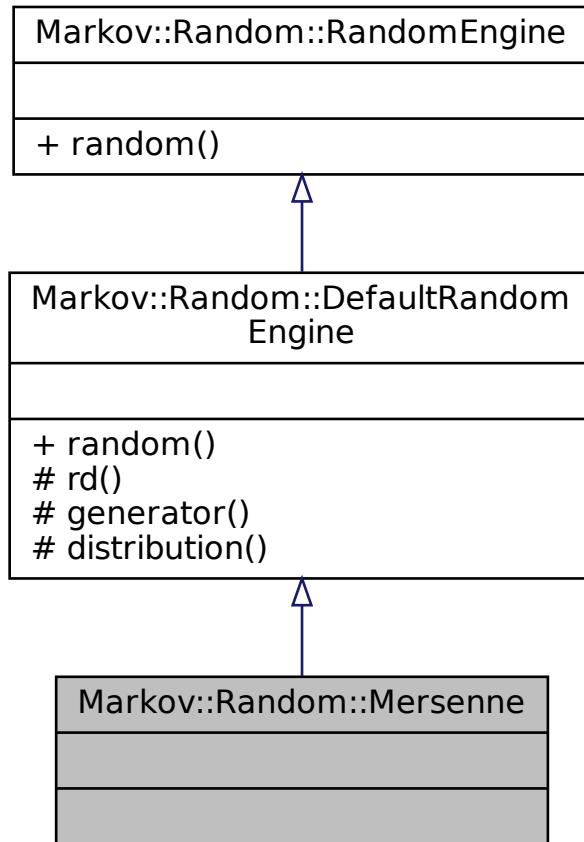
Implementation of [Mersenne](#) Twister Engine.

```
#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.*

### 9.15.1 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](#).

## 9.15.2 Member Function Documentation

### 9.15.2.1 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](#).

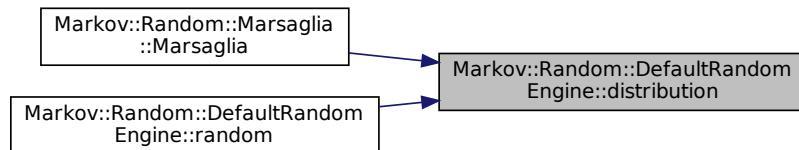
```

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:



### 9.15.2.2 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](#).

```

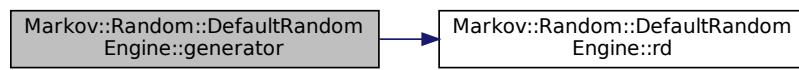
00082
00083     static std::default_random_engine _generator(rd()());
00084     return _generator;
00085 }

```

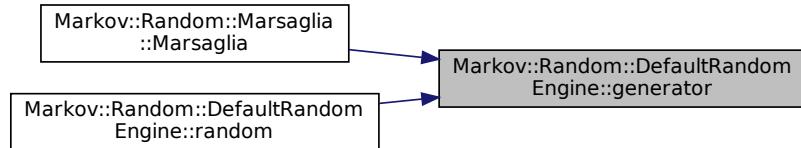
References [Markov::Random::DefaultRandomEngine::rd\(\)](#).

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:



### 9.15.2.3 random()

`unsigned long Markov::Random::DefaultRandomEngine::random () [inline], [virtual], [inherited]`  
Generate Random Number.

Returns

random number in long range.

Implements [Markov::Random::RandomEngine](#).

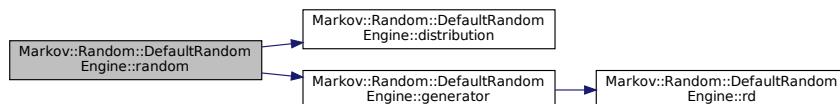
Reimplemented in [Markov::Random::Marsaglia](#).

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:



### 9.15.2.4 rd()

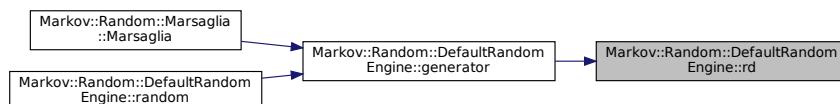
`std::random_device& Markov::Random::DefaultRandomEngine::rd () [inline], [protected], [inherited]`  
Default random device for seeding.

Definition at line 74 of file [random.h](#).

```
00074     static std::random_device _rd;
00075     return _rd;
00076 }
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:

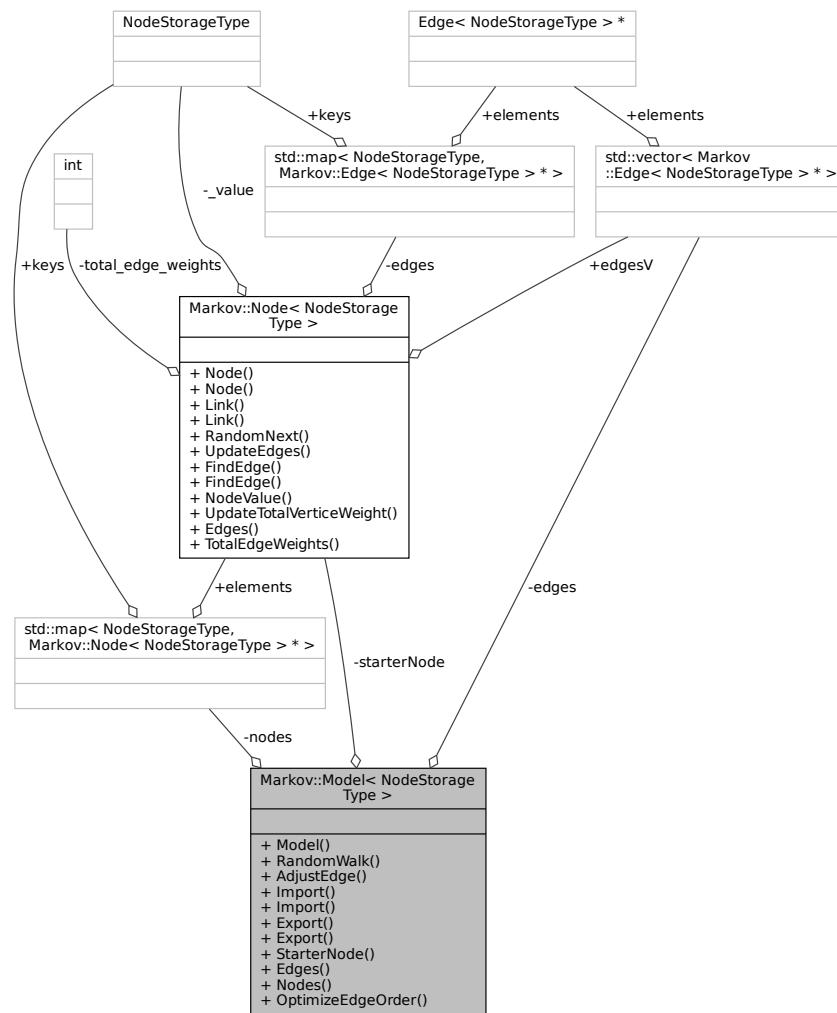
- [random.h](#)

## 9.16 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 \(const NodeStorageType \\*payload, long int occurrence\)](#)  
*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< 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.*

### 9.16.1 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](#).

### 9.16.2 Constructor & Destructor Documentation

#### 9.16.2.1 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 }
```

### 9.16.3 Member Function Documentation

### 9.16.3.1 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

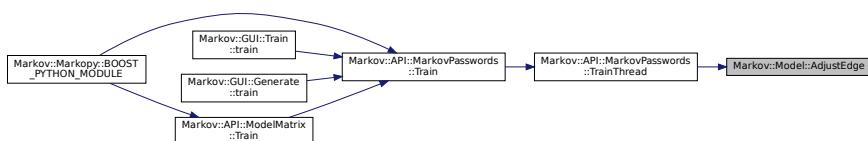
string	- String that is passed from the training, and will be used to AdjustEdge the model with
occurrence	- Occurrence of this string.

Definition at line 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:



### 9.16.3.2 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; }
```

### 9.16.3.3 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

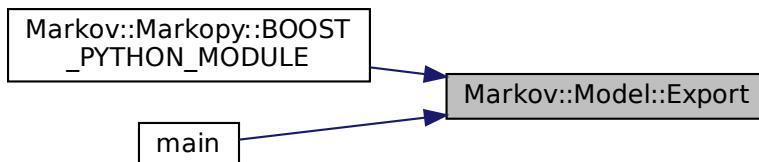
```
Markov::Model<char> model;
model.Export("test.mdl");
```

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::BOOST\\_PYTHON\\_MODULE\(\)](#), and [main\(\)](#).

Here is the caller graph for this function:



### 9.16.3.4 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 ofstream

```
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         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <<
00294         "\n";
00295     }
00296     return true;
00297 }
```

Referenced by [Markov::API::MarkovPasswords::Save\(\)](#).

Here is the caller graph for this function:



### 9.16.3.5 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

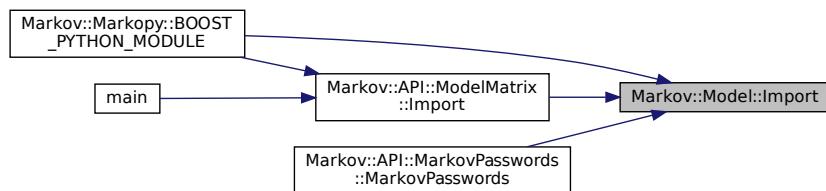
```
Markov::Model<char> model;
model.Import("test.mdl");
```

**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:



### 9.16.3.6 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

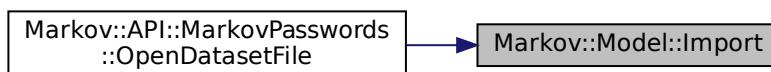
```
Markov::Model<char> model;
std::ifstream file("test.mdl");
model.Import(&file);
```

**Definition at line 216 of file model.h.**

```
00216     std::string cell;
00217     {
00218         std::string cell;
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 }
```

Referenced by [Markov::API::MarkovPasswords::OpenDatasetFile\(\)](#).

Here is the caller graph for this function:

**9.16.3.7 Nodes()**

```
template<typename NodeStorageType >
std::map<NodeStorageType, Node<NodeStorageType>*>* Markov::Model< NodeStorageType >::Nodes (
) [inline]
Return starter Node.
```

**Returns**

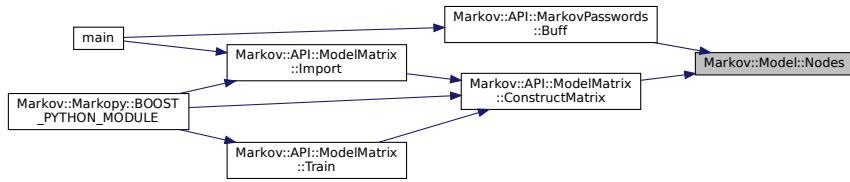
starter node with 00 NodeValue

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:

**9.16.3.8 OptimizeEdgeOrder()**

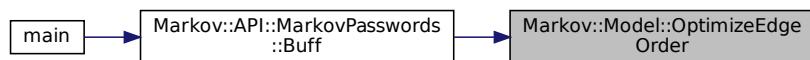
```
template<typename NodeStorageType >
void Markov::Model< NodeStorageType >::OptimizeEdgeOrder
Sort edges of all nodes in the model ordered by edge weights.
```

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() << ", ";
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 }
```

Referenced by [Markov::API::MarkovPasswords::Buff\(\)](#).

Here is the caller graph for this function:

**9.16.3.9 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 <a href="#">Markov::Random::Mersenne</a> and <a href="#">Markov::Random::Marsaglia</a>
<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

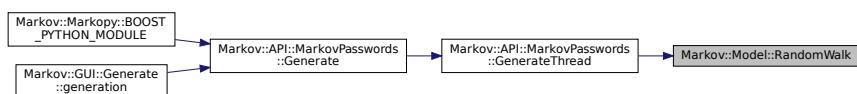
Null terminated string that was generated.

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         else if (temp_node == NULL) {
00320             break;
00321         }
00322         n = temp_node;
00323         n->RandomWalk(randomEngine, minSetting, maxSetting, buffer, len);
00324         buffer[len] = n->NodeValue();
00325         len++;
00326     }
00327     //null terminate the string
00328     buffer[len] = 0x00;
00329     //do something with the generated string
00330     return buffer; //for now
00331
00332
00333
00334 }
```

Referenced by [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

Here is the caller graph for this function:



### 9.16.3.10 StarterNode()

```
template<typename NodeStorageType >
Node<NodeStorageType>* Markov::Model< NodeStorageType >::StarterNode ( ) [inline]
Return starter Node.
```

Returns

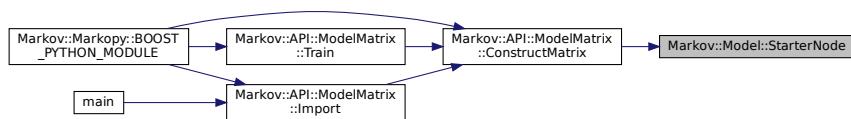
starter node with 00 NodeValue

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:



## 9.16.4 Member Data Documentation

### 9.16.4.1 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\(\)](#).

### 9.16.4.2 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\(\)](#).

### 9.16.4.3 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:

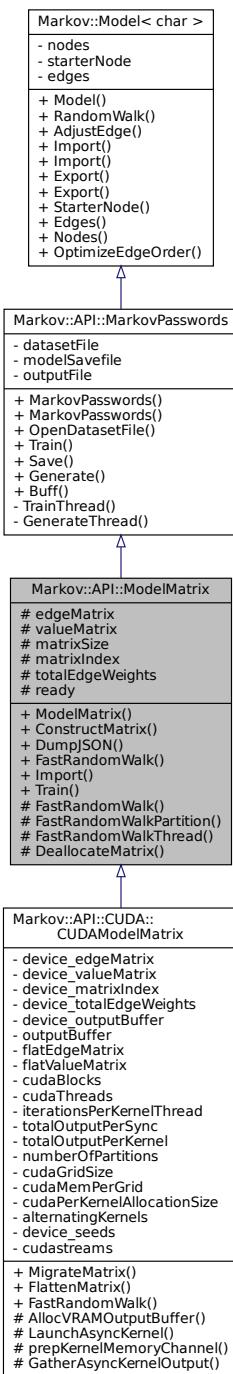
- [model.h](#)

## 9.17 Markov::API::ModelMatrix Class Reference

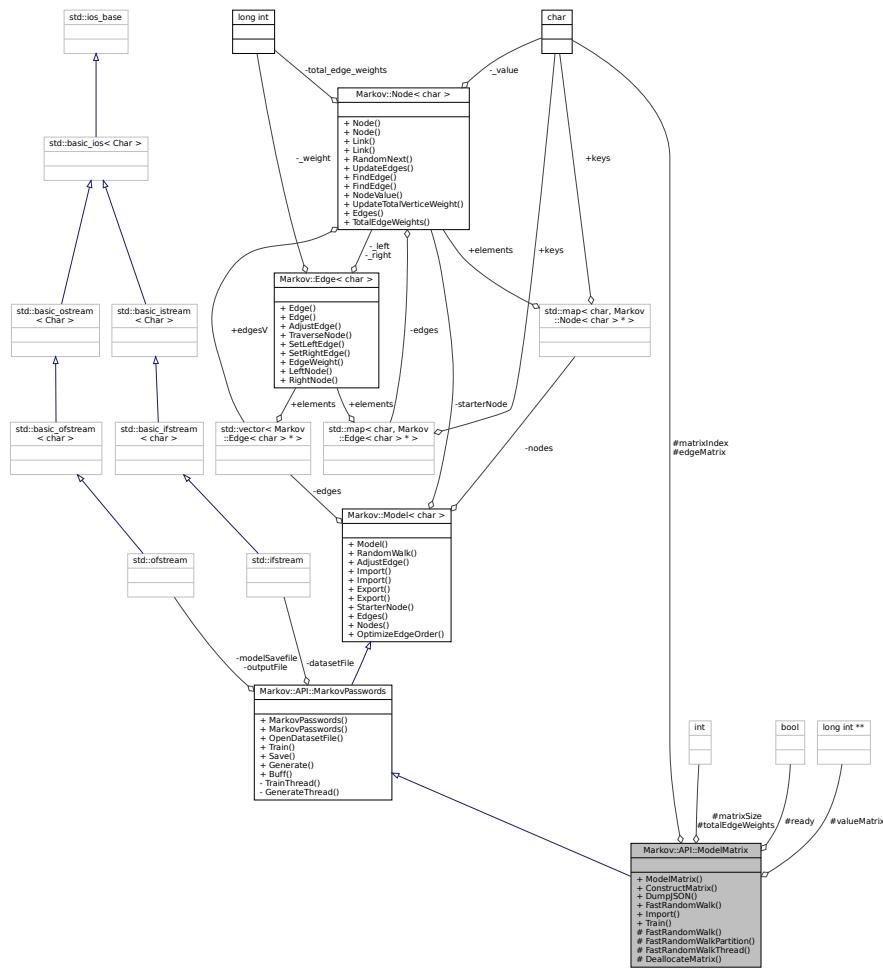
Class to flatten and reduce [Markov::Model](#) to a Matrix.

```
#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](#)

*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

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](#).

### 9.17.2 Constructor & Destructor Documentation

#### 9.17.2.1 ModelMatrix()

```
Markov::API::ModelMatrix::ModelMatrix ( )
Definition at line 15 of file modelMatrix.cpp.
00015
00016     this->ready = false;
00017 }
```

References [ready](#).

### 9.17.3 Member Function Documentation

#### 9.17.3.1 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 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 }
```

#### 9.17.3.2 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
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                 }
00169                 long int weight = edge->EdgeWeight();
00170                 weight = weight*multiplier;
```

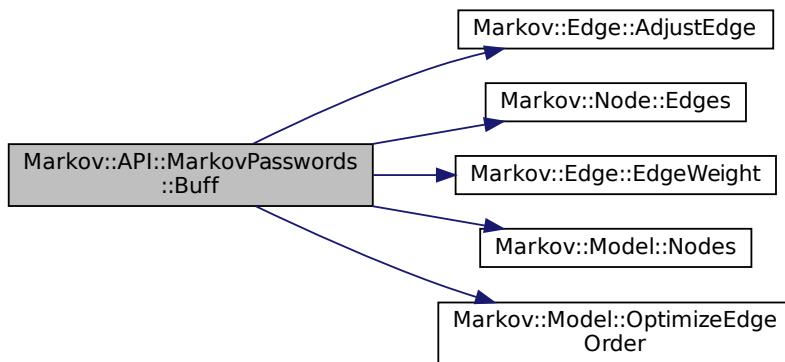
```

00171             edge->AdjustEdge(weight);
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:



### 9.17.3.3 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 }
```

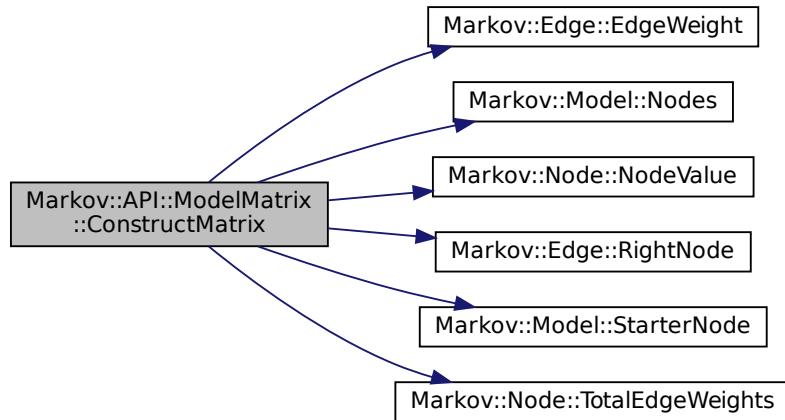
References [edgeMatrix](#), [Markov::Edge< NodeStorageType >::EdgeWeight\(\)](#), [matrixIndex](#), [matrixSize](#), [Markov::Model< NodeStorageType >::Model\(\)](#),

[Markov::Node< storageType >::NodeValue\(\)](#), [ready](#), [Markov::Edge< NodeStorageType >::RightNode\(\)](#), [Markov::Model< NodeStorageType >::TotalEdgeWeights\(\)](#),

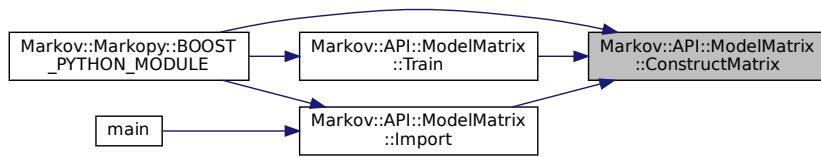
[totalEdgeWeights](#), and [valueMatrix](#).

Referenced by [Markov::Markopy::BOOST\\_PYTHON\\_MODULE\(\)](#), [Import\(\)](#), and [Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 9.17.3.4 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`.

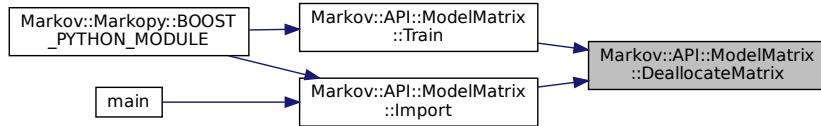
```

00081     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 }
```

References `edgeMatrix`, `matrixIndex`, `matrixSize`, `ready`, `totalEdgeWeights`, and `valueMatrix`.

Referenced by [Import\(\)](#), and [Train\(\)](#).

Here is the caller graph for this function:



### 9.17.3.5 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](#).

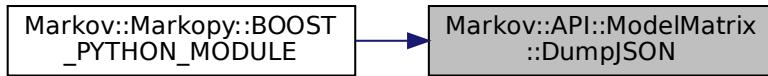
```

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     }
00132     std::cout << "],\\n";
00133 }
00134 std::cout << "},\\n";
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 [edgeMatrix](#), [matrixIndex](#), [matrixSize](#), and [valueMatrix](#).

Referenced by [Markov::Markopy::BOOST\\_PYTHON\\_MODULE\(\)](#).

Here is the caller graph for this function:



### 9.17.3.6 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;` }

### 9.17.3.7 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 300 of file `model.h`.

```
00300
00301     std::ofstream exportfile;
00302     exportfile.open(filename);
00303     return this->Export(&exportfile);
00304 }
```

### 9.17.3.8 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 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         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <
00294         "\n";
00295     }
00296     return true;
00297 }
```

### 9.17.3.9 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 }
```

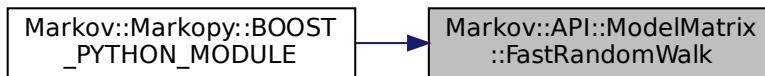
References [FastRandomWalk\(\)](#).

Referenced by [Markov::Markopy::BOOST\\_PYTHON\\_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.17.3.10 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 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);
```

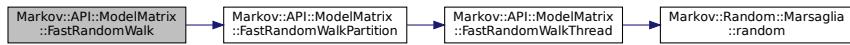
[Definition at line 204 of file modelMatrix.cpp.](#)

```
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 }
```

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.17.3.11 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
00227     int iterationsPerThread = n/threads;
00228     int iterationsPerThreadCarryOver = n%threads;
00229
00230     std::vector<std::thread*> threadsV;
00231 }
```

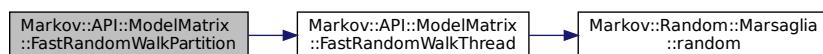
```

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 [FastRandomWalkThread\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.17.3.12 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
<i>id</i>	- <b>DEPRECATED</b> 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

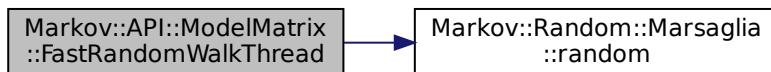
```

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:



### 9.17.3.13 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](#).

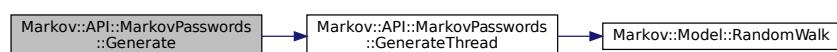
```

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 }
```

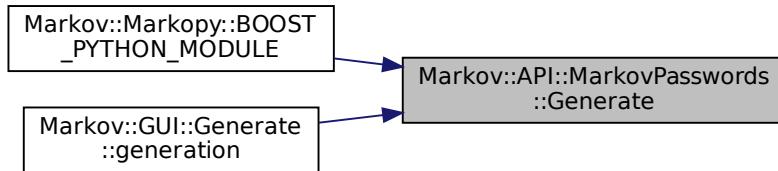
References [Markov::API::MarkovPasswords::GenerateThread\(\)](#).

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:



#### 9.17.3.14 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

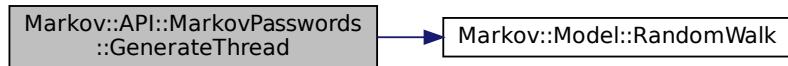
Definition at line 140 of file [markovPasswords.cpp](#).

```
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 }
```

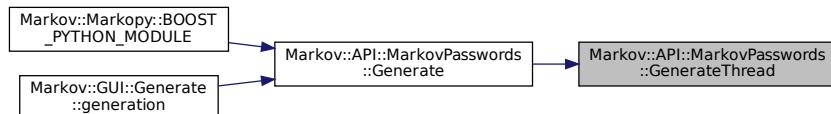
References [Markov::Model< NodeStorageType >::RandomWalk\(\)](#).

Referenced by [Markov::API::MarkovPasswords::Generate\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.17.3.15 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.

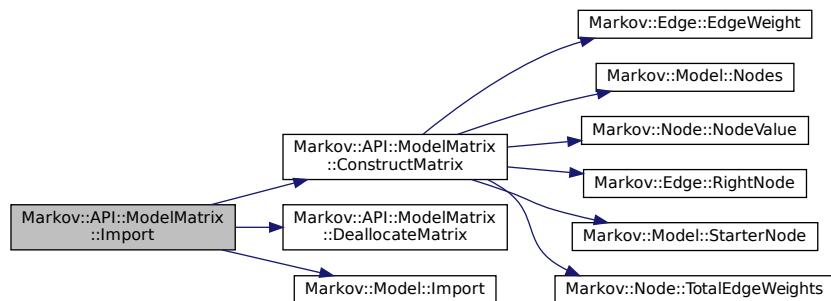
Definition at line 19 of file [modelMatrix.cpp](#).

```
00019
00020     this->DeallocateMatrix();
00021     this->Markov::API::MarkovPasswords::Import(filename);
00022     this->ConstructMatrix();
00023 }
```

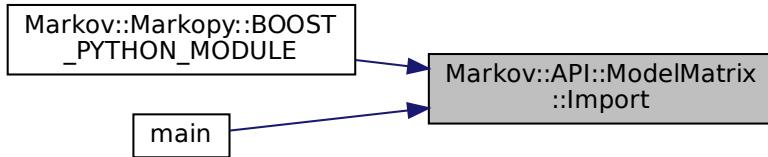
References [ConstructMatrix\(\)](#), [DeallocateMatrix\(\)](#), and [Markov::Model< NodeStorageType >::Import\(\)](#).

Referenced by [Markov::Markopy::BOOST\\_PYTHON\\_MODULE\(\)](#), and [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.17.3.16 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 216 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 }
```

### 9.17.3.17 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; }
```

### 9.17.3.18 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:



### 9.17.3.19 OptimizeEdgeOrder()

`void Markov::Model< char >::OptimizeEdgeOrder [inherited]`

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() << ", ";
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 }
```

### 9.17.3.20 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 <a href="#">Markov::Random::Mersenne</a> and <a href="#">Markov::Random::Marsaglia</a>
<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     }
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 }
```

### 9.17.3.21 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.

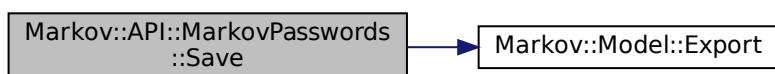
Definition at line 106 of file [markovPasswords.cpp](#).

```

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:



### 9.17.3.22 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; }
```

**9.17.3.23 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.

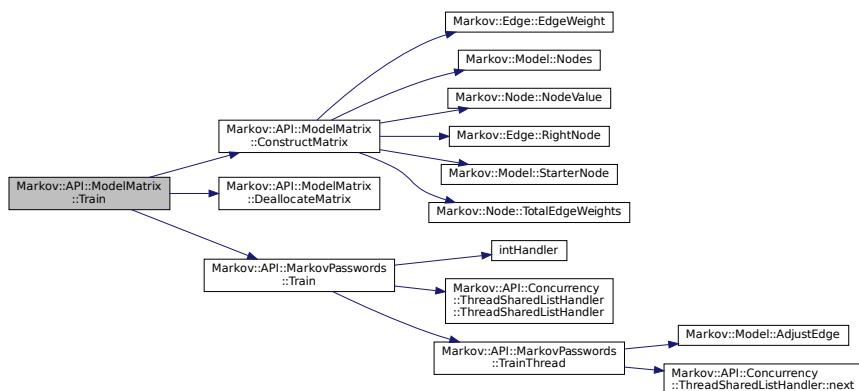
Definition at line 25 of file [modelMatrix.cpp](#).

```
00025
00026     this->DeallocateMatrix();
00027     this->Markov::API::MarkovPasswords::Train(datasetFileName,delimiter,threads);
00028     this->ConstructMatrix();
00029 }
```

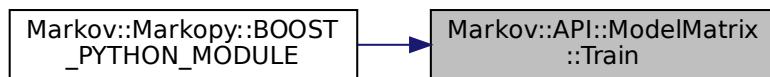
References [ConstructMatrix\(\)](#), [DeallocateMatrix\(\)](#), and [Markov::API::MarkovPasswords::Train\(\)](#).

Referenced by [Markov::Markopy::BOOST\\_PYTHON\\_MODULE\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.17.3.24 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

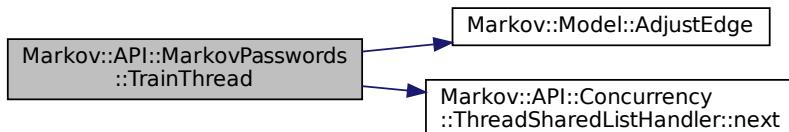
Definition at line 85 of file markovPasswords.cpp.

```
00085
00086     {
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             this->AdjustEdge((const char*)linebuf, oc);
00101             delete linebuf;
00102         }
00103 }
```

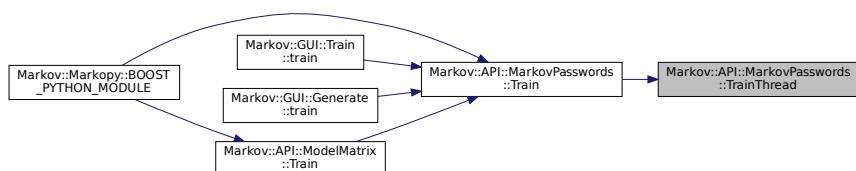
References [Markov::Model< NodeStorageType >::AdjustEdge\(\)](#), [keepRunning](#), and [Markov::API::Concurrency::ThreadSharedListHandler::next\(\)](#).

Referenced by [Markov::API::MarkovPasswords::Train\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 9.17.4 Member Data Documentation

#### 9.17.4.1 datasetFile

`std::ifstream* Markov::API::MarkovPasswords::datasetFile [private], [inherited]`  
 Definition at line 123 of file [markovPasswords.h](#).

#### 9.17.4.2 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\(\)](#).

#### 9.17.4.3 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](#).

#### 9.17.4.4 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\(\)](#).

#### 9.17.4.5 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](#)

#### 9.17.4.6 modelSavefile

`std::ofstream* Markov::API::MarkovPasswords::modelSavefile [private], [inherited]`  
 Dataset file input of our system  
 Definition at line 124 of file [markovPasswords.h](#).

#### 9.17.4.7 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](#).

#### 9.17.4.8 outputFile

`std::ofstream* Markov::API::MarkovPasswords::outputFile [private], [inherited]`  
 File to save model of our system

Definition at line 125 of file [markovPasswords.h](#).

#### 9.17.4.9 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\(\).
```

#### 9.17.4.10 starterNode

```
Node<char>\*> Markov::Model< char >::starterNode \[private\], \[inherited\]
Starter Node of this model.
Definition at line 198 of file model.h.
```

#### 9.17.4.11 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\(\).
```

#### 9.17.4.12 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:
```

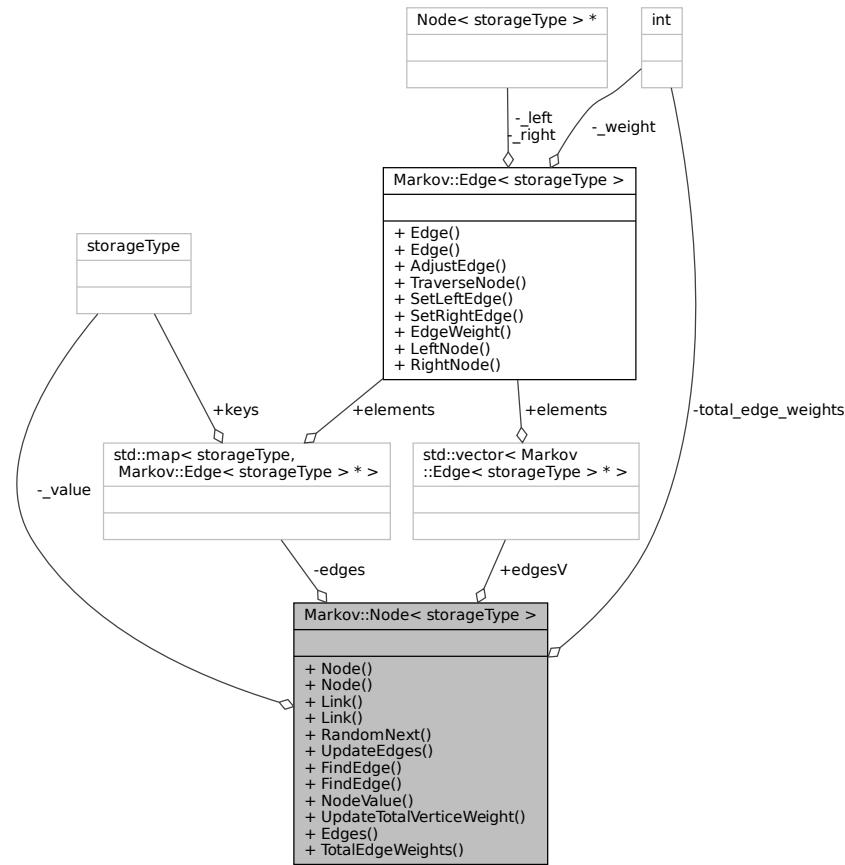
- [modelMatrix.h](#)
- [modelMatrix.cpp](#)

## 9.18 **Markov::Node< storageType >** Class Template Reference

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

```
#include <model.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.*

## 9.18.1 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 27 of file [model.h](#).

## 9.18.2 Constructor & Destructor Documentation

### 9.18.2.1 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 }
```

### 9.18.2.2 Node() [2/2]

```
template<typename storageType >
Markov::Node< storageType >::Node (
    storageType \_value)
Constructor. Creates a Node with no edges and with given NodeValue.
```

#### Parameters

<a href="#">_value</a>	- 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 };

```

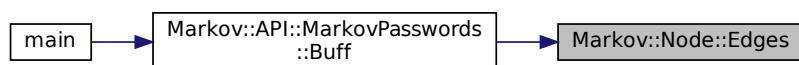
### 9.18.3 Member Function Documentation

#### 9.18.3.1 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:

```



#### 9.18.3.2 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.

#### 9.18.3.3 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 };
```

**9.18.3.4 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**

<a href="#">Edge</a>	- <a href="#">Edge</a> 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 }
```

**9.18.3.5 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**

<a href="#">target</a>	- 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 }

```

### 9.18.3.6 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.

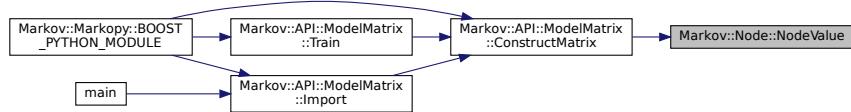
```

00215
00216     return _value;
00217 }

```

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#).

Here is the caller graph for this function:



### 9.18.3.7 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     return NULL;
00249 }
```

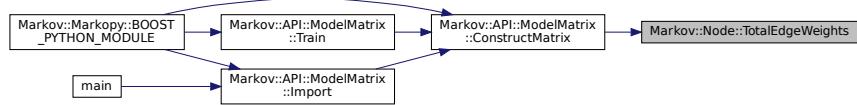
### 9.18.3.8 TotalEdgeWeights()

```

template<typename storageType >
long int Markov::Node< storageType >::TotalEdgeWeights [inline]
return total edge weights
Definition at line 277 of file node.h.
00277
00278     return this->total_edge_weights;
00279 }
```

Referenced by [Markov::API::ModelMatrix::ConstructMatrix\(\)](#).

Here is the caller graph for this function:



### 9.18.3.9 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 }
```

### 9.18.3.10 UpdateTotalVerticeWeight()

```
template<typename storageType >
void Markov::Node< storageType >::UpdateTotalVerticeWeight (
    long int offset )
Change total weights with offset.
```

#### Parameters

<i>offset</i>	to adjust the vertice weight with
---------------	-----------------------------------

#### Definition at line 267 of file node.h.

```
00267
00268     this->total_edge_weights += offset;
00269 }
```

## 9.18.4 Member Data Documentation

### 9.18.4.1 \_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< NodeStorageType >::NodeValue\(\).
```

### 9.18.4.2 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.
```

### 9.18.4.3 edgesV

```
template<typename storageType >
std::vector<Edge<storageType>*> Markov::Node< storageType >::edgesV
Definition at line 173 of file node.h.
```

### 9.18.4.4 total\_edge\_weights

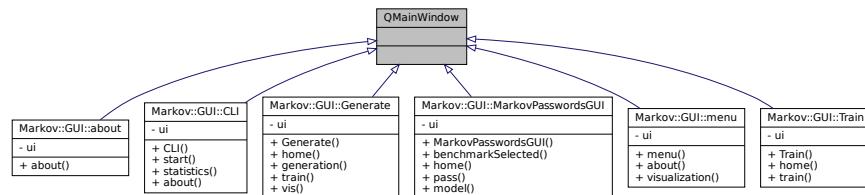
```
template<typename storageType >
long int Markov::Node< storageType >::total_edge_weights [private]
Total weights of the vertices, required by RandomNext.
Definition at line 184 of file node.h.
```

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

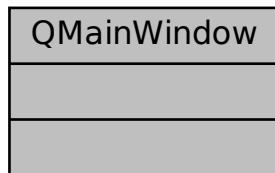
- [model.h](#)
- [node.h](#)

## 9.19 QMainWindow Class Reference

Inheritance diagram for QMainWindow:



Collaboration diagram for QMainWindow:



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

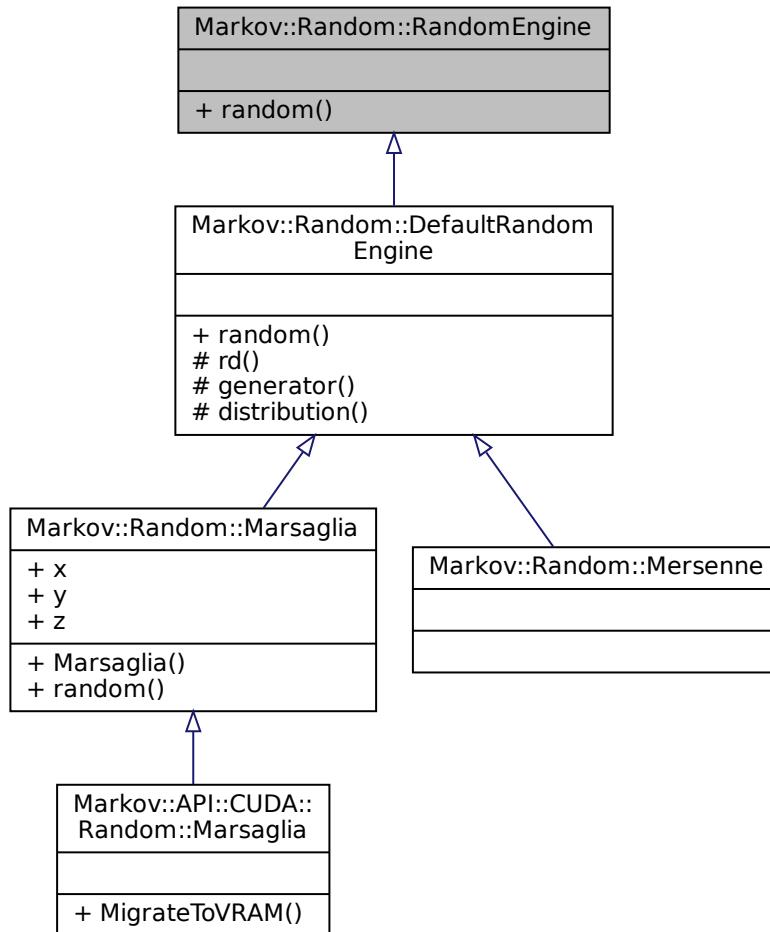
- [menu.h](#)

## 9.20 Markov::Random::RandomEngine Class Reference

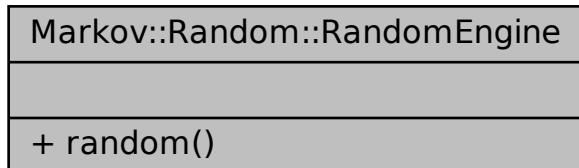
An abstract class for [Random](#) Engine.

```
#include <random.h>
```

Inheritance diagram for `Markov::Random::RandomEngine`:



Collaboration diagram for `Markov::Random::RandomEngine`:



## Public Member Functions

- `virtual unsigned long random ()=0`

### 9.20.1 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](#).

### 9.20.2 Member Function Documentation

#### 9.20.2.1 random()

```
virtual unsigned long Markov::Random::RandomEngine::random () [inline], [pure virtual]
```

Implemented in [Markov::Random::Marsaglia](#), and [Markov::Random::DefaultRandomEngine](#).

Referenced by [Markov::Node< NodeStorageType >::RandomNext\(\)](#).

Here is the caller graph for this function:



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

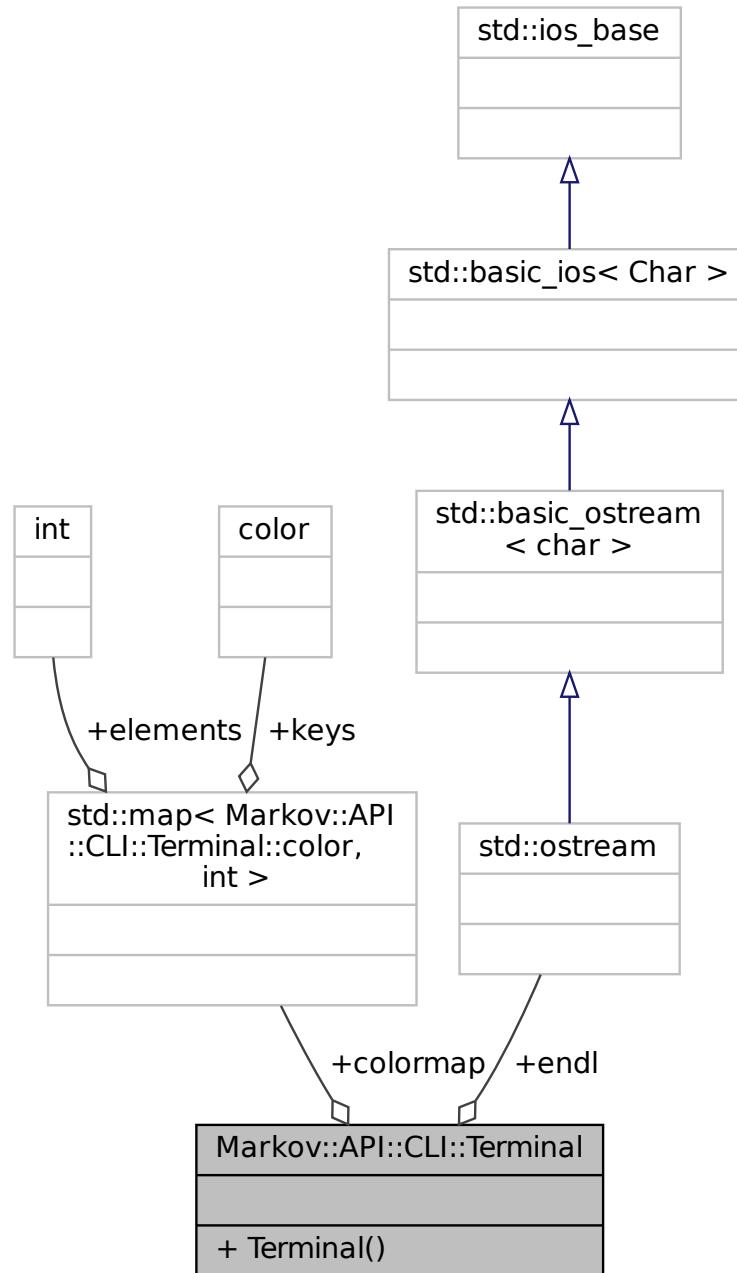
- [random.h](#)

## 9.21 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](#)

### 9.21.1 Detailed Description

pretty colors for [Terminal](#). Windows Only.  
Definition at line [25](#) of file [term.h](#).

### 9.21.2 Member Enumeration Documentation

#### 9.21.2.1 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 };
```

### 9.21.3 Constructor & Destructor Documentation

#### 9.21.3.1 Terminal()

[Terminal::Terminal \(\)](#)

Default constructor. Get references to stdout and stderr handles.

Definition at line [62](#) of file [term.cpp](#).

```
00062 {
00063     /*this->;*/
00064 }
```

### 9.21.4 Member Data Documentation

#### 9.21.4.1 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<<\(\)](#).

#### 9.21.4.2 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:

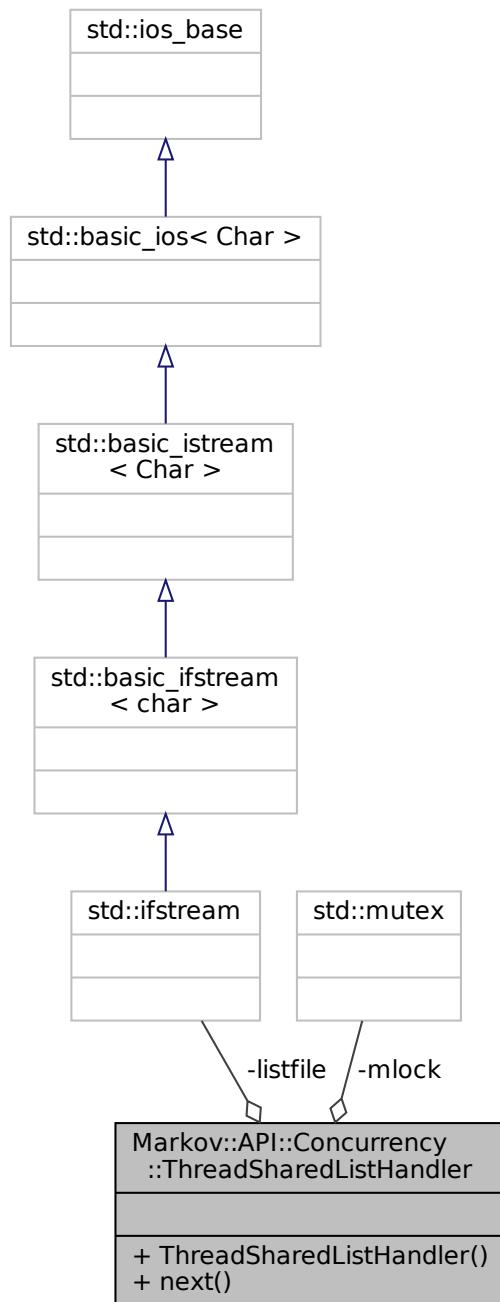
- [term.h](#)
  
  
  
  
  
- [term.cpp](#)

## 9.22 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](#)

### 9.22.1 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](#).

### 9.22.2 Constructor & Destructor Documentation

#### 9.22.2.1 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

<a href="#">filename</a>	Filename for the file to manage.
--------------------------	----------------------------------

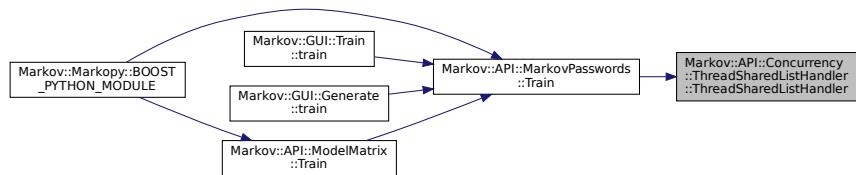
Definition at line 12 of file [threadSharedListHandler.cpp](#).

```
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:



### 9.22.3 Member Function Documentation

#### 9.22.3.1 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:



### 9.22.4 Member Data Documentation

#### 9.22.4.1 listfile

```
std::ifstream Markov::API::Concurrency::ThreadSharedListHandler::listfile [private]
```

Definition at line 95 of file [threadSharedListHandler.h](#).

Referenced by [next\(\)](#), and [ThreadSharedListHandler\(\)](#).

#### 9.22.4.2 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:

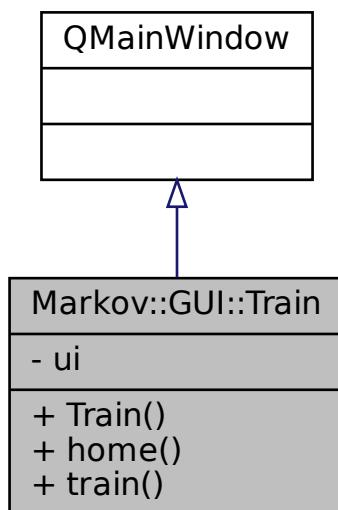
- [threadSharedListHandler.h](#)
- [threadSharedListHandler.cpp](#)

## 9.23 Markov::GUI::Train Class Reference

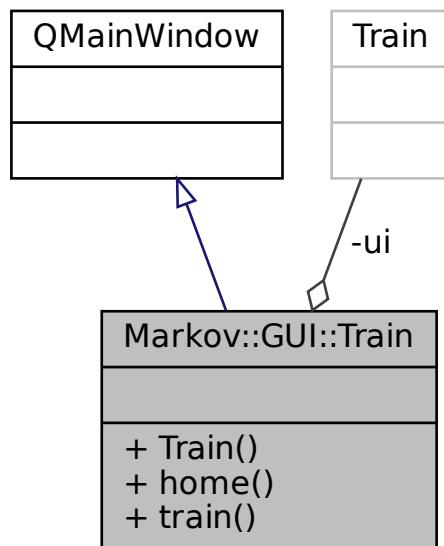
QT Training page class.

```
#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`

### 9.23.1 Detailed Description

QT Training page class.

Definition at line 15 of file `Train.h`.

### 9.23.2 Constructor & Destructor Documentation

#### 9.23.2.1 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](#).

### 9.23.3 Member Function Documentation

#### 9.23.3.1 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 }
```

#### 9.23.3.2 train

```
void Markov::GUI::Train::train ( ) [slot]
```

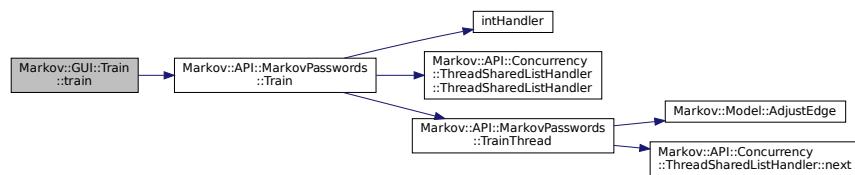
Definition at line 38 of file [Train.cpp](#).

```

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](#).

Here is the call graph for this function:



## 9.23.4 Member Data Documentation

### 9.23.4.1 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:

- [Train.h](#)
- [Train.cpp](#)



# CHAPTER10

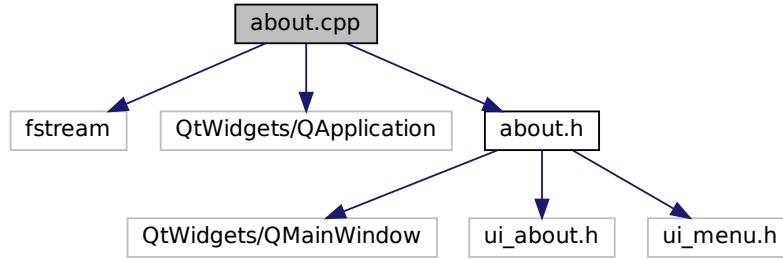
## File Documentation

### 10.1 about.cpp File Reference

About page.

```
#include <fstream>
#include <QtWidgets/QApplication>
#include "about.h"
```

Include dependency graph for about.cpp:



#### 10.1.1 Detailed Description

About page.

Authors

Yunus Emre Yilmaz

Definition in file [about.cpp](#).

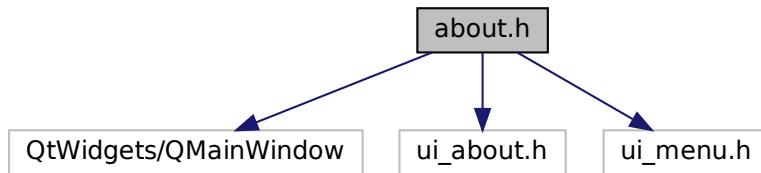
### 10.2 about.cpp

```
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 }
```

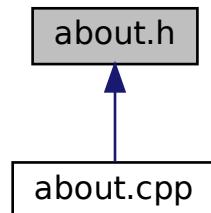
### 10.3 about.h File Reference

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*

### 10.3.1 Detailed Description

About page.

#### Authors

Yunus Emre Yılmaz

Definition in file [about.h](#).

## 10.4 about.h

```

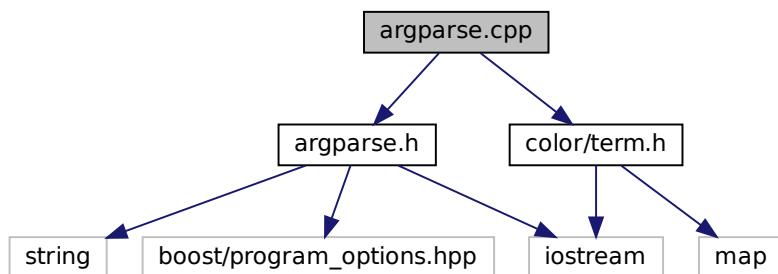
00001 /** @file about.h
00002 * @brief About page
00003 * @authors Yunus Emre Yilmaz
00004 *
00005 */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_about.h"
00010 #include <ui_menu.h>
00011
00012 /** @brief namespace for MarkovPasswords API GUI wrapper
00013 */
00014 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 
```

## 10.5 argparse.cpp File Reference

Arguement handler class for native CPP cli.

```
#include "argparse.h"
#include "color/term.h"
```

Include dependency graph for argparse.cpp:



### 10.5.1 Detailed Description

Arguement handler class for native CPP cli.

#### Authors

Celal Sahir Çetiner

Parse command line arguements.

Definition in file [argparse.cpp](#).

## 10.6 argparse.cpp

```

00001 /** @file argparse.cpp
00002 * @brief Arguement handler class for native CPP cli
  
```

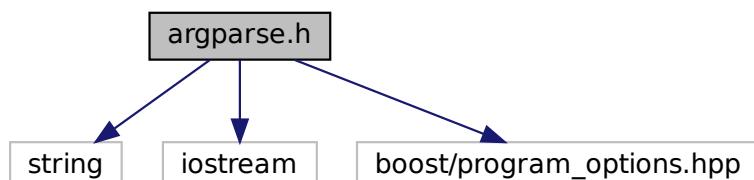
```

00003 * @authors Celal Sahir Çetiner
00004 *
00005 * @copydoc Markov::API::CLI::Argparse
00006 */
00007
00008 #include "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 }
```

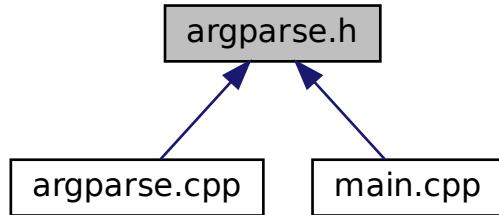
## 10.7 argparse.h File Reference

Argument 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 arguments.*
- class [Markov::API::CLI::Argparse](#)  
*Parse command line arguments.*

## 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 arguments.*

## 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 arguments.*

### 10.7.1 Detailed Description

Argument handler class for native CPP cli.

#### Authors

Celal Sahir Çetiner

Definition in file [argparse.h](#).

### 10.7.2 Macro Definition Documentation

### 10.7.2.1 BOOST\_ALL\_STATIC\_LIB

```
#define BOOST_ALL_STATIC_LIB 1
Definition at line 11 of file argparse.h.
```

### 10.7.2.2 BOOST\_PROGRAM\_OPTIONS\_STATIC\_LIB

```
#define BOOST_PROGRAM_OPTIONS_STATIC_LIB 1
Definition at line 12 of file argparse.h.
```

## 10.8 argparse.h

```
00001 /** @file argparse.h
00002 * @brief Argument 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 arguments.
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 arguments. */
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 Separator character to use with training data. (character between occurrence 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      @brief The name of the given dataset
00069      */
00070      std::string datasetname;
00071
00072      /**
00073          @brief Number of passwords to be generated
00074      */
00075      int generateN;
00076
00077  } ProgramOptions;
00078
00079
00080  /** @brief Parse command line arguments
00081  */
00082  class Argparse {
00083  public:
00084      Argparse();
00085
00086      /** @brief Parse command line arguments.
00087      *
00088      * Parses command line arguments to populate ProgramOptions structure.
00089      *
00090      * @param argc Number of command line arguments
00091      * @param argv Array of command line parameters
00092      */
00093
00094  Argparse(int argc, char** argv) {
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
be ignored for generation mode")
00116          ("separator", 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      if (vm.count("dataset") == 0) this->po.datasetname = "NULL";
00145      else if (vm.count("dataset") == 1) {
00146          this->po.datasetname = vm["dataset"].as<std::string>();
00147      }
00148      else {
00149          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
00195         //else if (vm.count("train")) std::cout << "train oldu" << std::endl;
00196     }
00197
00198     /** @brief Getter for command line options
00199      *
00200      * @return ProgramOptions populated by the argument 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 separator 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.separator = 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 argument count
00240      * @param argv - Program argument values array
00241      * @return ProgramOptions structure.
00242      */
00243     static Markov::API::CLI::ProgramOptions* parse(int argc, char** argv);
00244
00245
00246     /**
00247      * @brief Print help string.
00248      */
00249     static void help();
00250
00251     private:
00252     /**
00253      * @brief ProgramOptions structure object
00254      */
00255     Markov::API::CLI::ProgramOptions po;
00256
00257 };
00258

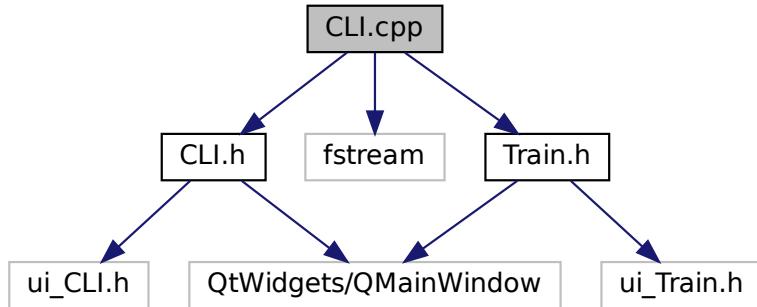
```

## 10.9 CLI.cpp File Reference

CLI page.

```
#include "CLI.h"
#include <fstream>
#include "Train.h"
```

Include dependency graph for CLI.cpp:



### 10.9.1 Detailed Description

CLI page.

Authors

Yunus Emre Yılmaz

Definition in file [CLI.cpp](#).

## 10.10 CLI.cpp

```

00001 /**
00002  * @file cli.cpp
00003  * @brief CLI page
00004  * @authors Yunus Emre Yılmaz
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 }

```

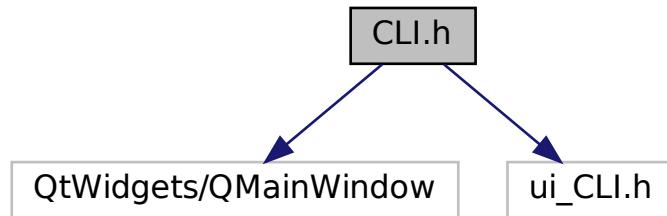
## 10.11 CLI.h File Reference

CLI page.

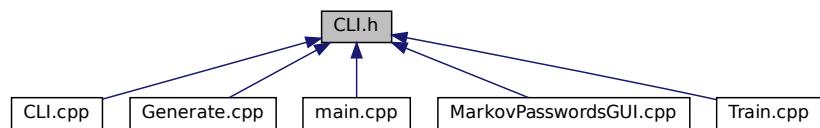
```
#include <QtWidgets/QMainWindow>
```

```
#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*

### 10.11.1 Detailed Description

CLI page.

#### Authors

Yunus Emre Yilmaz

Definition in file [CLI.h](#).

## 10.12 CLI.h

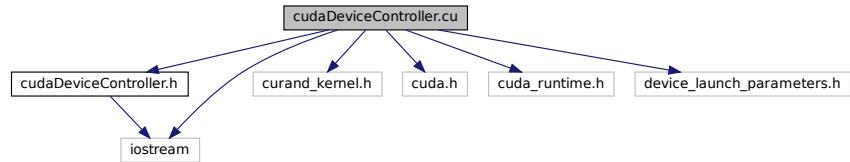
```
00001 /** @file cli.h
00002  * @brief CLI page
00003  * @authors Yunus Emre Yilmaz
00004  *
00005  */
00006
00007 #pragma once
00008 #include <QtWidgets/QMainWindow>
00009 #include "ui_CLI.h"
00010
00011 namespace Markov::GUI{
00012     /** @brief QT CLI Class
00013     */
00014     class CLI :public 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 };
```

## 10.13 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.*

### 10.13.1 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](#).

## 10.14 cudaDeviceController.cu

```

00001 /**
00002 * @file cudaDeviceController.cu
00003 * @brief Simple static class for basic CUDA device controls.
00004 * @authors Ata Hakçıl
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
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     }
00031
00032     __host__ int Markov::API::CUDA::CUDADeviceController::CudaCheckNotifyErr(cudaError_t _status,
00033     const char* msg, bool bExit) {
00034         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 }
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     template <typename T>
00051     __host__ cudaError_t Markov::API::CUDA::CUDADeviceController::CudaMemcpy2DToFlat(T* dst, T** src,
00052     int row, int col){
00053         cudaError_t cudastatus;
00054         for(int i=0;i<row;i++){
00055             cudastatus = cudaMemcpy(dst + (i*col*sizeof(T)),
00056             src[i], col*sizeof(T), cudaMemcpyHostToDevice);
00057             if(cudastatus != cudaSuccess) return cudastatus;
00058         }
00059     }
00060 */
00061
00062 };

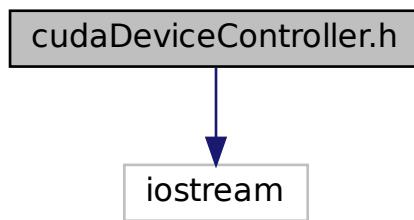
```

## 10.15 cudaDeviceController.h File Reference

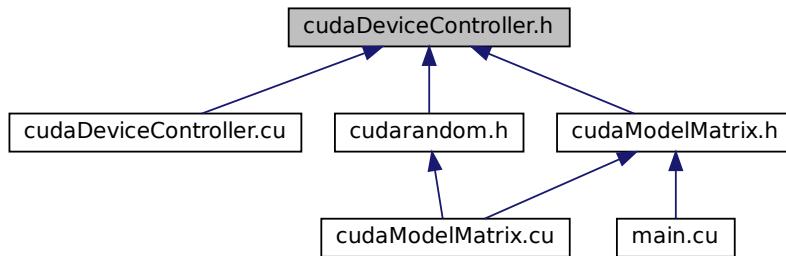
Simple static class for basic CUDA device controls.

```
#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.*

### 10.15.1 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](#).

## 10.16 cudaDeviceController.h

```

00001 /** @file cudaDeviceController.h
00002  * @brief Simple static class for basic CUDA device controls.
00003  * @authors Ata Hakçıl
00004  *
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     *
00025     * @b Example @b output:
00026     * @code{.txt}
00027     * Device Number: 0
00028     * Device name: GeForce RTX 2070
00029     * Memory Clock Rate (KHz): 7001000
00030     * Memory Bus Width (bits): 256
00031     * Peak Memory Bandwidth (GB/s): 448.064
00032     * Max Linear Threads: 1024
00033     * @endcode
00034
00035     */
00036
00037     __host__ static void ListCudaDevices();
00038
00039     protected:
00040         /** @brief Check results of the last operation on GPU.
00041         *
00042         * Check the status returned from cudaMalloc/cudaMemcpy to find failures.
00043         *
00044         * If a failure occurs, its assumed beyond redemption, and exited.
00045         * @param _status Cuda error status to check
00046         * @param msg Message to print in case of a failure
00047         * @return 0 if successful, 1 if failure.
00048         * @b Example @b output:
00049         * @code{.cpp}
00050         * char *da, a = "test";
00051         * cudastatus = cudaMalloc((char **)&da, 5*sizeof(char));
00052         * CudaCheckNotifyErr(cudastatus, "Failed to allocate VRAM for *da.\n");
00053         * @endcode
00054
00055     __host__ static int CudaCheckNotifyErr(cudaError_t _status, const char* msg, bool bExit=true);
00056
00057     /** @brief Malloc a 2D array in device space
00058     *
00059     * This function will allocate enough space on VRAM for flattened 2D array.
00060     *
00061     * @param dst destination pointer
00062     * @param row row size of the 2d array
00063     * @param col column size of the 2d array
00064     * @return cudaError_t status of the cudaMalloc operation
00065
00066     * @b Example @b output:
00067     * @code{.cpp}
00068     *     cudaError_t cudastatus;
00069     *     char* dst;
00070     *     cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
00071     *     if(cudastatus!=cudaSuccess){
00072     *         CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
00073     *     }
00074     *     @endcode
00075
00076     template <typename T>
00077     __host__ static cudaError_t CudaMalloc2DToFlat(T** dst, int row, int col){
00078         cudaError_t cudastatus = cudaMalloc((T **)dst, row*col*sizeof(T));
00079         CudaCheckNotifyErr(cudastatus, "cudaMalloc Failed.", false);
00080         return cudastatus;
00081     }
00082
00083     /** @brief Memcpy a 2D array in device space after flattening
00084     *
00085     * Resulting buffer will not be true 2D array.
00086     *
00087     * @param dst destination pointer
00088     * @param rc source pointer
00089     * @param row row size of the 2d array
00090     * @param col column size of the 2d array
00091     * @return cudaError_t status of the cudaMemcpy operation
00092
00093     * @b Example @b output:
00094     * @code{.cpp}
00095     *     cudaError_t cudastatus;
00096     *     char* dst;
00097     *     cudastatus = CudaMalloc2DToFlat<char>(&dst, 5, 15);
00098     *     CudaCheckNotifyErr(cudastatus, " CudaMalloc2DToFlat Failed.", false);
00099     *     cudastatus = CudaMemcpy2DToFlat<char>(*dst,src,15,15);
00100     *     CudaCheckNotifyErr(cudastatus, " CudaMemcpy2DToFlat Failed.", false);
00101     *     @endcode
00102
00103     template <typename T>
00104     __host__ static cudaError_t CudaMemcpy2DToFlat(T* dst, T** src, int row, int col){
00105         T* tempbuf = new T[row*col];
00106         for(int i=0;i<row;i++){
00107             memcpy(&(tempbuf[row*i]), src[i], col);
00108         }
00109         return cudaMemcpy(dst, tempbuf, row*col*sizeof(T), cudaMemcpyHostToDevice);
00110

```

```

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
00145   private:
00146 };
00147 };

```

## 10.17 cudaModelMatrix.cu File Reference

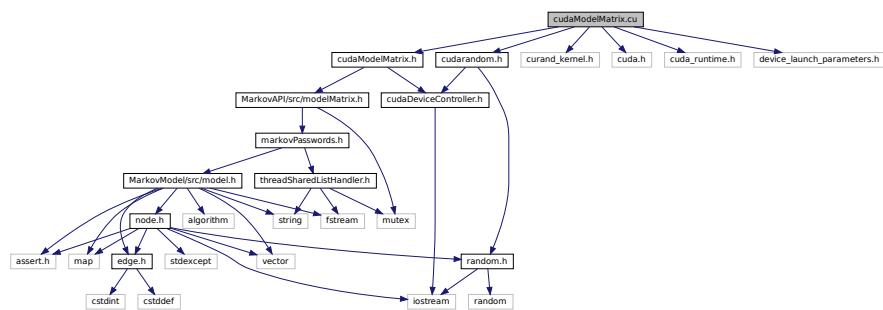
CUDA accelerated extension of [Markov::API::ModelMatrix](#).

```

#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*

### 10.17.1 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](#).

## 10.18 cudaModelMatrix.cu

```

00001 /**
00002  * @brief CUDA accelerated extension of Markov::API::ModelMatrix
00003  * @authors Ata Hakçıl
00004  *
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         cudaError_t 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.");

```

```

00041     cudastatus = CudaMigrate2DFlat<long int>(
00042         &(this->device_valueMatrix), this->valueMatrix, this->matrixSize, this->matrixSize);
00043     CudaCheckNotifyErr(cudastatus, "    Cuda failed to initialize value matrix row.");
00044 }
00045
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 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->cuadastreams[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
00134    over to CPU generation.\n";
00135    this->iterationsPerKernelThread = leftover/cudaGridSize;
00136
00137    leftover -= this->iterationsPerKernelThread;
00138
00139    if(!leftover) return;
00140    std::cerr << "Remaining " << leftover << " lines are absolutely not worth printing.\n";
00141    Markov::API::ModelMatrix::ConstructMatrix();
00142    Markov::API::ModelMatrix::FastRandomWalk(leftover, &wordlist, minLen, maxLen, 1, bFileIO);
00143
00144 }
00145
00146 __host__ void Markov::API::CUDA::CUDAModelMatrix::prepKernelMemoryChannel(int numberOfStreams) {
00147
00148    this->cuadastreams = new cudaStream_t[numberOfStreams];
00149    for(int i=0;i<numberOfStreams;i++)
00150        cudaStreamCreate(&this->cuadastreams[i]);
00151
00152    this->outputBuffer = new char*[numberOfStreams];
00153    for(int i=0;i<numberOfStreams;i++)
00154        this->outputBuffer[i] = new char[cudaPerKernelAllocationSize];
00155
00156    cudaError_t cudastatus;
00157    this->device_outputBuffer = new char*[numberOfStreams];
00158    for(int i=0;i<numberOfStreams;i++){
00159        cudastatus = cudaMalloc((char**)&(device_outputBuffer[i]),
00160                               cudaPerKernelAllocationSize);
00161        CudaCheckNotifyErr(cudastatus, "Failed to establish memory channel. Possibly out of
00162        VRAM?");
00163
00164        this->device_seeds = new unsigned long*[numberOfStreams];
00165        for(int i=0;i<numberOfStreams;i++){
00166            Markov::API::CUDA::Random::Marsaglia *MEarr = new
00167            Markov::API::CUDA::Random::Marsaglia[cudaGridSize];
00168            this->device_seeds[i] = Markov::API::CUDA::Random::Marsaglia::MigrateToVRAM(MEarr,
00169            cudaGridSize);
00170            delete[] MEarr;
00171        }
00172    }
00173
00174    __host__ void Markov::API::CUDA::CUDAModelMatrix::LaunchAsyncKernel(int kernelID, int minLen, int
00175    maxLen){
00176
00177        //if(kernelID == 0); // cudaStreamSynchronize(this->cuadastreams[2]);
00178        //else cudaStreamSynchronize(this->cuadastreams[kernelID-1]);
00179        FastRandomWalkCUDAKernel<<(cudaBlocks,cudaThreads,0,
00180        this->cuadastreams[kernelID]>>(iterationsPerKernelThread, minLen, maxLen,
00181        this->device_outputBuffer[kernelID], this->device_matrixIndex,
00182        this->device_totalEdgeWeights, this->device_valueMatrix, this->device_edgeMatrix,
00183        this->matrixSize, cudaMemPerGrid, this->device_seeds[kernelID]);
00184        //std::cerr << "Started kernel" << kernelID << "\n";
00185    }
00186
00187    __host__ void Markov::API::CUDA::CUDAModelMatrix::GatherAsyncKernelOutput(int kernelID, bool
00188    bFileIO, std::ofstream &wordlist){
00189
00190        cudaMemcpy(this->outputBuffer[kernelID],this->device_outputBuffer[kernelID],cudaPerKernelAllocationSize,
00191        cudaMemcpyDeviceToHost);
00192        //std::cerr << "Kernel" << kernelID << " output copied\n";
00193        if(bFileIO){
00194            for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00195                wordlist << &this->outputBuffer[kernelID][j];
00196            }
00197        }else{
00198            for(long int j=0;j<cudaPerKernelAllocationSize;j+=cudaMemPerGrid){
00199                std::cout << &this->outputBuffer[kernelID][j];
00200            }
00201        }
00202    }
00203
00204    __global__ void FastRandomWalkCUDAKernel(unsigned long int n, int minLen, int maxLen, char*
00205    outputBuffer,
00206    char* matrixIndex, long int* totalEdgeWeights, long int* valueMatrix, char *edgeMatrix, int
00207    matrixSize, int memoryPerKernelGrid, unsigned long *seed);
00208

```

```

00197     int kernelWorkerIndex = threadIdx.x + blockIdx.x * blockDim.x;
00198
00199     if(n==0) return;
00200
00201     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
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 }
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::CUDA::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 };

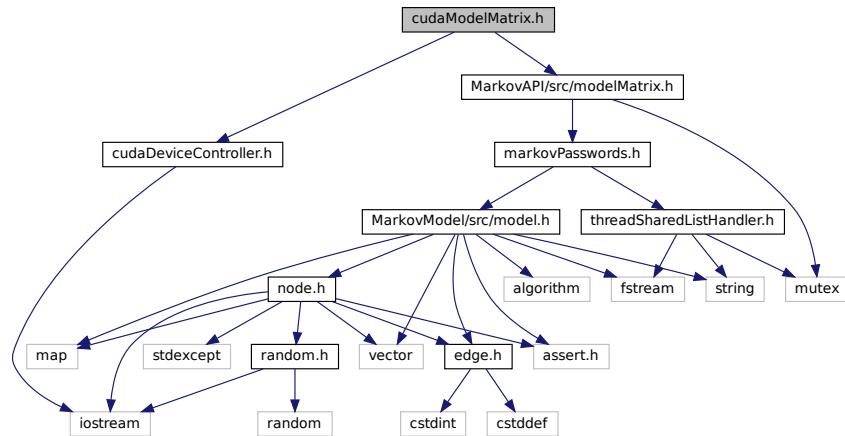
```

## 10.19 cudaModelMatrix.h File Reference

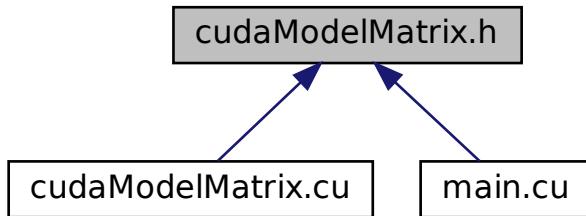
CUDA accelerated extension of [Markov::API::ModelMatrix](#).

```
#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*

### 10.19.1 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](#).

## 10.20 cudaModelMatrix.h

```

00001 /**
00002  * @file cudaModelMatrix.h
00003  * @brief CUDA accelerated extension of Markov::API::ModelMatrix
00004  * @authors Ata Hakçıl
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     /** @brief Migrate the class members to the VRAM
00022      *
00023      * Cannot be used without calling Markov::API::ModelMatrix::ConstructMatrix at least once.
00024      * This function will manage the memory allocation and data transfer from CPU RAM to GPU VRAM.
00025      *
00026      * Newly allocated VRAM pointers are set in the class member variables.
00027      *
00028 */
00029     __host__ void MigrateMatrix();
00030
00031     /** @brief Flatten migrated matrix from 2d to 1d
00032      *
00033      */
00034     __host__ void FlattenMatrix();
00035
00036     /** @brief Random walk on the Matrix-reduced Markov::Model
00037      *
00038      * TODO
00039      *
00040      *
00041      *
00042      * @param n - Number of passwords to generate.
00043      * @param wordlistFileName - Filename to write to
00044      * @param minLen - Minimum password length to generate
00045      * @param maxLen - Maximum password length to generate
00046      * @param threads - number of OS threads to spawn
00047      * @param bFileIO - If false, filename will be ignored and will output to stdout.
00048      *
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
00061     protected:
00062
00063     /** @brief Allocate the output buffer for kernel operation
00064     *
00065     * TODO
00066     *
00067     * @param n - Number of passwords to generate.
00068     * @param singleGenMaxLen - maximum string length for a single generation
00069     * @param CUDAKernelGridSize - Total number of grid members in CUDA kernel
00070     * @param sizePerGrid - Size to allocate per grid member
00071     * @return pointer to the allocation on VRAM
00072     *
00073     */
00074
00075     __host__ char* AllocVRAMOutputBuffer(long int n, long int singleGenMaxLen, long int
00076     CUDAKernelGridSize, long int sizePerGrid);
00077
00078     __host__ void LaunchAsyncKernel(int kernelID, int minLen, int maxLen);
00079
00080     __host__ void prepKernelMemoryChannel(int numberOfWorkstreams);
00081
00082     __host__ void GatherAsyncKernelOutput(int kernelID, bool bFileIO, std::ofstream &wordlist);
00083
00084     private:
00085
00086     /**
00087     * @brief VRAM Address pointer of edge matrix (from modelMatrix.h)
00088     */
00089     char* device_edgeMatrix;
00090
00091     /**
00092     * @brief VRAM Address pointer of value matrix (from modelMatrix.h)
00093     */
00094     long int *device_valueMatrix;
00095
00096     /**
00097     * @brief VRAM Address pointer of matrixIndex (from modelMatrix.h)
00098     */
00099     char *device_matrixIndex;
00100
00101     /**
00102     * @brief VRAM Address pointer of total edge weights (from modelMatrix.h)
00103     */
00104     long int *device_totalEdgeWeights;
00105
00106     /**
00107     * @brief RandomWalk results in device
00108     */
00109     char** device_outputBuffer;
00110
00111     /**
00112     * @brief RandomWalk results in host
00113     */
00114     char** outputBuffer;
00115
00116     /**
00117     * @brief Adding Edge matrix end-to-end and resize to 1-D array for better performance on
00118     * traversing
00119     */
00120     char* flatEdgeMatrix;
00121
00122     /**
00123     * @brief Adding Value matrix end-to-end and resize to 1-D array for better performance on
00124     * traversing
00125     */
00126     long int* flatValueMatrix;
00127
00128     int cudaBlocks;
00129     int cudaThreads;
00130     int iterationsPerKernelThread;
00131     long int totalOutputPerSync;
00132     long int totalOutputPerKernel;
00133     int numberOfWorkPartitions;
00134     int cudaGridSize;
00135     int cudaMemPerGrid;
00136     long int cudaPerKernelAllocationSize;
00137     int alternatingKernels;
00138
00139     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 * @brief srtchr implementation on __device__ space
00171 *
00172 * Find the first matching index of a string
00173 *
00174 * @param p - string to check
00175 * @param c - character to match
00176 * @param s_len - maximum string length
00177 * @returns pointer to the match
00178 */
00179 __device__ char* strchr(char* p, char c, int s_len);
00180 };

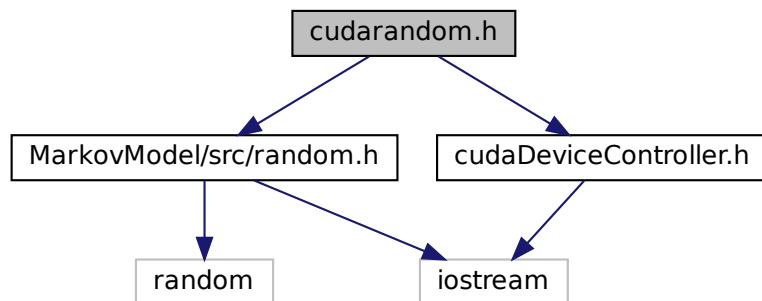
```

## 10.21 cudarandom.h File Reference

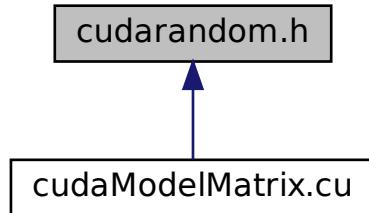
Extension of [Markov::Random::Marsaglia](#) for CUDA.

```
#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.*

### 10.21.1 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](#).

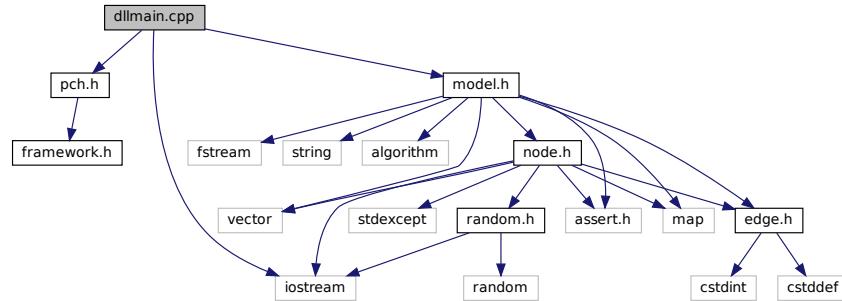
## 10.22 cudarandom.h

```
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 of 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 };
```

## 10.23 dllmain.cpp File Reference

DLLMain for dynamic windows library.

```
#include "pch.h"
#include "model.h"
#include <iostream>
Include dependency graph for dllmain.cpp:
```



### 10.23.1 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](#).

## 10.24 dllmain.cpp

```

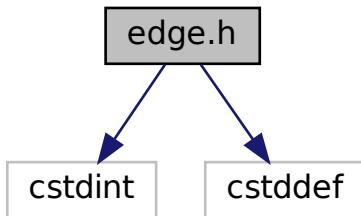
00001 /**
00002 * @file dllmain.cpp
00003 * @brief DLLMain for dynamic windows library
00004 * @authors Ata Hakçıl
00005 * @copydoc Markov::Model
00006 */
00007
00008 #include "pch.h"
00009 #include "model.h"
00010 #include <iostream>
00011
00012
00013 #ifdef _WIN32
00014 __declspec(dllexport) void dll_loadtest() {
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

```

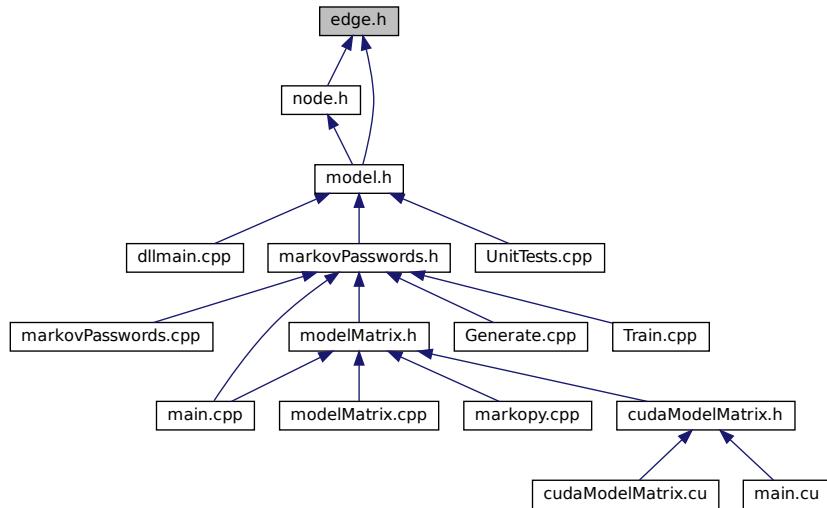
## 10.25 edge.h File Reference

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

### 10.25.1 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](#).

## 10.26 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>*> _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      * e1->AdjustEdge(25);
00071      * Markov::Edge<unsigned char>* e2 = e1->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      * 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 template <typename NodeStorageType>
00138 void Markov::Edge<NodeStorageType>::AdjustEdge(long int offset) {
00139     this->_weight += offset;
00140     this->LeftNode()->UpdateTotalVerticeWeight(offset);
00141 }
00142 //to TraverseNode the node
00143 template <typename NodeStorageType>
00144 inline Markov::Node<NodeStorageType>* Markov::Edge<NodeStorageType>::TraverseNode() {
00145     if (this->RightNode()->NodeValue() == 0xff) //terminator node
00146         return NULL;
00147     return _right;
00148 }
00149 //to set the LeftNode of the node
00150 template <typename NodeStorageType>
00151 void Markov::Edge<NodeStorageType>::SetLeftEdge(Markov::Node<NodeStorageType>* n) {
00152     this->_left = n;
00153 }
00154 //to set the RightNode of the node

```

```

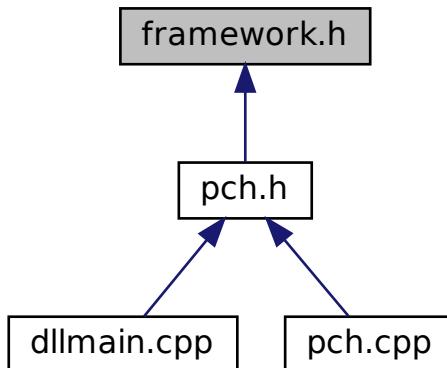
00155 void Markov::Edge<NodeStorageType>::SetRightEdge(Markov::Node<NodeStorageType>* n) {
00156     this->_right = n;
00157 }
00158 //to get the EdgeWeight of the node
00159 template <typename NodeStorageType>
00160 inline uint64_t Markov::Edge<NodeStorageType>::EdgeWeight() {
00161     return this->_weight;
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 }

```

## 10.27 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

#### 10.27.1 Detailed Description

for windows dynamic library

##### Authors

Ata Hakçıl

Definition in file [framework.h](#).

#### 10.27.2 Macro Definition Documentation

### 10.27.2.1 WIN32\_LEAN\_AND\_MEAN

```
#define WIN32_LEAN_AND_MEAN
Definition at line 9 of file framework.h.
```

## 10.28 framework.h

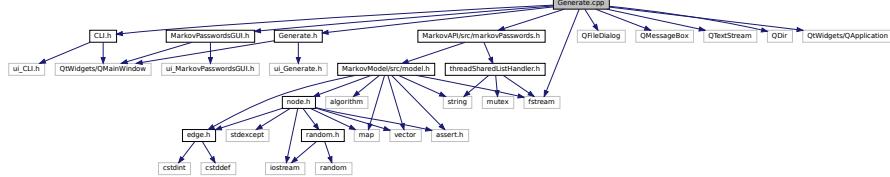
```
00001 /** @file framework.h
00002 * @brief for windows dynamic library
00003 * @authors Ata Hakçıl
00004 *
00005 */
00006
00007 #pragma once
00008
00009 #define WIN32_LEAN_AND_MEAN // Exclude rarely-used stuff from Windows headers
00010 // Windows Header Files
00011
00012 #ifdef _WIN32
00013 #include <windows.h>
00014 #endif
```

## 10.29 Generate.cpp File Reference

Generation Page.

```
#include "Generate.h"
#include <fstream>
#include <QFileDialog>
#include <QMMessageBox>
#include <QTextStream>
#include <QDir>
#include "CLI.h"
#include "MarkovAPI/src/markovPasswords.h"
#include <QtWidgets/QApplication>
#include "MarkovPasswordsGUI.h"
```

Include dependency graph for Generate.cpp:



### 10.29.1 Detailed Description

Generation Page.

Authors

Yunus Emre Yılmaz

Definition in file [Generate.cpp](#).

## 10.30 Generate.cpp

```
00001 /** @file Generate.cpp
00002 * @brief Generation Page
00003 * @authors Yunus Emre Yılmaz
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
00031     ui.pushButton->setVisible(false);
00032     ui.lineEdit->setVisible(false);
00033     ui.lineEdit_2->setVisible(false);
00034     ui.lineEdit_3->setVisible(false);
00035     ui.label_3->setVisible(false);
00036     ui.label_4->setVisible(false);
00037     ui.label_5->setVisible(false);
00038
00039 }
00040
00041 void Generate::generation() {
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 }
00080
00081
00082 void Generate::train() {
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;
```

```

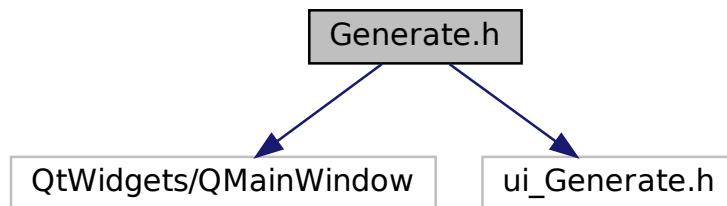
00094     std::string fname = file_name.toStdString();
00095     cstr = new char[fname.size() + 1];
00096     strcpy(cstr, fname.c_str());
00097
00098
00099
00100    char a = ',';
00101    Markov::API::MarkovPasswords mp;
00102    mp.Import("models\\2gram.mdl");
00103    mp.Train(cstr, a, 10);
00104    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 }
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 }
```

## 10.31 Generate.h File Reference

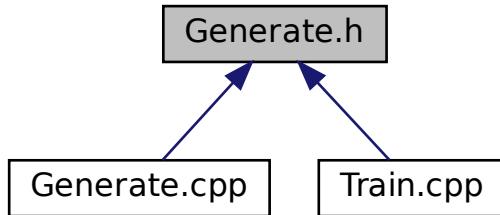
Generation Page.

```
#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*

### 10.31.1 Detailed Description

Generation Page.

#### Authors

Yunus Emre Yilmaz

Definition in file [Generate.h](#).

## 10.32 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"
0010
0011
0012 namespace Markov::GUI{
0013     /** @brief QT Generation page class
0014     */
0015     class Generate :public QMainWindow {
0016         Q_OBJECT
0017     public:
0018         Generate(QWidget* parent = Q_NULLPTR);
0019
0020     private:
0021         Ui::Generate ui;
0022
0023     public slots:
0024         void home();
0025         void generation();
  
```

```

00026     void train();
00027     void vis();
00028 };
00029 };

```

## 10.33 main.cpp File Reference

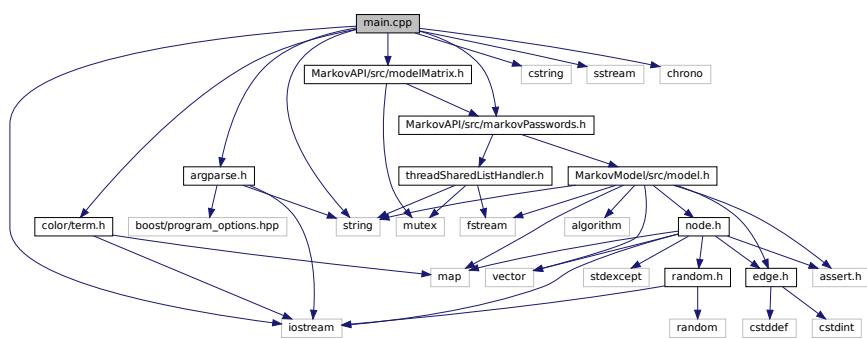
Test cases for [Markov::API::ModelMatrix](#).

```

#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 [MarkovAPICLI/src/main.cpp](#):



## Functions

- int [main](#) (int argc, char \*\*argv)
- Launch CLI tool.*

### 10.33.1 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 [MarkovAPICLI/src/main.cpp](#).

### 10.33.2 Function Documentation

### 10.33.2.1 main()

```

int main (
    int argc,
    char ** argv )

```

Launch CLI tool.

Definition at line 23 of file [MarkovAPICLI/src/main.cpp](#).

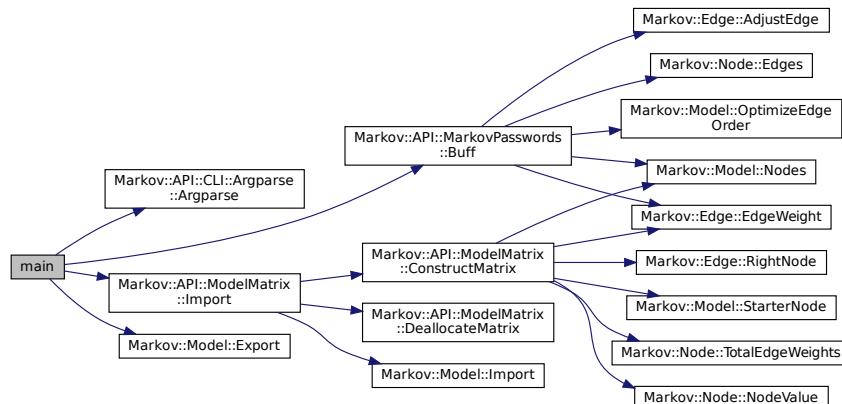
```

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(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     return 0;
00051 }

```

References [Markov::API::CLI::Argparse::Argparse\(\)](#), [Markov::API::MarkovPasswords::Buff\(\)](#), [Markov::Model< NodeStorageType >::Buff\(\)](#) and [Markov::API::ModelMatrix::Import\(\)](#).

Here is the call graph for this function:



## 10.34 MarkovAPICLI/src/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>

```

```

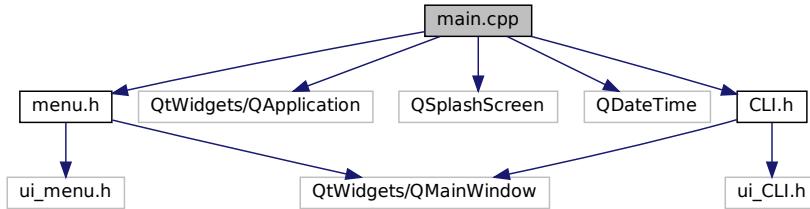
00012 #include "color/term.h"
00013 #include "argparse.h"
00014 #include <string>
00015 #include <cstring>
00016 #include <sstream>
00017 #include "MarkovAPI/src/markovPasswords.h"
00018 #include "MarkovAPI/src/modelMatrix.h"
00019 #include <chrono>
00020
00021 /** @brief Launch CLI tool.
00022 */
00023 int main(int argc, char** argv) {
00024     Markov::API::CLI::Terminal t;
00025     /*
00026     ProgramOptions* p = Argparse::parse(argc, argv);
00027
00028     if (p==0 || p->bFailure) {
00029         std::cout << TERM_FAIL << "Arguments Failed to Parse" << std::endl;
00030         Argparse::help();
00031     }
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;
00050     return 0;
00051 }
```

## 10.35 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 MarkovPasswordsGUI/src/main.cpp:



## Functions

- int `main` (int argc, char \*argv[ ])

*Launch UI.*

### 10.35.1 Detailed Description

Entry point for GUI.

Authors

Yunus Emre Yilmaz

Definition in file [MarkovPasswordsGUI/src/main.cpp](#).

### 10.35.2 Function Documentation

#### 10.35.2.1 main()

```
int main (
    int argc,
    char * argv[] )
```

Launch UI.

Definition at line 18 of file [MarkovPasswordsGUI/src/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 }
```

## 10.36 MarkovPasswordsGUI/src/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 }

```

## 10.37 main.cu File Reference

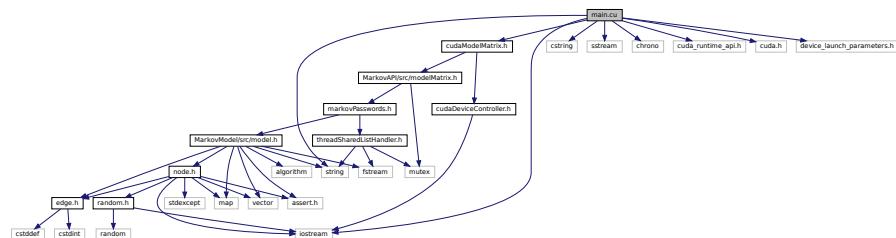
Simple test file to check libcudamarkov.

```

#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)

### 10.37.1 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](#).

### 10.37.2 Function Documentation

#### 10.37.2.1 main()

```

int main (
    int argc,
    char ** argv )

```

Definition at line 21 of file [main.cu](#).

```

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 }
```

## 10.38 main.cu

```

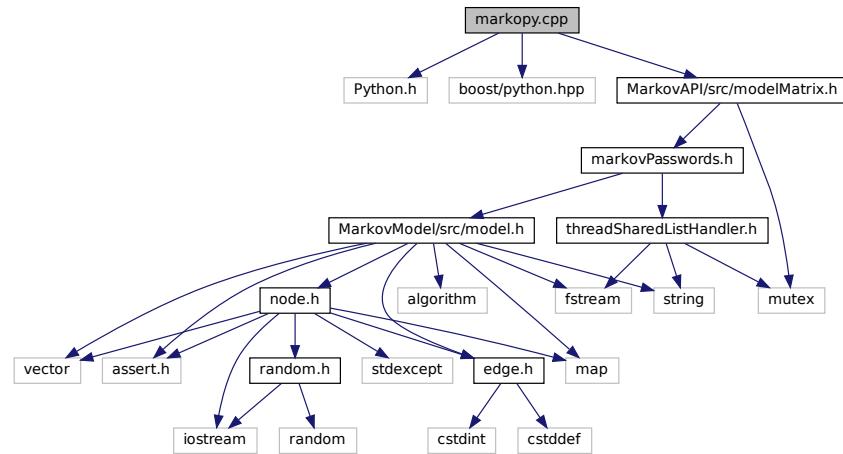
00001 /**
00002 * @file main.cu
00003 * @brief Simple test file to check libcudamarkov
00004 * @authors Ata Hakçıl
00005 * @copydoc Markov::API::CUDA::CUDAModelMatrix
00006 * @copydoc Markov::API::CUDA::CUDADeviceController
00007 */
00008
00009 #include <iostream>
00010 #include <string>
00011 #include <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 }
```

## 10.39 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](#)

## Macros

- `#define BOOST_ALL_STATIC_LIB 1`
- `#define BOOST_PYTHON_STATIC_LIB 1`

## Functions

- [Markov::Markopy::BOOST\\_PYTHON\\_MODULE](#) (markopy)

### 10.39.1 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](#).

## 10.39.2 Macro Definition Documentation

### 10.39.2.1 BOOST\_ALL\_STATIC\_LIB

```
#define BOOST_ALL_STATIC_LIB 1
```

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

### 10.39.2.2 BOOST\_PYTHON\_STATIC\_LIB

```
#define BOOST_PYTHON_STATIC_LIB 1
```

Definition at line [25](#) of file [markopy.cpp](#).

## 10.40 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 namespace Markov::Markopy{
00034     BOOST_PYTHON_MODULE(markopy)
00035     {
00036         bool (Markov::API::MarkovPasswords::*Import)(const char*) = &Markov::Model<char>::Import;
00037         bool (Markov::API::MarkovPasswords::*Export)(const char*) = &Markov::Model<char>::Export;
00038         class_<Markov::API::MarkovPasswords>("MarkovPasswords", init<>())
00039             .def(init<>())
00040             .def("Train", &Markov::API::MarkovPasswords::Train)
00041             .def("Generate", &Markov::API::MarkovPasswords::Generate)
00042             .def("Import", Import, "Import a model file.")
00043     }
00044 }
```

```

00043     .def("Export", Export, "Export a model to file.")
00044     ;
00045
00046     int (Markov::API::ModelMatrix::*FastRandomWalk)(unsigned long int, const char*, int, int, int,
00047     bool) = &Markov::API::ModelMatrix::FastRandomWalk;
00048     class<Markov::API::ModelMatrix>("ModelMatrix", init<>())
00049
00050     .def(init<>())
00051     .def("Train", &Markov::API::ModelMatrix::Train)
00052     .def("Import", &Markov::API::ModelMatrix::Import, "Import a model file.")
00053     .def("Export", Export, "Export a model to file.")
00054     .def("ConstructMatrix", &Markov::API::ModelMatrix::ConstructMatrix)
00055     .def("DumpJSON", &Markov::API::ModelMatrix::DumpJSON)
00056     .def("FastRandomWalk", FastRandomWalk)
00057     ;
00058 };
00059 };

```

## 10.41 markopy.py File Reference

### Namespaces

- [markopy](#)

### Functions

- def [markopy.cli\\_init](#) (str input\_model)
- def [markopy.cli\\_train](#) (markopy.ModelMatrix model, str dataset, str seperator, str output, bool output\_← forced=False, bool bulk=False)
- def [markopy.cli\\_generate](#) (markopy.ModelMatrix model, str wordlist, bool bulk=False)

### Variables

- [markopy.parser](#)
- [markopy.help](#)
- [markopy.default](#)
- [markopy.action](#)
- [markopy.args](#) = parser.parse\_args()

## 10.42 markopy.py

```

00001 """ @package markopy
00002 @file markopy.py
00003 @namespace Python:::Markopy
00004 @brief Command line
00005 @authors Ata Hakçıl
00006 """
00007
00008 import markopy
00009 import argparse
00010 import allegate as logging
00011 import re
00012 import os
00013
00014 parser = argparse.ArgumentParser(description="Python wrapper for MarkovPasswords.",
00015 epilog=f"""Sample runs:
00016 {__file__} train untrained.mdl -d dataset.dat -s "\t" -o trained.mdl
00017     Import untrained.mdl, train it with dataset.dat which has tab delimited data, output resulting
model to trained.mdl\n
00018
00019 {__file__} generate trained.mdl -n 500 -w output.txt
00020     Import trained.mdl, and generate 500 lines to output.txt
00021
00022 {__file__} combine untrained.mdl -d dataset.dat -s "\t" -n 500 -w output.txt
00023     Train and immediately generate 500 lines to output.txt. Do not export trained model.
00024
00025 {__file__} combine untrained.mdl -d dataset.dat -s "\t" -n 500 -w output.txt -o trained.mdl
00026     Train and immediately generate 500 lines to output.txt. Export trained model.
00027 """, formatter_class=argparse.RawTextHelpFormatter)
00028 parser.add_argument("mode", help="Operation mode, supported modes:
    \\"generate\\", \\"train\\" and \\"combine\\\"")

```

```

00029 parser.add_argument("input",
    imported before starting operation.")
00030 parser.add_argument("-o", "--output",
    exported when done. Will be ignored for generation mode.")
00031 parser.add_argument("-d", "--dataset",
    training. Will be ignored for generation mode.")
00032 parser.add_argument("-s", "--separator",
    data.(character between occurrence and value)")
00033 parser.add_argument("-w", "--wordlist",
    results to. Will be ignored for training mode")
00034 parser.add_argument("--min", default=6,
    generation")
00035 parser.add_argument("--max", default=12,
    generation")
00036 parser.add_argument("-n", "--count",
    training mode")
00037 parser.add_argument("-t", "--threads", default=10,
    training mode")
00038 parser.add_argument("-v", "--verbosity", action="count", help="Output verbosity.")
00039 parser.add_argument("-b", "--bulk", action="store_true", help="Bulk generate or bulk train every
    corpus/model in the folder.")
00040 args = parser.parse_args()
00041
00042
00043
00044
00045 def cli_init(input_model : str):
00046     """ Initialize the model for CLI interation
00047     @param input_model Model filename
00048     """
00049     logging.VERBOSITY = 0
00050     if args.verbosity:
00051         logging.VERBOSITY = args.verbosity
00052         logging pprint(f"Verbosity set to {args.verbosity}.", 2)
00053
00054     logging pprint("Initializing model.", 1)
00055     model = markopy.ModelMatrix()
00056     logging pprint("Model initialized.", 2)
00057
00058     logging pprint("Importing model file.", 1)
00059
00060     if(not os.path.isfile(input_model)):
00061         logging pprint(f"Model file at {input_model} not found. Check the file path, or working
    directory")
00062         exit(1)
00063
00064     model.Import(input_model)
00065     logging pprint("Model imported successfully.", 2)
00066     return model
00067
00068 def cli_train(model : markopy.ModelMatrix, dataset : str, seperator : str, output : str, output_forced
    : bool=False, bulk : bool=False):
00069     """ Train a model via CLI parameters
00070     @param model Model instance
00071     @param dataset filename for the dataset
00072     @param seperator seperator used with the dataset
00073     @param output output filename
00074     @param output_forced force overwrite
00075     @param bulk marks bulk operation with directories
00076     """
00077     if not (dataset and seperator and (output or not output_forced)):
00078         logging pprint(f"Training mode requires -d/--dataset(',-o/--output' if output_forced else")
    and -s/--separator parameters. Exiting.")
00079         exit(2)
00080
00081     if(not bulk and not os.path.isfile(dataset)):
00082         logging pprint(f"{dataset} doesn't exists. Check the file path, or working directory")
00083         exit(3)
00084
00085     if(output and os.path.isfile(output)):
00086         logging pprint(f"{output} exists and will be overwritten.", 1 )
00087
00088     if(seperator == '\\\\t'):
00089         logging pprint("Escaping seperator.", 3)
00090         seperator = '\\t'
00091
00092     if(len(seperator)!=1):
00093         logging pprint(f'Delimiter must be a single character, and "{seperator}" is not accepted.')
00094         exit(4)
00095
00096     logging pprint(f'Starting training.', 3)
00097     model.Train(dataset,seperator, int(args.threads))
00098     logging pprint(f'Training completed.', 2)
00099
00100    if(output):
00101        logging pprint(f'Exporting model to {output}', 2)
00102        model.Export(output)

```

```

00103     else:
00104         logging pprint(f'Model will not be exported.', 1)
00105
00106 def cli_generate(model : markopy.ModelMatrix, wordlist : str, bulk : bool=False):
00107     """ Generate strings from the model
00108     @param model: model instance
00109     @param wordlist wordlist filename
00110     @param bulk marks bulk operation with directories
00111     """
00112     if not (wordlist or args.count):
00113         logging pprint("Generation mode requires -w/--wordlist and -n/--count parameters. Exiting.")
00114         exit(2)
00115
00116     if(bulk and os.path.isfile(wordlist)):
00117         logging pprint(f'{wordlist} exists and will be overwritten.", 1)
00118     model.Generate(int(args.count), wordlist, int(args.min), int(args.max), int(args.threads))
00119
00120
00121 if(args.bulk):
00122     logging pprint(f'Bulk mode operation chosen.', 4)
00123
00124     if (args.mode.lower() == "train"):
00125         if (os.path.isdir(args.output) and not os.path.isfile(args.output)) and
00126             (os.path.isdir(args.dataset) and not os.path.isfile(args.dataset)):
00127             corpus_list = os.listdir(args.dataset)
00128             for corpus in corpus_list:
00129                 model = cli_init(args.input)
00130                 logging pprint(f"Training {args.input} with {corpus}", 2)
00131                 output_file_name = corpus
00132                 model_extension = ""
00133                 if "." in args.input:
00134                     model_extension = args.input.split(".")[-1]
00135                 cli_train(model, f'{args.dataset}/{corpus}', args.separator,
00136                           f'{args.output}/{corpus}.{model_extension}', output_forced=True, bulk=True)
00137             else:
00138                 logging pprint("In bulk training, output and dataset should be a directory.")
00139                 exit(1)
00140
00141     elif (args.mode.lower() == "generate"):
00142         if (os.path.isdir(args.wordlist) and not os.path.isfile(args.wordlist)) and
00143             (os.path.isdir(args.input) and not os.path.isfile(args.input)):
00144             model_list = os.listdir(args.input)
00145             print(model_list)
00146             for input in model_list:
00147                 logging pprint(f"Generating from {args.input}/{input} to {args.wordlist}/{input}.txt",
00148                               2)
00149
00150             model = cli_init(f'{args.input}/{input}')
00151             model_base = input
00152             if "." in args.input:
00153                 model_base = input.split(".")[1]
00154             cli_generate(model, f'{args.wordlist}/{model_base}.txt', bulk=True)
00155         else:
00156             logging pprint("In bulk generation, input and wordlist should be directory.")
00157
00158     model = cli_init(args.input)
00159     if (args.mode.lower() == "generate"):
00160         cli_generate(model, args.wordlist)
00161
00162     elif (args.mode.lower() == "train"):
00163         cli_train(model, args.dataset, args.separator, args.output, output_forced=True)
00164
00165     elif(args.mode.lower() == "combine"):
00166         cli_train(model, args.dataset, args.separator, args.output)
00167         cli_generate(model, args.wordlist)
00168
00169     else:
00170         logging pprint("Invalid mode argument given.")
00171         logging pprint("Accepted modes: 'Generate', 'Train', 'Combine'")
00172         exit(5)

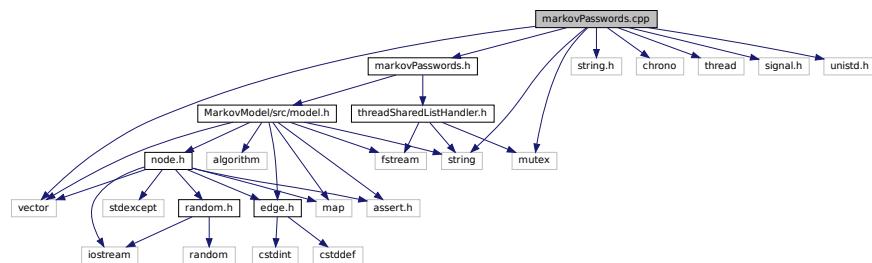
```

## 10.43 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

### 10.43.1 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](#).

### 10.43.2 Function Documentation

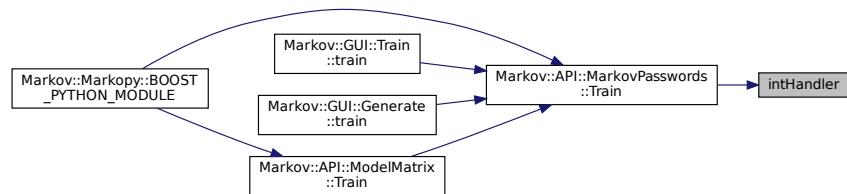
#### 10.43.2.1 intHandler()

```
void intHandler (
    int dummy )
Definition at line 26 of file markovPasswords.cpp.
00026     {
00027         std::cout << "You wanted this man by presing CTRL-C ! Ok bye." ;
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:



### 10.43.3 Variable Documentation

#### 10.43.3.1 keepRunning

```
volatile int keepRunning = 1 [static]
Definition at line 24 of file markovPasswords.cpp.
Referenced by intHandler(), and Markov::API::MarkovPasswords::TrainThread().
```

## 10.44 markovPasswords.cpp

```

00001 /**
00002 * @file markovPasswords.cpp
00003 * @brief Wrapper for Markov::Model to use with char represented models.
00004 * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
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 << "You wanted this man by presing CTRL-C ! Ok bye." ;
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         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;
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     }

```

```

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::ofstream *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 }

```

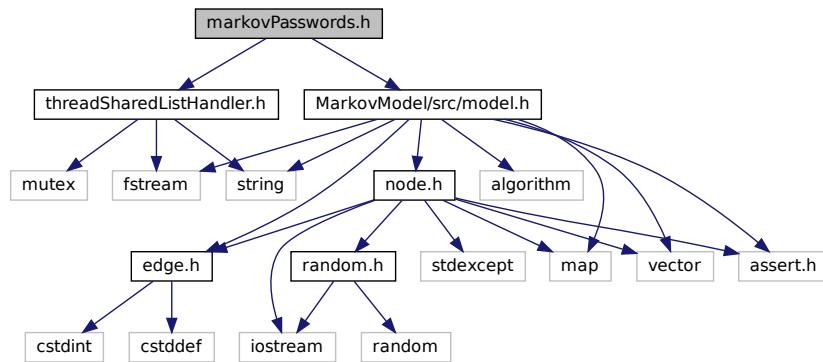
## 10.45 markovPasswords.h File Reference

Wrapper for [Markov::Model](#) to use with char represented models.

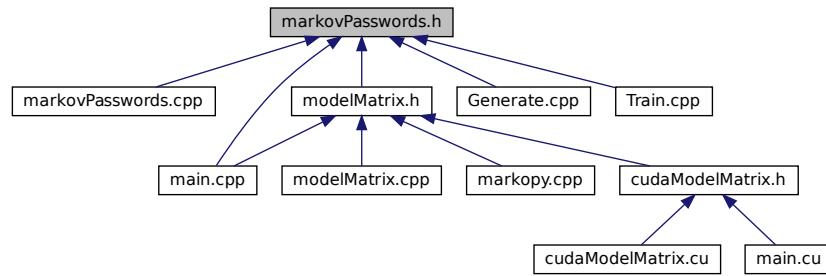
```
#include "threadSharedListHandler.h"
```

```
#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.*

### 10.45.1 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](#).

## 10.46 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      *
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      * @param filename - Filename to import
00041      *
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,
00092      int threads=20);
00093
00094      /** @brief Buff expression of some characters in the model
00095      * @param str A string containing all the characters to be buffed
00096      * @param multiplier A constant value to buff the nodes with.
00097      * @param bDontAdjustSelfEdges Do not adjust weights if target node is same as source node
00098      * @param bDontAdjustExtendedLoops Do not adjust if both source and target nodes are in first
00099      * parameter
00100      */
00101  void Buff(const char* str, double multiplier, bool bDontAdjustSelfLoops=true, bool
bDontAdjustExtendedLoops=false);
00102
00103  private:
00104

```

```

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
00109     delimiter);
00110
00111     /** @brief A single thread invoked by the Generate function.
00112      *
00113      * @b DEPRECATED: See Markov::API::MatrixModel::FastRandomWalkThread for more information.
00114      This has been replaced with
00115      * a much more performance-optimized method. FastRandomWalk will reduce the runtime by %96.5
00116      on average.
00117      *
00118      * @param outputLock - shared mutex lock to lock during output operation. Prevents race
00119      condition on write.
00120      *
00121      */
00122     void GenerateThread(std::mutex *outputLock, unsigned long int n, std::ofstream *wordlist, int
00123     minLen, int maxLen);
00124     std::ifstream* datasetFile; /* @brief Dataset file input of our system */
00125     std::ofstream* modelSavefile; /* @brief File to save model of our system */
00126     std::ofstream* outputFile; /* @brief Generated output file of our system */
00127
00128
00129
00130 };

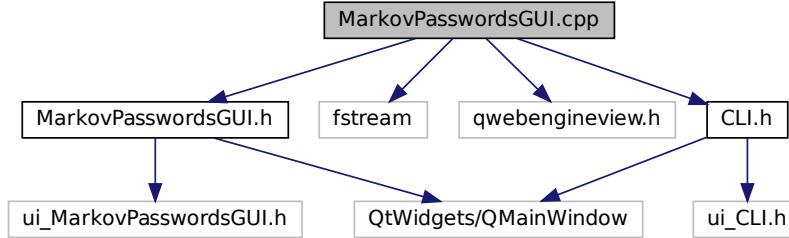
```

## 10.47 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:



### 10.47.1 Detailed Description

Main activity page for GUI.

#### Authors

Yunus Emre Yılmaz

Definition in file [MarkovPasswordsGUI.cpp](#).

## 10.48 MarkovPasswordsGUI.cpp

```
00001 /** @file MarkovPasswordsGUI.cpp
```

```

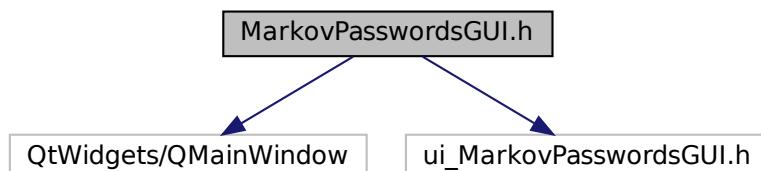
00002 * @brief Main activity page for GUI
00003 * @authors Yunus Emre Yilmaz
00004 *
00005 */
00006
00007 #include "MarkovPasswordsGUI.h"
00008 #include <fstream>
00009 #include <qwebengineview.h>
00010 #include "CLI.h"
00011
00012 using namespace Markov::GUI;
00013
00014 Markov::GUI::MarkovPasswordsGUI::MarkovPasswordsGUI(QWidget *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 }

```

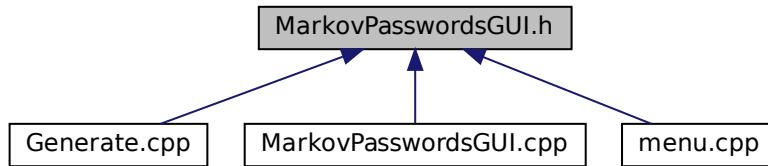
## 10.49 MarkovPasswordsGUI.h File Reference

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*

### 10.49.1 Detailed Description

Main activity page for GUI.

#### Authors

Yunus Emre Yilmaz

Definition in file [MarkovPasswordsGUI.h](#).

## 10.50 MarkovPasswordsGUI.h

```

00001 /** @file MarkovPasswordsGUI.h
00002 * @brief Main activity page for GUI
00003 * @authors Yunus Emre Yilmaz
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 MarkovPassword
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
  
```

```

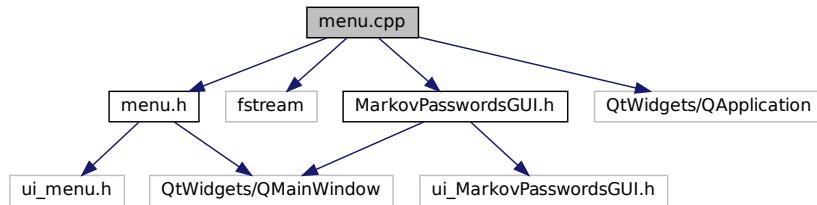
00031     void benchmarkSelected();
00032     //void MarkovPasswordsGUI::modelvisSelected();
00033     //void MarkovPasswordsGUI::visualDebugSelected();
00034     //void MarkovPasswordsGUI::comparisonSelected();
00035
00036
00037 public slots:
00038
00039     void home();
00040     void pass();
00041     void model();
00042 };
00043 };

```

## 10.51 menu.cpp File Reference

menu page

```
#include "menu.h"
#include <fstream>
#include "MarkovPasswordsGUI.h"
#include <QtWidgets/QApplication>
Include dependency graph for menu.cpp:
```



### 10.51.1 Detailed Description

menu page

Authors

Yunus Emre Yılmaz

Definition in file [menu.cpp](#).

## 10.52 menu.cpp

```

00001 /**
00002 * @file menu.cpp
00003 * @brief menu page
00004 * @authors Yunus Emre Yılmaz
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() {

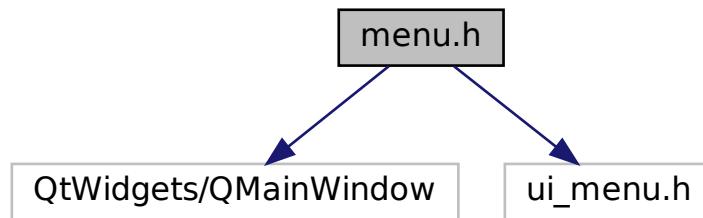
```

```

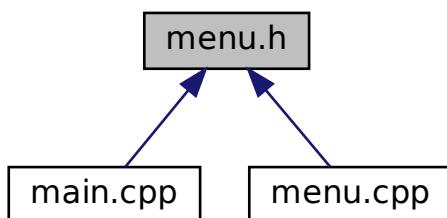
00024
00025
00026 }
00027 void menu::visualization() {
00028     MarkovPasswordsGUI* w = new MarkovPasswordsGUI;
00029     w->show();
00030     this->close();
00031 }
```

## 10.53 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*

### 10.53.1 Detailed Description

menu page

Authors

Yunus Emre Yılmaz

Definition in file [menu.h](#).

## 10.54 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 };

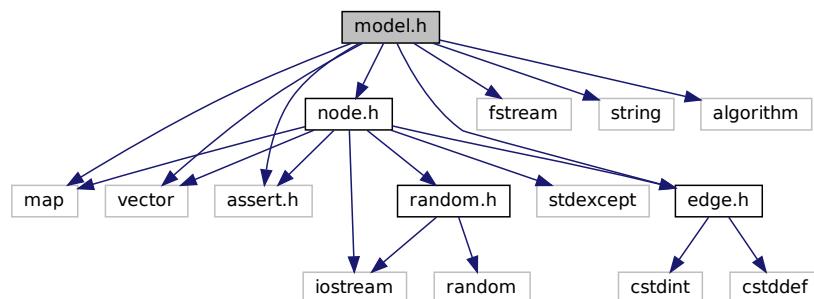
```

## 10.55 model.h File Reference

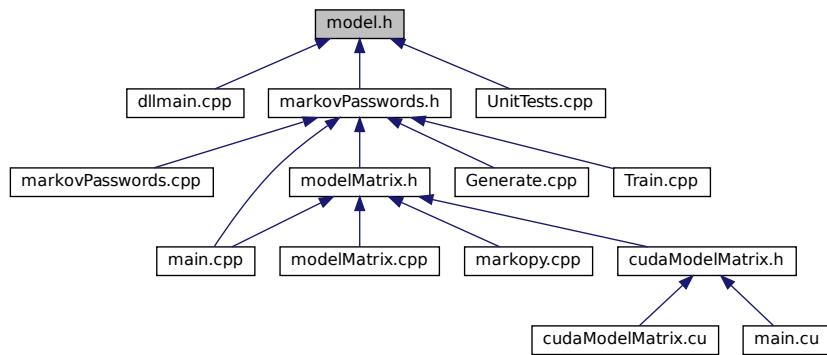
Model class template.

```
#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:



This graph shows which files directly or indirectly include this file:



## Classes

- class [Markov::Node< storageType >](#)  
*A node class that for the vertices of model. Connected with eachother using [Edge](#).*
- class [Markov::Edge< NodeStorageType >](#)  
*Edge class used to link nodes in the model together.*
- 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.*

### 10.55.1 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](#).

## 10.56 model.h

```

00001 /** @file model.h
00002 * @brief Model class template
00003 * @authors Ata Hakçıl, Osman Ömer Yıldıztugay
00004 *
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  * @brief Namespace for the markov-model related classes.
00021  * Contains Model, Node and Edge classes
00022 */
00023 namespace Markov {
00024
00025     template <typename NodeStorageType>
00026     class Node;
00027
00028     template <typename NodeStorageType>
00029     class Edge;
00030
00031     template <typename NodeStorageType>
00032
00033     /** @brief class for the final Markov Model, constructed from nodes and edges.
00034      *
00035      * Each atomic piece of the generation result is stored in a node, while edges contain the
00036      * relation weights.
00037      * *Extending:*
00038      * To extend the class, implement the template and inherit from it, as "class MyModel : public
00039      * Markov::Model<char>".
00040      * For a complete demonstration of how to extend the class, see MarkovPasswords.
00041      * Whole model can be defined as a list of the edges, as dangling nodes are pointless. This
00042      * approach is used for the import/export operations.
00043      * For more information on importing/exporting model, check out the github readme and wiki page.
00044      */
00045     class Model {
00046     public:
00047
00048     /** @brief Initialize a model with only start and end nodes.
00049      *
00050      * Initialize an empty model with only a starterNode
00051      * Starter node is a special kind of node that has constant 0x00 value, and will be used to
00052      * initiate the generation execution from.
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      * If maximum length criteria is reached but final node is not, cut off the generation and
00062      * proceed to the final node.
00063      * This function takes Markov::Random::RandomEngine as a parameter to generate pseudo random
00064      * numbers from
00065      * This library is shipped with two random engines, Marsaglia and Mersenne. While mersenne
00066      * output is higher in entropy, most use cases
00067      * don't really need super high entropy output, so Markov::Random::Marsaglia is preferable for
00068      * better performance.
00069      *
00070      * This function WILL NOT reallocate buffer. Make sure no out of bound writes are happening
00071      * via maximum length criteria.
00072
00073     * @b Example @b Use: Generate 10 lines, with 5 to 10 characters, and print the output. Use
00074     * Marsaglia
00075     * @code{.cpp}
00076     * Markov::Model<char> model;
00077     * Model.import("model.mdl");
00078     * char* res = new char[11];
00079     * Markov::Random::Marsaglia MarsagliaRandomEngine;
00080     * for (int i = 0; i < 10; i++) {
00081     *     this->RandomWalk(&MarsagliaRandomEngine, 5, 10, res);
00082     *     std::cout << res << "\n";
00083     * }
00084
00085     * @endcode
00086
00087     * @param randomEngine Random Engine to use for the random walks. For examples, see
00088     * Markov::Random::Mersenne and Markov::Random::Marsaglia
00089     * @param minSetting Minimum number of characters to generate
00090     * @param maxSetting Maximum number of character to generate
00091     * @param buffer buffer to write the result to
00092     * @return Null terminated string that was generated.

```

```

00085      */
00086      NodeStorageType* RandomWalk(Markov::Random::RandomEngine* randomEngine, int minSetting, int
00087      maxSetting, NodeStorageType* buffer);
00088
00089      /** @brief Adjust the model with a single string.
00090      *
00091      * Start from the starter node, and for each character, AdjustEdge the edge EdgeWeight from
00092      * current node to the next, until NULL character is reached.
00093      *
00094      * Then, update the edge EdgeWeight from current node, to the terminator node.
00095      *
00096      * This function is used for training purposes, as it can be used for adjusting the model with
00097      * each line of the corpus file.
00098      *
00099      * @b Example @b Use: Create an empty model and train it with string: "testdata"
00100      * @code{.cpp}
00101      * Markov::Model<char> model;
00102      * char test[] = "testdata";
00103      * model.AdjustEdge(test, 15);
00104      * @endcode
00105      *
00106      * @param string - String that is passed from the training, and will be used to AdjustEdge the
00107      * model with
00108      * @param occurrence - Occurrence of this string.
00109      */
00110      void AdjustEdge(const NodeStorageType* payload, long int occurrence);
00111
00112      /** @brief Import a file to construct the model.
00113      *
00114      * File contains a list of edges. For more info on the file format, check out the wiki and
00115      * github readme pages.
00116      * Format is: Left_repr;EdgeWeight;right_repr
00117      *
00118      * Iterate over this list, and construct nodes and edges accordingly.
00119      * @return True if successful, False for incomplete models or corrupt file formats
00120      *
00121      * @b Example @b Use: Import a file from ifstream
00122      * @code{.cpp}
00123      * Markov::Model<char> model;
00124      * std::ifstream file("test.mdl");
00125      * model.Import(&file);
00126      * @endcode
00127      */
00128      bool Import(std::ifstream* );
00129
00130      /** @brief Open a file to import with filename, and call bool Model::Import with std::ifstream
00131      * @return True if successful, False for incomplete models or corrupt file formats
00132      *
00133      * @b Example @b Use: Import a file with filename
00134      * @code{.cpp}
00135      * Markov::Model<char> model;
00136      * model.Import("test.mdl");
00137      * @endcode
00138      */
00139      bool Import(const char* filename);
00140
00141      /** @brief Export a file of the model.
00142      *
00143      * File contains a list of edges.
00144      * Format is: Left_repr;EdgeWeight;right_repr.
00145      * For more information on the format, check out the project wiki or github readme.
00146      *
00147      * Iterate over this vertices, and their edges, and write them to file.
00148      * @return True if successful, False for incomplete models.
00149      *
00150      * @b Example @b Use: Export file to ofstream
00151      * @code{.cpp}
00152      * Markov::Model<char> model;
00153      * std::ofstream file("test.mdl");
00154      * model.Export(&file);
00155      * @endcode
00156      */
00157      bool Export(std::ofstream* );
00158
00159      /** @brief Open a file to export with filename, and call bool Model::Export with std::ofstream
00160      * @return True if successful, False for incomplete models or corrupt file formats
00161      *
00162      * @b Example @b Use: Export file to filename
00163      * @code{.cpp}
00164      * Markov::Model<char> model;
00165      * model.Export("test.mdl");
00166      * @endcode
00167      */
00168      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     /**
00201      * @brief A list of all edges in this model.
00202      */
00203     std::vector<Edge<NodeStorageType>> edges;
00204 };
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     }
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 }
00278
00279 template <typename NodeStorageType>
00280 bool Markov::Model<NodeStorageType>::Import(const char* filename) {
00281     std::ifstream importfile;
00282     importfile.open(filename);
00283     return this->Import(&importfile);
00284 }
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         *f << e->LeftNode()->NodeValue() << "," << e->EdgeWeight() << "," << e->RightNode()->NodeValue() <
00294         "\n";
00295     }
00296     return true;
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         } 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 }
00335

```

```

00336 template <typename NodeStorageType>
00337 void Markov::Model<NodeStorageType>::AdjustEdge(const NodeStorageType* payload, long int occurrence) {
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 }
```

## 10.57 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*

## 10.58 model\_2gram.py

```

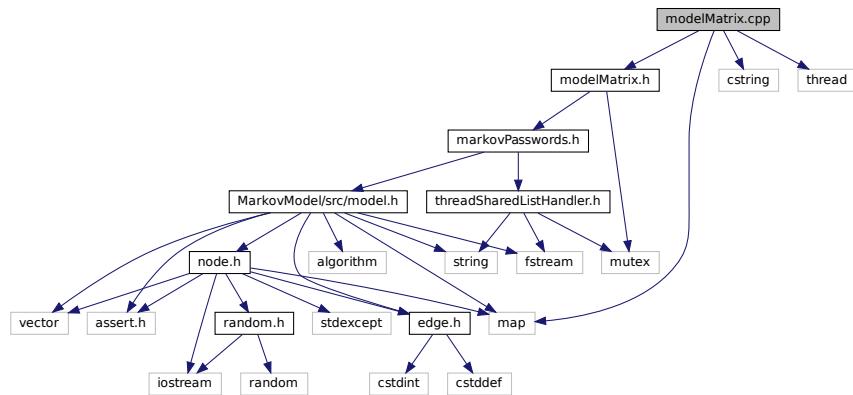
00001 #!/usr/bin/python3
00002 """
00003     python script for generating a 2gram model
00004 """
00005
00006 import string
00007 import re
00008
00009
00010 alphabet = string.printable
00011 alphabet = re.sub('\s', " ", alphabet)
00012 print(f"alphabet={alphabet}")
00013 #exit()
00014
00015
00016 f = 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")
```

## 10.59 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:
```



### 10.59.1 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](#).

## 10.60 modelMatrix.cpp

```

00001 /**
00002 * @brief An extension of Markov::API::MarkovPasswords
00003 * @authors Ata Hakçıl
00004 *
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 << "\\\\\\\\x0f";
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 }
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     int len=0;
00163     long int selection;
00164     char cur;
00165     long int bufferctr = 0;
00166     for (int i = 0; i < n; i++) {
00167         cur=199;
00168         len=0;
00169         while (true) {
00170             e = strchr(this->matrixIndex, cur);
00171             index = e - this->matrixIndex;
00172             selection = MarsagliaRandomEngine.random() % this->totalEdgeWeights[index];
00173             for(int j=0;j<this->matrixSize;j++){
00174                 selection -= this->valueMatrix[index][j];
00175                 if (selection < 0){
00176                     next = this->edgeMatrix[index][j];
00177                     break;
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){  
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 }  
00218  
00219 int Markov::API::ModelMatrix::FastRandomWalk(unsigned long int n, const char* wordlistFileName, int  
00220     minLen, int maxLen, int threads, bool bFileIO){  
00221     std::ofstream wordlist;  
00222     if(bFileIO)  
00223         wordlist.open(wordlistFileName);  
00224     this->FastRandomWalk(n, &wordlist, minLen, maxLen, threads, bFileIO);  
00225     return 0;  
00226 }  
00227  
00228 void Markov::API::ModelMatrix::FastRandomWalkPartition(std::mutex *mlock, std::ofstream *wordlist,  
00229     unsigned long int n, int minLen, int maxLen, bool bFileIO, int threads){  
00230     int iterationsPerThread = n/threads;  
00231     int iterationsPerThreadCarryOver = n%threads;  
00232  
00233     std::vector<std::thread*> threadsV;  
00234     int id = 0;  
00235     for(int i=0;i<threads;i++){  
00236         threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this,  
00237             mlock, wordlist, iterationsPerThread, minLen, maxLen, id, bFileIO));  
00238         id++;  
00239     }  
00240     threadsV.push_back(new std::thread(&Markov::API::ModelMatrix::FastRandomWalkThread, this, mlock,  
00241             wordlist, iterationsPerThreadCarryOver, minLen, maxLen, id, bFileIO));  
00242     for(int i=0;i<threads;i++){  
00243         threadsV[i]->join();  
00244     }  
00245 }
```

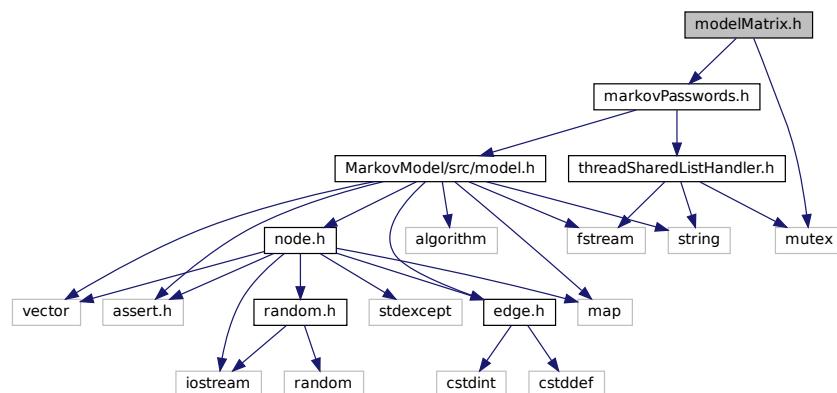
## 10.61 modelMatrix.h File Reference

An extension of [Markov::API::MarkovPasswords](#).

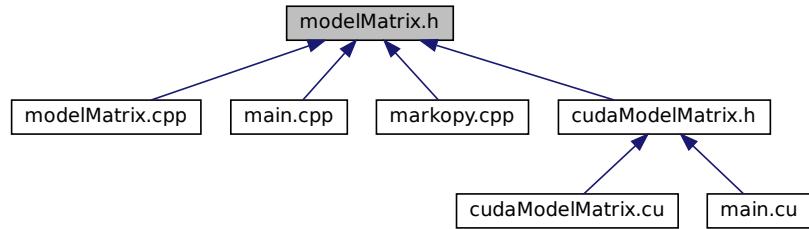
```
#include "markovPasswords.h"
```

```
#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](#).*

### 10.61.1 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](#).

## 10.62 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)
  
```

```

00019      *
00020      * To limit the maximum memory usage, each generation operation is partitioned into 50M chunks for
00021      * allocation. Threads are synchronized and files are flushed every 50M operations.
00022      *
00023      */
00024  class ModelMatrix : public Markov::API::MarkovPasswords{
00025  public:
00026      ModelMatrix();
00027
00028      /** @brief Construct the related Matrix data for the model.
00029      *
00030      * This operation can be used after importing/training to allocate and populate the matrix
00031      * content.
00032      *
00033      * this will initialize:
00034      * char** edgeMatrix -> a 2D array of mapping left and right connections of each edge.
00035      * long int **valueMatrix -> a 2D array representing the edge weights.
00036      * int matrixSize -> Size of the matrix, aka total number of nodes.
00037      * char* matrixIndex -> order of nodes in the model
00038      * long int *totalEdgeWeights -> total edge weights of each Node.
00039      *
00040      * @returns True if constructed. False if already constructed.
00041      */
00042  bool ConstructMatrix();
00043
00044      /** @brief Debug function to dump the model to a JSON file.
00045      *
00046      * Might not work 100%. Not meant for production use.
00047      */
00048  void DumpJSON();
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      * While it has the same functionality, this operation reduces
00059      * Markov::API::MarkovPasswords::Generate runtime by ~96.5
00060      *
00061      * This function has deprecated Markov::API::MarkovPasswords::Generate, and will eventually
00062      * replace it.
00063      *
00064      * @param n - Number of passwords to generate.
00065      * @param wordlistFileName - Filename to write to
00066      * @param minLen - Minimum password length to generate
00067      * @param maxLen - Maximum password length to generate
00068      * @param threads - number of OS threads to spawn
00069      * @param bFileIO - If false, filename will be ignored and will output to stdout.
00070      *
00071      * @endcode{.cpp}
00072      * Markov::API::ModelMatrix mp;
00073      * mp.Import("models/finished.mdl");
00074      * mp.FastRandomWalk(50000000,"./wordlist.txt",6,12,25, true);
00075      *
00076      int FastRandomWalk(unsigned long int n, const char* wordlistFileName, int minLen=6, int
00077      maxLen=12, int threads=20, bool bFileIO=true);
00078      /** @copydoc Markov::Model::Import(const char *filename)
00079      * Construct the matrix when done.
00080      *
00081      */
00082  void Import(const char *filename);
00083
00084      /** @copydoc Markov::API::MarkovPasswords::Train(const char *datasetFileName, char delimiter,
00085      * int threads)
00086      * Construct the matrix when done.
00087      *
00088      void Train(const char *datasetFileName, char delimiter, int threads);
00089
00090  protected:
00091
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

```

```

        be reallocated every 50M generations.
00097     * This comes at a minor performance penalty.
00098     *
00099     * 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
replace it.
00102     *
00103     * @param n - Number of passwords to generate.
00104     * @param wordlistFileName - Filename to write to
00105     * @param minLen - Minimum password length to generate
00106     * @param maxLen - Maximum password length to generate
00107     * @param threads - number of OS threads to spawn
00108     * @param bFileIO - If false, filename will be ignored and will output to stdout.
00109     *
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,
int threads=20, bool bFileIO=true);
00119
00120
00121     /** @brief A single partition of FastRandomWalk event
00122     *
00123     * Since FastRandomWalk has to allocate its output buffer before operation starts and writes
data in chunks,
00124     * large n parameters would lead to huge memory allocations.
00125     * @b Without @b Partitioning:
00126     * - 50M results 12 characters max -> 550 Mb Memory allocation
00127     *
00128     * - 5B results 12 characters max -> 55 Gb Memory allocation
00129     *
00130     * - 50B results 12 characters max -> 550GB Memory allocation
00131     *
00132     * Instead, FastRandomWalk is partitioned per 50M generations to limit the top memory need.
00133     *
00134     * @param mlock - mutex lock to distribute to child threads
00135     * @param wordlist - Reference to the wordlist file to write to
00136     * @param n - Number of passwords to generate.
00137     * @param wordlistFileName - Filename to write to
00138     * @param minLen - Minimum password length to generate
00139     * @param maxLen - Maximum password length to generate
00140     * @param threads - number of OS threads to spawn
00141     * @param bFileIO - If false, filename will be ignored and will output to stdout.
00142     *
00143     *
00144     */
00145     void FastRandomWalkPartition(std::mutex *mlock, std::ofstream *wordlist, unsigned long int n,
int minLen, int maxLen, bool bFileIO, int threads);
00146
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
minLen, int maxLen, int id, bool bFileIO);
00165
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 /**
00183  @brief to hold Matrix size
00184 */
00185 int matrixSize;
00186
00187 /**
00188  @brief to hold the Matrix index (To hold the orders of 2-D arrays')
00189 */
00190 char* matrixIndex;
00191
00192 /**
00193  @brief Array of the Total Edge Weights
00194 */
00195 long int *totalEdgeWeights;
00196
00197 /**
00198  @brief True when matrix is constructed. False if not.
00199 */
00200 bool ready;
00201 };
00202
00203
00204
00205 };

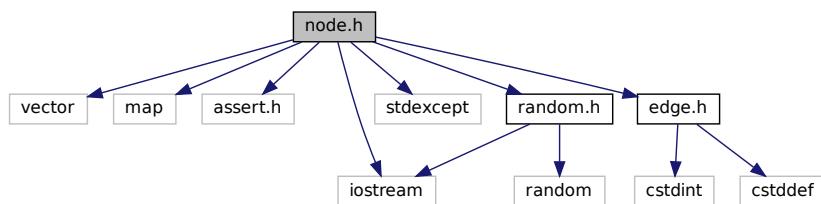
```

## 10.63 node.h File Reference

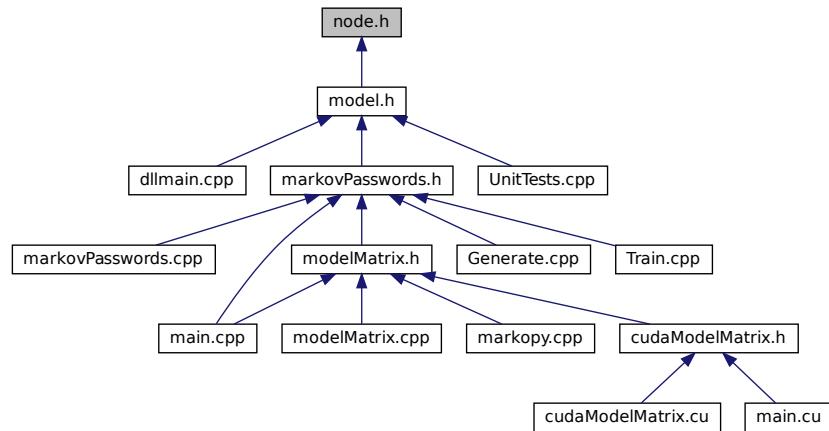
Node class template.

```
#include <vector>
#include <map>
#include <assert.h>
#include <iostream>
#include <stdexcept>
#include "edge.h"
#include "random.h"
```

Include dependency graph for node.h:



This graph shows which files directly or indirectly include this file:



## 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.*

### 10.63.1 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](#).

## 10.64 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     /** @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     /** @brief Default constructor. Creates an empty Node.
00027     */
00028     Node<storageType>();
00029
00030     /** @brief Constructor. Creates a Node with no edges and with given NodeValue.
00031      * @param _value - Nodes character representation.
00032      */
00033     * @b Example @b Use: Construct nodes
00034     * @code{.cpp}
00035     * Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00036     * Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00037     * @endcode
00038     */
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
00074      * TraverseNode to that.
00075      */
00076     * This operation is done by generating a random number in range of 0-this.total_edge_weights,
00077     * and then iterating over the list of edges.
00078     * At each step, EdgeWeight of the edge is subtracted from the random number, and once it is
00079     * 0, next node is selected.
00080     * @return Node that was chosen at EdgeWeight biased random.
00081
00082     * @b Example @b Use: Use randomNext to do a random walk on the model
00083     * @code{.cpp}
00084     * char* buffer[64];
00085     * Markov::Model<char> model;
00086     * model.Import("model.mdl");
00087     * Markov::Node<char>* n = model.starterNode;
00088     * int len = 0;
00089     * Markov::Node<char>* temp_node;
00090     * while (true) {
00091         *     temp_node = n->RandomNext(randomEngine);
00092         *     if (len >= maxSetting) {
00093             *         break;
00094             *     }
00095         *     else if ((temp_node == NULL) && (len < minSetting)) {
00096             *         continue;
00097             *     }
00098         *     else if (temp_node == NULL){
00099             *         break;
00100             *     }
00101         *     n = temp_node;
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>* el = new Markov::Edge<unsigned char>(src, target1);
00119   * Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00120   * el->AdjustEdge(25);
00121   * src->UpdateEdges(el);
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
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
00246     //if this assertion is reached, it means there is an implementation error above
00247     std::cout << "This should never be reached (node failed to walk to next)\n"; //cant assert from
00248     child thread
00249     assert(true && "This should never be reached (node failed to walk to next)");
00250     return NULL;
00251 }
00252
00253 template <typename storageType>
00254 bool Markov::Node<storageType>::UpdateEdges(Markov::Edge<storageType>* v) {
00255     this->edges.insert({ v->RightNode()->NodeValue(), v });
00256     this->edgesV.push_back(v);
00257     //this->total_edge_weights += v->EdgeWeight();
00258     return v->TraverseNode();
00259 }
00260
00261 template <typename storageType>
00262 Markov::Edge<storageType>* Markov::Node<storageType>::FindEdge(storageType repr) {
00263     auto e = this->edges.find(repr);
00264     if (e == this->edges.end()) return NULL;
00265     return e->second;
00266 }
00267
00268 template <typename storageType>
00269 void Markov::Node<storageType>::UpdateTotalVerticeWeight(long int offset) {
00270     this->total_edge_weights += offset;
00271 }
00272
00273 template <typename storageType>
00274 inline std::map<storageType, Markov::Edge<storageType>*>& Markov::Node<storageType>::Edges() {
00275     return &(this->edges);
00276 }
```

```

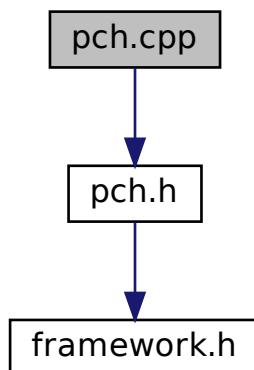
00275
00276 template <typename storageType>
00277 inline long int Markov::Node<storageType>::TotalEdgeWeights() {
00278     return this->total_edge_weights;
00279 }
```

## 10.65 pch.cpp File Reference

For windows dynamic library.

```
#include "pch.h"
```

Include dependency graph for MarkovModel/src/pch.cpp:



### 10.65.1 Detailed Description

For windows dynamic library.

#### Authors

Ata Hakçıl

Definition in file [MarkovModel/src/pch.cpp](#).

## 10.66 MarkovModel/src/pch.cpp

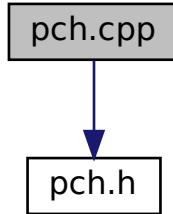
```

00001 /**
00002 * @file pch.cpp
00003 * @brief For windows dynamic library
00004 * @authors Ata Hakçıl
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.
```

## 10.67 pch.cpp File Reference

For windows dynamic library.

```
#include "pch.h"
Include dependency graph for UnitTests/pch.cpp:
```



### 10.67.1 Detailed Description

For windows dynamic library.

#### Authors

Ata Hakçıl

Definition in file [UnitTests/pch.cpp](#).

## 10.68 UnitTests/pch.cpp

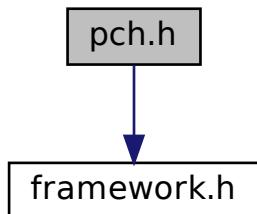
```
00001 /** @file pch.cpp
00002 * @brief For windows dynamic library
00003 * @authors Ata Hakçıl
00004 *
00005 */
00006 // pch.cpp: source file corresponding to the pre-compiled header
00007
00008 #include "pch.h"
00009
00010 // When you are using pre-compiled headers, this source file is necessary for compilation to succeed.
```

### 10.69 pch.h File Reference

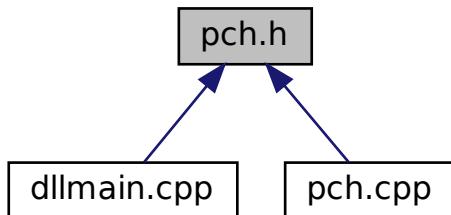
For windows dynamic library.

```
#include "framework.h"
```

Include dependency graph for MarkovModel/src/pch.h:



This graph shows which files directly or indirectly include this file:



### 10.69.1 Detailed Description

For windows dynamic library.

#### Authors

Ata Hakçıl

Definition in file [MarkovModel/src/pch.h](#).

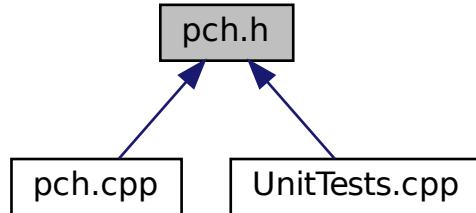
## 10.70 MarkovModel/src/pch.h

```
00001 /** @file pch.h
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
00015 // add headers that you want to pre-compile here
00016 #include "framework.h"
00017
00018 #endif //PCH_H
```

## 10.71 pch.h File Reference

For windows dynamic library.

This graph shows which files directly or indirectly include this file:



### 10.71.1 Detailed Description

For windows dynamic library.

#### Authors

Ata Hakçıl

Definition in file [UnitTests/pch.h](#).

## 10.72 UnitTests/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
00015 // add headers that you want to pre-compile here
00016
00017 #endif //PCH_H
  
```

## 10.73 random-model.py File Reference

### Namespaces

- [random-model](#)
- [random](#)

### Variables

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

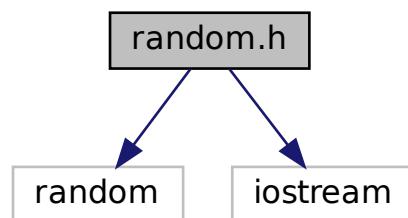
## 10.74 random-model.py

```
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")
```

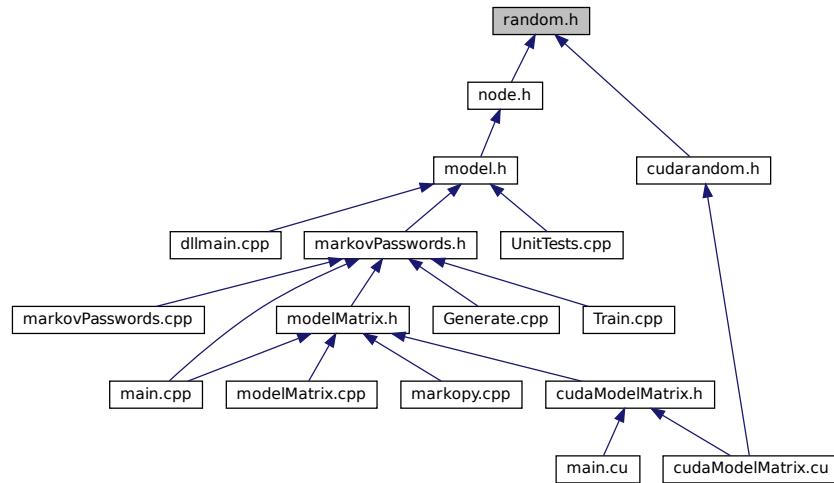
## 10.75 random.h File Reference

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.*

### 10.75.1 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](#).

## 10.76 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     *
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     /** @brief Default random device for seeding
00071      *
00072      */
00073     inline std::random_device& rd() {
00074         static std::random_device _rd;
00075         return _rd;
00076     }
00077
00078     /** @brief Default random engine for seeding
00079      *
00080      */
00081     inline std::default_random_engine& generator() {
00082         static std::default_random_engine _generator(rd());
00083         return _generator;
00084     }
00085
00086     /** @brief Distribution schema for seeding.
00087      *
00088      */
00089     inline std::uniform_int_distribution<long long unsigned>& distribution() {
00090         static std::uniform_int_distribution<long long unsigned> _distribution(0, 0xffffffff);
00091         return _distribution;
00092     }
00093 }
00094
00095 /**
00096 /**
00097     * @brief Implementation of Marsaglia Random Engine
00098     *
00099     * This is an implementation of Marsaglia Random engine, which for most use cases is a better fit
00100 than other solutions.
00101     * Very simple mathematical formula to generate pseudorandom integer, so its crazy fast.
00102     *
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

```

```

00136         //std::cout << "x: " << x << ", y: " << y << ", z: " << z << "\n";
00137     }
00138
00139
00140     inline unsigned long random() {
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     }
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 };

```

## 10.77 README.md File Reference

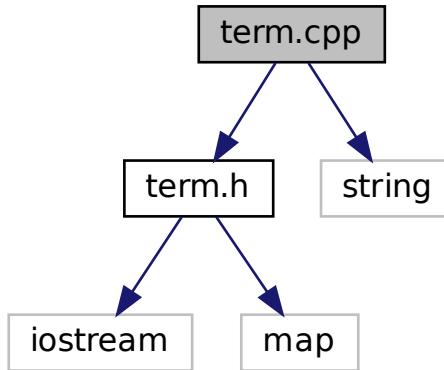
### 10.78 report.md File Reference

### 10.79 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)

### 10.79.1 Detailed Description

Terminal handler for pretty stuff like colors.

#### Authors

Ata Hakçıl

pretty colors for Terminal. Windows Only.

Definition in file [term.cpp](#).

### 10.79.2 Function Documentation

#### 10.79.2.1 operator<<()

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

overload for std::cout.

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[1;" << buf << "m";
00070     return os;
00071 }
```

References [Markov::API::CLI::Terminal::colormap](#).

## 10.80 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

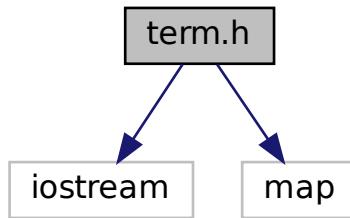
```

## 10.81 term.h File Reference

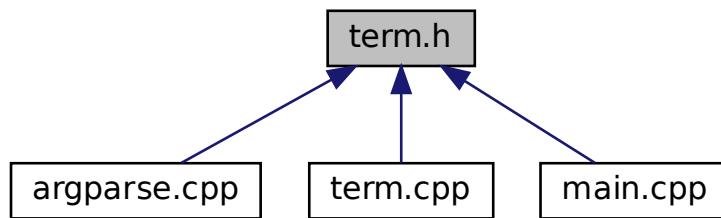
Terminal handler for pretty stuff like colors.

```
#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)

### 10.81.1 Detailed Description

Terminal handler for pretty stuff like colors.

#### Authors

Ata Hakçıl

pretty colors for Terminal. Windows Only.

Definition in file [term.h](#).

### 10.81.2 Macro Definition Documentation

#### 10.81.2.1 TERM\_FAIL

```
#define TERM_FAIL "[" << Markov::API::CLI::Terminal::color::RED << "+" << Markov::API::CLI::Terminal::color  
<< "] "
```

Definition at line 17 of file [term.h](#).

#### 10.81.2.2 TERM\_INFO

```
#define TERM_INFO "[" << Markov::API::CLI::Terminal::color::BLUE << "+" << Markov::API::CLI::Terminal::color  
<< "] "
```

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

#### 10.81.2.3 TERM\_SUCC

```
#define TERM_SUCC "[" << Markov::API::CLI::Terminal::color::GREEN << "+" << Markov::API::CLI::Terminal::color  
<< "] "
```

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

#### 10.81.2.4 TERM\_WARN

```
#define TERM_WARN "[" << Markov::API::CLI::Terminal::color::YELLOW << "+" << Markov::API::CLI::Terminal::color  
<< "] "
```

Definition at line 19 of file [term.h](#).

## 10.82 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 << "] "
```

```

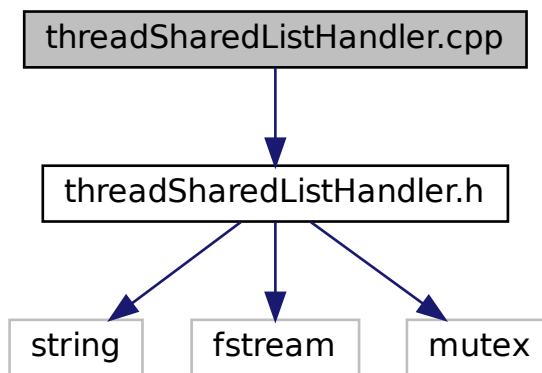
00018 #define TERM_INFO "[" « Markov::API::CLI::Terminal::color::BLUE « "+" «
Markov::API::CLI::Terminal::color::RESET « "] "
00019 #define TERM_WARN "[" « Markov::API::CLI::Terminal::color::YELLOW « "+" «
Markov::API::CLI::Terminal::color::RESET « "] "
00020 #define TERM_SUCC "[" « Markov::API::CLI::Terminal::color::GREEN « "+" «
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         static std::ostream endl;
00044
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 }
```

## 10.83 threadSharedListHandler.cpp File Reference

Thread-safe wrapper for std::ifstream.

```
#include "threadSharedListHandler.h"
```

Include dependency graph for threadSharedListHandler.cpp:



### 10.83.1 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](#).

## 10.84 threadSharedListHandler.cpp

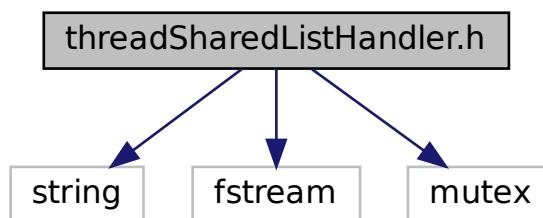
```

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 }
```

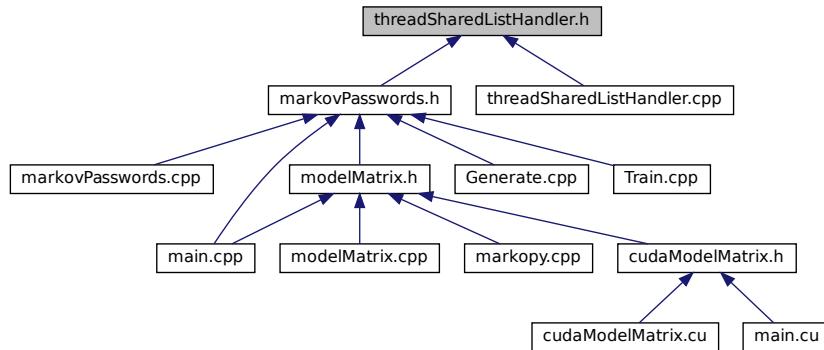
## 10.85 threadSharedListHandler.h File Reference

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.*

### 10.85.1 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](#).

## 10.86 threadSharedListHandler.h

```

00001 /** @file threadSharedListHandler.h
00002 * @brief Thread-safe wrapper for std::ifstream
00003 * @authors Ata Hakçıl
00004 *
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     *
00039     * @b Example @b Use: Example use case from MarkovPasswords showing multithreaded access
00040     * @code{.cpp}
00041     * void MarkovPasswords::Train(const char* datasetFileName, char delimiter, int threads) {
00042     *     ThreadSharedListHandler listhandler(datasetFileName);
00043     *     auto start = std::chrono::high_resolution_clock::now();
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         *         }
00075         *     @endcode
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     * @return bool next(std::string* line);
00093
00094 private:
00095     std::ifstream listfile;

```

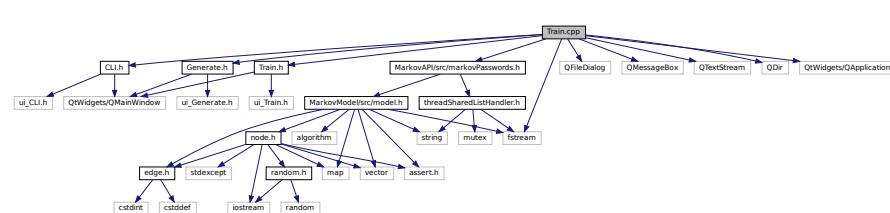
```
00096     std::mutex mlock;
00097 };
00098
00099 };
```

## 10.87 Train.cpp File Reference

training page for GUI

```
#include "Train.h"
#include <fstream>
#include <QFileDialog>
#include <QMessageBox>
#include <QTextStream>
#include <QDir>
#include "CLI.h"
#include "MarkovAPI/src/markovPasswords.h"
#include <QtWidgets/QApplication>
#include "Generate.h"
```

Include dependency graph for Train.cpp:



### 10.87.1 Detailed Description

training page for GUI

Authors

Yunus Emre Yilmaz

Definition in file [Train.cpp](#).

## 10.88 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 "CLI.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 }*/

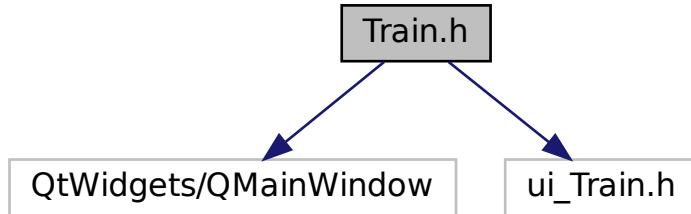
```

## 10.89 Train.h File Reference

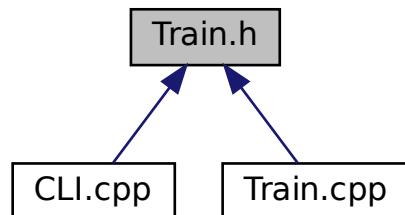
training page for GUI

```
#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*

### 10.89.1 Detailed Description

training page for GUI

#### Authors

Yunus Emre Yılmaz

Definition in file [Train.h](#).

## 10.90 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 };

```

## 10.91 UnitTests.cpp File Reference

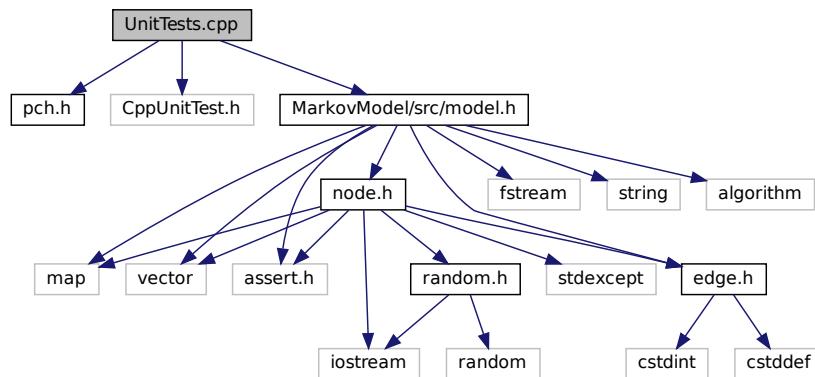
Unit tests with Microsoft::VisualStudio::CppUnitTestFramework.

```

#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.*

### 10.91.1 Detailed Description

Unit tests with Microsoft::VisualStudio::CppUnitTestFramework.

#### Authors

Ata Hakçıl, Osman Ömer Yıldıztugay, Yunus Emre Yılmaz

Definition in file [UnitTests.cpp](#).

## 10.92 UnitTests.cpp

```

00001 /**
00002 * @file UnitTests.cpp
00003 * @brief Unit tests with Microsoft::VisualStudio::CppUnitTestFramework
00004 * @authors Ata Hakçıl, Osman Ömer Yıldıztugay, Yunus Emre Yılmaz
00005 */
00006
00007 #include "pch.h"
00008 #include "CppUnitTest.h"
00009 #include "MarkovModel/src/model.h"
0010
0011 using namespace Microsoft::VisualStudio::CppUnitTestFramework;
0012
0013
0014 /** @brief Namespace for Microsoft Native Unit Testing Classes
0015 */
0016 namespace Testing {
0017
0018     /** @brief Testing Namespace for Minimal Viable Product
0019     */
0020     namespace MVP {
0021         /** @brief Testing Namespace for MVP MarkovModel
0022         */
0023         namespace MarkovModel {
0024             {
0025                 /** @brief Test class for minimal viable Edge
0026                 */
0027                 TEST_CLASS(Edge)
0028                 {
0029                     public:
0030
0031                         /** @brief test default constructor
0032                         */
0033                         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');
00059         Markov::Edge<unsigned char*>* e = new Markov::Edge<unsigned char>(LeftNode,
00060             RightNode);
00061         e->AdjustEdge(15);
00062         Assert::AreEqual(15ull, e->EdgeWeight());
00063         e->AdjustEdge(15);
00064         Assert::AreEqual(30ull, e->EdgeWeight());
00065         delete LeftNode;
00066         delete RightNode;
00067         delete e;
00068     }
00069
00070     /** @brief test TraverseNode returning RightNode
00071     */
00072     TEST_METHOD(TraverseNode) {
00073         Markov::Node<unsigned char*>* LeftNode = new Markov::Node<unsigned char>('l');
00074         Markov::Node<unsigned char*>* RightNode = new Markov::Node<unsigned char>('r');
00075         Markov::Edge<unsigned char*>* e = new Markov::Edge<unsigned char>(LeftNode,
00076             RightNode);
00077         Assert::IsTrue(RightNode == e->TraverseNode());
00078         delete LeftNode;
00079         delete RightNode;
00080         delete e;
00081     }
00082
00083     /** @brief test LeftNode/RightNode setter
00084     */
00085     TEST_METHOD(set_left_and_right) {
00086         Markov::Node<unsigned char*>* LeftNode = new Markov::Node<unsigned char>('l');
00087         Markov::Node<unsigned char*>* RightNode = new Markov::Node<unsigned char>('r');
00088         Markov::Edge<unsigned char*>* e1 = new Markov::Edge<unsigned char>(LeftNode,
00089             RightNode);
00090
00091         Markov::Edge<unsigned char*>* e2 = new Markov::Edge<unsigned char>;
00092         e2->SetLeftEdge(LeftNode);
00093         e2->SetRightEdge(RightNode);
00094
00095         Assert::IsTrue(e1->LeftNode() == e2->LeftNode());
00096         Assert::IsTrue(e1->RightNode() == e2->RightNode());
00097         delete LeftNode;
00098         delete RightNode;
00099         delete e1;
00100         delete e2;
00101     }
00102
00103     /** @brief test negative adjustments
00104     */
00105     TEST_METHOD(negative_adjust) {
00106         Markov::Node<unsigned char*>* LeftNode = new Markov::Node<unsigned char>('l');
00107         Markov::Node<unsigned char*>* RightNode = new Markov::Node<unsigned char>('r');
00108         Markov::Edge<unsigned char*>* e = new Markov::Edge<unsigned char>(LeftNode,
00109             RightNode);
00110         e->AdjustEdge(15);
00111         Assert::AreEqual(15ull, e->EdgeWeight());
00112         e->AdjustEdge(-15);
00113         Assert::AreEqual(0ull, e->EdgeWeight());
00114         delete LeftNode;
00115         delete RightNode;
00116         delete e;
00117     };
00118
00119     /** @brief Test class for minimal viable Node

```

```

00116      */
00117  TEST_CLASS(Node)
00118  {
00119  public:
00120
00121      /** @brief test default constructor
00122      */
00123  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      /** @brief test custom constructor with unsigned char
00130      */
00131  TEST_METHOD(uchar_constructor) {
00132      Markov::Node<unsigned char>* n = NULL;
00133      unsigned char test_cases[] = { 'c', 0x00, 0xff, -32 };
00134      for (unsigned char tcase : test_cases) {
00135          n = new Markov::Node<unsigned char>(tcase);
00136          Assert::AreEqual(tcase, n->nodeValue());
00137          delete n;
00138      }
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 /** @brief random next on a node with no follow-ups
00221 */
00222 TEST_METHOD(rand_next_choice_2) {
00223     Markov::Random::Marsaglia MarsagliaRandomEngine;
00224     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00225     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00226     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00227     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 /** @brief test updateEdges
00242 */
00243 TEST_METHOD(update_edges_count) {
00244
00245     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00246     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00247     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00248     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00249     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target2);
00250     e1->AdjustEdge(25);
00251     src->UpdateEdges(e1);
00252     e2->AdjustEdge(30);
00253     src->UpdateEdges(e2);
00254
00255     Assert::AreEqual((size_t)2, src->Edges()->size());
00256
00257     delete src;
00258     delete target1;
00259     delete e1;
00260     delete e2;
00261
00262 }
00263
00264 /** @brief test updateEdges
00265 */
00266 TEST_METHOD(update_edges_total) {
00267
00268     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00269     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00270     Markov::Edge<unsigned char>* e1 = new Markov::Edge<unsigned char>(src, target1);
00271     Markov::Edge<unsigned char>* e2 = new Markov::Edge<unsigned char>(src, target1);
00272     e1->AdjustEdge(25);
00273     src->UpdateEdges(e1);
00274     e2->AdjustEdge(30);
00275     src->UpdateEdges(e2);
00276
00277 //Assert::AreEqual(55ull, src->TotalEdgeWeights());
00278
00279     delete src;
00280     delete target1;
00281     delete e1;
00282     delete e2;
00283
00284
00285
00286 /** @brief test FindVertice
00287 */
00288 TEST_METHOD(find_vertice) {
00289

```

```

00290     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00291     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00292     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00293     Markov::Edge<unsigned char>* res = NULL;
00294     src->Link(target1);
00295     src->Link(target2);
00296
00297
00298     res = src->FindEdge('b');
00299     Assert::IsNotNull(res);
00300     Assert::AreEqual((unsigned char)'b', res->TraverseNode()->nodeValue());
00301     res = src->FindEdge('c');
00302     Assert::IsNotNull(res);
00303     Assert::AreEqual((unsigned char)'c', res->TraverseNode()->nodeValue());
00304
00305     delete src;
00306     delete target1;
00307     delete target2;
00308
00309
00310 }
00311
00312
00313 /** @brief test FindVertice
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 /** @brief test FindVertice
00332 */
00333 TEST_METHOD(find_vertice_nonexistent) {
00334
00335     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00336     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00337     Markov::Node<unsigned char>* target2 = new Markov::Node<unsigned char>('c');
00338     Markov::Edge<unsigned char>* res = NULL;
00339     src->Link(target1);
00340     src->Link(target2);
00341
00342     res = src->FindEdge('D');
00343     Assert::IsNull(res);
00344
00345     delete src;
00346     delete target1;
00347     delete target2;
00348 }
00349
00350
00351 /** @brief Test class for minimal viable Model
00352 */
00353 TEST_CLASS(Model)
00354 {
00355     public:
00356         /** @brief test model constructor for starter node
00357 */
00358         TEST_METHOD(model_constructor) {
00359             Markov::Model<unsigned char> m;
00360             Assert::AreEqual((unsigned char)'\\0', m.StarterNode()->nodeValue());
00361         }
00362
00363         /** @brief test import
00364 */
00365         TEST_METHOD(import_filename) {
00366             Markov::Model<unsigned char> m;
00367             Assert::IsTrue(m.Import("../MarkovPasswords/Models/2gram.mdl"));
00368         }
00369
00370         /** @brief test export
00371 */
00372         TEST_METHOD(export_filename) {
00373             Markov::Model<unsigned char> m;
00374             Assert::IsTrue(m.Export("../MarkovPasswords/Models/testcase.mdl"));
00375         }
00376

```

```

00377         /** @brief test random walk
00378         */
00379         TEST_METHOD(random_walk) {
00380             unsigned char* res = new unsigned char[12 + 5];
00381             Markov::Random::Marsaglia MarsagliaRandomEngine;
00382             Markov::Model<unsigned char> m;
00383             Assert::IsTrue(m.Import("../Models/finished2.mdl"));
00384             Assert::IsNotNull(m.RandomWalk(&MarsagliaRandomEngine, 1, 12, res));
00385         }
00386     };
00387 }
00388
00389 /** @brief Testing namespace for MVP MarkovPasswords
00390 */
00391 namespace MarkovPasswords
00392 {
00393     /** @brief Test Class for Argparse class
00394     */
00395     TEST_CLASS(ArgParser)
00396     {
00397         public:
00398             /** @brief test basic generate
00399             */
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
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             /** @brief test basic generate param longnames
00432             */
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
00438                 /*ProgramOptions* p = Argparse::parse(argc, argv);
00439                 Assert::IsNotNull(p);
00440
00441                 Assert::AreEqual(p->bImport, true);
00442                 Assert::AreEqual(p->bExport, false);
00443                 Assert::AreEqual(p->importname, "model.mdl");
00444                 Assert::AreEqual(p->outputfilename, "passwords.txt");
00445                 Assert::AreEqual(p->generateN, 100); */
00446
00447             /** @brief test basic generate
00448             */
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
00454                 /*ProgramOptions* p = Argparse::parse(argc, argv);
00455                 Assert::IsNull(p); */
00456
00457             /** @brief test basic train
00458             */
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     }
00471
00472     /** @brief test basic generate
00473     */
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     };
00489 }
00490
00491 /**
00492 ** @brief Testing namespace for MarkovModel
00493 */
00494 namespace MarkovModel {
00495
00496     /** @brief Test class for rest of Edge cases
00497     */
00498     TEST_CLASS(Edge)
00499     {
00500         public:
00501             /** @brief send exception on integer underflow
00502             */
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
00509                     RightNode);
00510                     e->AdjustEdge(15);
00511                     e->AdjustEdge(-30);
00512                     delete LeftNode;
00513                     delete RightNode;
00514                     delete e;
00515                 };
00516                 Assert::ExpectException<std::underflow_error>(_underflow_adjust);
00517             }
00518
00519             /** @brief test integer overflows
00520             */
00521             TEST_METHOD(except_integer_overflow) {
00522                 auto _overflow_adjust = [] {
00523                     Markov::Node<unsigned char>* LeftNode = new Markov::Node<unsigned char>('l');
00524                     Markov::Node<unsigned char>* RightNode = new Markov::Node<unsigned char>('r');
00525                     Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(LeftNode,
00526
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                     /** @brief test RandomNext with 64 bit high values
00543                     */
00544                     TEST_METHOD(rand_next_u64) {

```

```

00545     Markov::Random::Marsaglia MarsagliaRandomEngine;
00546     Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00547     Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00548     Markov::Edge<unsigned char>* e = src->Link(target1);
00549     e->AdjustEdge((unsigned long)(ull << 63));
00550     Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00551     Assert::IsTrue(res == target1);
00552     delete src;
00553     delete target1;
00554     delete e;
00555
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
00574 /** @brief randomNext when no edges are present
00575 */
00576 TEST_METHOD(uninitialized_rand_next) {
00577
00578     auto _invalid_next = [] {
00579         Markov::Random::Marsaglia MarsagliaRandomEngine;
00580         Markov::Node<unsigned char>* src = new Markov::Node<unsigned char>('a');
00581         Markov::Node<unsigned char>* target1 = new Markov::Node<unsigned char>('b');
00582         Markov::Edge<unsigned char>* e = new Markov::Edge<unsigned char>(src, target1);
00583         Markov::Node<unsigned char>* res = src->RandomNext(&MarsagliaRandomEngine);
00584
00585         delete src;
00586         delete target1;
00587         delete e;
00588     };
00589
00590     Assert::ExpectException<std::logic_error>(_invalid_next);
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     }

```

```
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

---

\_left  
    Markov::Edge< NodeStorageType >, 117  
\_right  
    Markov::Edge< NodeStorageType >, 118  
\_value  
    Markov::Node< storageType >, 206  
\_weight  
    Markov::Edge< NodeStorageType >, 118

about  
    Markov::GUI::about, 55  
    Markov::GUI::CLI, 63  
    Markov::GUI::menu, 159

about.cpp, 221

about.h, 221, 223

action  
    markopy, 30

AdjustEdge  
    Markov::API::CUDA::CUDAModelMatrix, 75  
    Markov::API::MarkovPasswords, 126  
    Markov::API::ModelMatrix, 177  
    Markov::Edge< NodeStorageType >, 115  
    Markov::Model< NodeStorageType >, 166

AllocVRAMOutputBuffer  
    Markov::API::CUDA::CUDAModelMatrix, 76

alphabet  
    model\_2gram, 38  
    random-model, 39

alternatingKernels  
    Markov::API::CUDA::CUDAModelMatrix, 104

Argparse  
    Markov::API::CLI::Argparse, 57

argparse.cpp, 223

argparse.h, 224, 226  
    BOOST\_ALL\_STATIC\_LIB, 225  
    BOOST\_PROGRAM\_OPTIONS\_STATIC\_LIB, 226

args  
    markopy, 30

benchmarkSelected  
    Markov::GUI::MarkovPasswordsGUI, 141

bExport  
    Markov::API::CLI::\_programOptions, 52

bFailure  
    Markov::API::CLI::\_programOptions, 52

blImport  
    Markov::API::CLI::\_programOptions, 52

BLACK  
    Markov::API::CLI::Terminal, 211

BLUE  
    Markov::API::CLI::Terminal, 211

BOOST\_ALL\_STATIC\_LIB  
    argparse.h, 225  
    markopy.cpp, 263

BOOST\_PROGRAM\_OPTIONS\_STATIC\_LIB  
    argparse.h, 226

BOOST\_PYTHON\_MODULE  
    Markov::Markopy, 37

BOOST\_PYTHON\_STATIC\_LIB  
    markopy.cpp, 263

BROWN  
    Markov::API::CLI::Terminal, 211

Buff  
    Markov::API::CUDA::CUDAModelMatrix, 76  
    Markov::API::MarkovPasswords, 127  
    Markov::API::ModelMatrix, 178

CLI  
    Markov::GUI::CLI, 62

CLI.cpp, 229

CLI.h, 230, 231

cli\_generate  
    markopy, 27

cli\_init  
    markopy, 28

cli\_train  
    markopy, 29

color  
    Markov::API::CLI::Terminal, 211

colormap  
    Markov::API::CLI::Terminal, 211

ConstructMatrix  
    Markov::API::CUDA::CUDAModelMatrix, 78  
    Markov::API::ModelMatrix, 179

cudaBlocks  
    Markov::API::CUDA::CUDAModelMatrix, 104

CudaCheckNotifyErr  
    Markov::API::CUDA::CUDADeviceController, 66  
    Markov::API::CUDA::CUDAModelMatrix, 79  
    Markov::API::CUDA::Random::Marsaglia, 150

cudaDeviceController.cu, 231, 232

cudaDeviceController.h, 233, 234

cudaGridSize  
    Markov::API::CUDA::CUDAModelMatrix, 104

CudaMalloc2DToFlat  
    Markov::API::CUDA::CUDADeviceController, 67  
    Markov::API::CUDA::CUDAModelMatrix, 80  
    Markov::API::CUDA::Random::Marsaglia, 151

CudaMemcpy2DToFlat  
    Markov::API::CUDA::CUDADeviceController, 68  
    Markov::API::CUDA::CUDAModelMatrix, 81  
    Markov::API::CUDA::Random::Marsaglia, 152

cudaMemPerGrid  
    Markov::API::CUDA::CUDAModelMatrix, 104

CudaMigrate2DFlat  
    Markov::API::CUDA::CUDADeviceController, 69  
    Markov::API::CUDA::CUDAModelMatrix, 82  
    Markov::API::CUDA::Random::Marsaglia, 153

cudaModelMatrix.cu, 236, 237  
 cudaModelMatrix.h, 240, 242  
 cudaPerKernelAllocationSize  
   Markov::API::CUDA::CUDAModelMatrix, 104  
 cudarandom.h, 244, 246  
 cudastreams  
   Markov::API::CUDA::CUDAModelMatrix, 104  
 cudaThreads  
   Markov::API::CUDA::CUDAModelMatrix, 104  
 CYAN  
   Markov::API::CLI::Terminal, 211

DARKGRAY  
   Markov::API::CLI::Terminal, 211

datasetFile  
   Markov::API::CUDA::CUDAModelMatrix, 105  
   Markov::API::MarkovPasswords, 138  
   Markov::API::ModelMatrix, 197

datasetname  
   Markov::API::CLI::\_programOptions, 52

DeallocateMatrix  
   Markov::API::CUDA::CUDAModelMatrix, 83  
   Markov::API::ModelMatrix, 181

default  
   markopy, 30

device\_edgeMatrix  
   Markov::API::CUDA::CUDAModelMatrix, 105

device\_matrixIndex  
   Markov::API::CUDA::CUDAModelMatrix, 105

device\_outputBuffer  
   Markov::API::CUDA::CUDAModelMatrix, 105

device\_seeds  
   Markov::API::CUDA::CUDAModelMatrix, 105

device\_totalEdgeWeights  
   Markov::API::CUDA::CUDAModelMatrix, 105

device\_valueMatrix  
   Markov::API::CUDA::CUDAModelMatrix, 105

devrandom  
   Markov::API::CUDA::Random, 35

distribution  
   Markov::API::CUDA::Random::Marsaglia, 154  
   Markov::Random::DefaultRandomEngine, 111  
   Markov::Random::Marsaglia, 145  
   Markov::Random::Mersenne, 163

dllmain.cpp, 246, 247

DumpJSON  
   Markov::API::CUDA::CUDAModelMatrix, 84  
   Markov::API::ModelMatrix, 182

Edge  
   Markov::Edge< NodeStorageType >, 114

edge.h, 248, 249

edgeMatrix  
   Markov::API::CUDA::CUDAModelMatrix, 105  
   Markov::API::ModelMatrix, 198

Edges  
   Markov::API::CUDA::CUDAModelMatrix, 85  
   Markov::API::MarkovPasswords, 128  
   Markov::API::ModelMatrix, 183

Markov::Model< NodeStorageType >, 167  
 Markov::Node< storageType >, 202

edges  
   Markov::API::CUDA::CUDAModelMatrix, 106  
   Markov::API::MarkovPasswords, 138  
   Markov::API::ModelMatrix, 198  
   Markov::Model< NodeStorageType >, 173  
   Markov::Node< storageType >, 206

edgesV  
   Markov::Node< storageType >, 206

EdgeWeight  
   Markov::Edge< NodeStorageType >, 115

endl  
   Markov::API::CLI::Terminal, 212

Export  
   Markov::API::CUDA::CUDAModelMatrix, 85  
   Markov::API::MarkovPasswords, 128, 129  
   Markov::API::ModelMatrix, 183  
   Markov::Model< NodeStorageType >, 167, 168

exportname  
   Markov::API::CLI::\_programOptions, 53

f  
   model\_2gram, 38  
   random-model, 39

FastRandomWalk  
   Markov::API::CUDA::CUDAModelMatrix, 86–88  
   Markov::API::ModelMatrix, 184, 185

FastRandomWalkCUDAKernel  
   Markov::API::CUDA, 33

FastRandomWalkPartition  
   Markov::API::CUDA::CUDAModelMatrix, 89  
   Markov::API::ModelMatrix, 186

FastRandomWalkThread  
   Markov::API::CUDA::CUDAModelMatrix, 90  
   Markov::API::ModelMatrix, 187

FindEdge  
   Markov::Node< storageType >, 202

flatEdgeMatrix  
   Markov::API::CUDA::CUDAModelMatrix, 106

FlattenMatrix  
   Markov::API::CUDA::CUDAModelMatrix, 92

flatValueMatrix  
   Markov::API::CUDA::CUDAModelMatrix, 106

framework.h, 251, 252  
   WIN32\_LEAN\_AND\_MEAN, 251

GatherAsyncKernelOutput  
   Markov::API::CUDA::CUDAModelMatrix, 92

Generate  
   Markov::API::CUDA::CUDAModelMatrix, 92  
   Markov::API::MarkovPasswords, 129  
   Markov::API::ModelMatrix, 188  
   Markov::GUI::Generate, 119

Generate.cpp, 252

Generate.h, 254, 255

generateN  
   Markov::API::CLI::\_programOptions, 53

GenerateThread

Markov::API::CUDA::CUDAModelMatrix, 94  
Markov::API::MarkovPasswords, 130  
Markov::API::ModelMatrix, 190  
generation  
    Markov::GUI::Generate, 120  
generator  
    Markov::API::CUDA::Random::Marsaglia, 154  
    Markov::Random::DefaultRandomEngine, 111  
    Markov::Random::Marsaglia, 146  
    Markov::Random::Mersenne, 163  
getProgramOptions  
    Markov::API::CLI::Argparse, 59  
GREEN  
    Markov::API::CLI::Terminal, 211  
  
help  
    markopy, 30  
    Markov::API::CLI::Argparse, 59  
home  
    Markov::GUI::Generate, 121  
    Markov::GUI::MarkovPasswordsGUI, 141  
    Markov::GUI::Train, 218  
  
Import  
    Markov::API::CUDA::CUDAModelMatrix, 95, 96  
    Markov::API::MarkovPasswords, 131, 132  
    Markov::API::ModelMatrix, 191, 192  
    Markov::Model< NodeStorageType >, 169  
importname  
    Markov::API::CLI::\_\_programOptions, 53  
intHandler  
    markovPasswords.cpp, 267  
iterationsPerKernelThread  
    Markov::API::CUDA::CUDAModelMatrix, 106  
  
keepRunning  
    markovPasswords.cpp, 268  
  
LaunchAsyncKernel  
    Markov::API::CUDA::CUDAModelMatrix, 97  
LeftNode  
    Markov::Edge< NodeStorageType >, 116  
LIGHTGRAY  
    Markov::API::CLI::Terminal, 211  
Link  
    Markov::Node< storageType >, 203  
ListCudaDevices  
    Markov::API::CUDA::CUDADeviceController, 70  
    Markov::API::CUDA::CUDAModelMatrix, 97  
    Markov::API::CUDA::Random::Marsaglia, 155  
listfile  
    Markov::API::Concurrency::ThreadSharedListHandler, 215  
  
MAGENTA  
    Markov::API::CLI::Terminal, 211  
main  
    main.cu, 260  
    MarkovAPICLI/src/main.cpp, 256  
                MarkovPasswordsGUI/src/main.cpp, 259  
    main.cpp, 256–259  
    main.cu, 260, 261  
        main, 260  
    markopy, 27  
        action, 30  
        args, 30  
        cli\_generate, 27  
        cli\_init, 28  
        cli\_train, 29  
        default, 30  
        help, 30  
        parser, 31  
    markopy.cpp, 261, 263  
        BOOST\_ALL\_STATIC\_LIB, 263  
        BOOST\_PYTHON\_STATIC\_LIB, 263  
    markopy.py, 264  
Markov, 31  
Markov::API, 31  
Markov::API::CLI, 32  
    operator<<, 32  
    ProgramOptions, 32  
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, 55  
    Argparse, 57  
    getProgramOptions, 59  
    help, 59  
    parse, 59  
    po, 61  
    setProgramOptions, 60  
Markov::API::CLI::Terminal, 209  
    BLACK, 211  
    BLUE, 211  
    BROWN, 211  
    color, 211  
    colormap, 211  
    CYAN, 211  
    DARKGRAY, 211  
    endl, 212  
    GREEN, 211  
    LIGHTGRAY, 211  
    MAGENTA, 211  
    RED, 211  
    RESET, 211  
    Terminal, 211  
    WHITE, 211  
    YELLOW, 211  
    Markov::API::Concurrency, 33

Markov::API::Concurrency::ThreadSharedListHandler,  
     212  
     listfile, 215  
     mlock, 215  
     next, 215  
     ThreadSharedListHandler, 214  
 Markov::API::CUDA, 33  
     FastRandomWalkCUDAKernel, 33  
     strchr, 35  
 Markov::API::CUDA::CUDADeviceController, 64  
     CudaCheckNotifyErr, 66  
     CudaMalloc2DToFlat, 67  
     CudaMemcpy2DToFlat, 68  
     CudaMigrate2DFlat, 69  
     ListCudaDevices, 70  
 Markov::API::CUDA::CUDAModelMatrix, 71  
     AdjustEdge, 75  
     AllocVRAMOutputBuffer, 76  
     alternatingKernels, 104  
     Buff, 76  
     ConstructMatrix, 78  
     cudaBlocks, 104  
     CudaCheckNotifyErr, 79  
     cudaGridSize, 104  
     CudaMalloc2DToFlat, 80  
     CudaMemcpy2DToFlat, 81  
     cudaMemPerGrid, 104  
     CudaMigrate2DFlat, 82  
     cudaPerKernelAllocationSize, 104  
     cudastreams, 104  
     cudaThreads, 104  
     datasetFile, 105  
     DeallocateMatrix, 83  
     device\_edgeMatrix, 105  
     device\_matrixIndex, 105  
     device\_outputBuffer, 105  
     device\_seeds, 105  
     device\_totalEdgeWeights, 105  
     device\_valueMatrix, 105  
     DumpJSON, 84  
     edgeMatrix, 105  
     Edges, 85  
     edges, 106  
     Export, 85  
     FastRandomWalk, 86–88  
     FastRandomWalkPartition, 89  
     FastRandomWalkThread, 90  
     flatEdgeMatrix, 106  
     FlattenMatrix, 92  
     flatValueMatrix, 106  
     GatherAsyncKernelOutput, 92  
     Generate, 92  
     GenerateThread, 94  
     Import, 95, 96  
     iterationsPerKernelThread, 106  
     LaunchAsyncKernel, 97  
     ListCudaDevices, 97  
     matrixIndex, 106  
     matrixSize, 106  
     MigrateMatrix, 97  
     modelSavefile, 106  
     Nodes, 98  
     nodes, 106  
     numberOfPartitions, 107  
     OpenDatasetFile, 98  
     OptimizeEdgeOrder, 99  
     outputBuffer, 107  
     outputFile, 107  
     prepKernelMemoryChannel, 99  
     RandomWalk, 100  
     ready, 107  
     Save, 101  
     StarterNode, 101  
     starterNode, 107  
     totalEdgeWeights, 107  
     totalOutputPerKernel, 107  
     totalOutputPerSync, 108  
     Train, 102  
     TrainThread, 103  
     valueMatrix, 108  
 Markov::API::CUDA::Random, 35  
     devrandom, 35  
 Markov::API::CUDA::Random::Marsaglia, 148  
     CudaCheckNotifyErr, 150  
     CudaMalloc2DToFlat, 151  
     CudaMemcpy2DToFlat, 152  
     CudaMigrate2DFlat, 153  
     distribution, 154  
     generator, 154  
     ListCudaDevices, 155  
     MigrateToVRAM, 155  
     random, 156  
     rd, 157  
     x, 157  
     y, 157  
     z, 157  
 Markov::API::MarkovPasswords, 122  
     AdjustEdge, 126  
     Buff, 127  
     datasetFile, 138  
     Edges, 128  
     edges, 138  
     Export, 128, 129  
     Generate, 129  
     GenerateThread, 130  
     Import, 131, 132  
     MarkovPasswords, 126  
     modelSavefile, 138  
     Nodes, 133  
     nodes, 138  
     OpenDatasetFile, 133  
     OptimizeEdgeOrder, 133  
     outputFile, 138  
     RandomWalk, 134  
     Save, 135  
     StarterNode, 135

starterNode, 138  
Train, 135  
TrainThread, 137  
Markov::API::ModelMatrix, 173  
  AdjustEdge, 177  
  Buff, 178  
  ConstructMatrix, 179  
  datasetFile, 197  
  DeallocateMatrix, 181  
  DumpJSON, 182  
  edgeMatrix, 198  
  Edges, 183  
  edges, 198  
  Export, 183  
  FastRandomWalk, 184, 185  
  FastRandomWalkPartition, 186  
  FastRandomWalkThread, 187  
  Generate, 188  
  GenerateThread, 190  
  Import, 191, 192  
  matrixIndex, 198  
  matrixSize, 198  
  ModelMatrix, 177  
  modelSavefile, 198  
  Nodes, 193  
  nodes, 198  
  OpenDatasetFile, 193  
  OptimizeEdgeOrder, 193  
  outputFile, 198  
  RandomWalk, 194  
  ready, 198  
  Save, 195  
  StarterNode, 195  
  starterNode, 199  
  totalEdgeWeights, 199  
  Train, 196  
  TrainThread, 197  
  valueMatrix, 199  
Markov::Edge< NodeStorageType >, 113  
  \_left, 117  
  \_right, 118  
  \_weight, 118  
  AdjustEdge, 115  
  Edge, 114  
  EdgeWeight, 115  
  LeftNode, 116  
  RightNode, 116  
  SetLeftEdge, 116  
  SetRightEdge, 117  
  TraverseNode, 117  
Markov::GUI, 36  
Markov::GUI::about, 54  
  about, 55  
  ui, 55  
Markov::GUI::CLI, 61  
  about, 63  
  CLI, 62  
  start, 63  
          statistics, 64  
          ui, 64  
Markov::GUI::Generate, 118  
  Generate, 119  
  generation, 120  
  home, 121  
  train, 121  
  ui, 122  
  vis, 121  
Markov::GUI::MarkovPasswordsGUI, 139  
  benchmarkSelected, 141  
  home, 141  
  MarkovPasswordsGUI, 140  
  model, 141  
  pass, 141  
  ui, 141  
Markov::GUI::menu, 158  
  about, 159  
  menu, 159  
  ui, 160  
  visualization, 159  
Markov::GUI::Train, 216  
  home, 218  
  Train, 217  
  train, 218  
  ui, 219  
Markov::Markopy, 36  
  BOOST\_PYTHON\_MODULE, 37  
Markov::Model< NodeStorageType >, 165  
  AdjustEdge, 166  
  Edges, 167  
  edges, 173  
  Export, 167, 168  
  Import, 169  
  Model, 166  
  Nodes, 170  
  nodes, 173  
  OptimizeEdgeOrder, 171  
  RandomWalk, 171  
  StarterNode, 172  
  starterNode, 173  
Markov::Node< storageType >, 199  
  \_value, 206  
  Edges, 202  
  edges, 206  
  edgesV, 206  
  FindEdge, 202  
  Link, 203  
  Node, 201  
  nodeValue, 204  
  RandomNext, 204  
  total\_edge\_weights, 206  
  TotalEdgeWeights, 205  
  UpdateEdges, 205  
  UpdateTotalVertexWeight, 206  
Markov::Random, 37  
Markov::Random::DefaultRandomEngine, 108  
  distribution, 111

generator, 111  
 random, 112  
 rd, 112  
**Markov::Random::Marsaglia**, 142  
 distribution, 145  
 generator, 146  
 Marsaglia, 145  
 random, 146  
 rd, 147  
 x, 147  
 y, 147  
 z, 147  
**Markov::Random::Mersenne**, 160  
 distribution, 163  
 generator, 163  
 random, 164  
 rd, 164  
**Markov::Random::RandomEngine**, 207  
 random, 209  
**MarkovAPICLI/src/main.cpp**  
 main, 256  
**MarkovPasswords**  
 Markov::API::MarkovPasswords, 126  
**markovPasswords.cpp**, 266, 268  
 intHandler, 267  
 keepRunning, 268  
**markovPasswords.h**, 270, 271  
**MarkovPasswordsGUI**  
 Markov::GUI::MarkovPasswordsGUI, 140  
**MarkovPasswordsGUI.cpp**, 273  
**MarkovPasswordsGUI.h**, 274, 275  
**MarkovPasswordsGUI/src/main.cpp**  
 main, 259  
**Marsaglia**  
 Markov::Random::Marsaglia, 145  
**matrixIndex**  
 Markov::API::CUDA::CUDAModelMatrix, 106  
 Markov::API::ModelMatrix, 198  
**matrixSize**  
 Markov::API::CUDA::CUDAModelMatrix, 106  
 Markov::API::ModelMatrix, 198  
**menu**  
 Markov::GUI::menu, 159  
**menu.cpp**, 276  
**menu.h**, 277, 278  
**MigrateMatrix**  
 Markov::API::CUDA::CUDAModelMatrix, 97  
**MigrateToVRAM**  
 Markov::API::CUDA::Random::Marsaglia, 155  
**mlock**  
 Markov::API::Concurrency::ThreadSharedListHandler  
 215  
**Model**  
 Markov::Model< NodeStorageType >, 166  
**model**  
 Markov::GUI::MarkovPasswordsGUI, 141  
**model.h**, 278, 279  
**model\_2gram**, 38  
 alphabet, 38  
 f, 38  
**model\_2gram.py**, 284  
**ModelMatrix**  
 Markov::API::ModelMatrix, 177  
**modelMatrix.cpp**, 284, 285  
**modelMatrix.h**, 288, 289  
**modelSavefile**  
 Markov::API::CUDA::CUDAModelMatrix, 106  
 Markov::API::MarkovPasswords, 138  
 Markov::API::ModelMatrix, 198  
**next**  
 Markov::API::Concurrency::ThreadSharedListHandler, 215  
**Node**  
 Markov::Node< storageType >, 201  
**node.h**, 292, 293  
**Nodes**  
 Markov::API::CUDA::CUDAModelMatrix, 98  
 Markov::API::MarkovPasswords, 133  
 Markov::API::ModelMatrix, 193  
 Markov::Model< NodeStorageType >, 170  
**nodes**  
 Markov::API::CUDA::CUDAModelMatrix, 106  
 Markov::API::MarkovPasswords, 138  
 Markov::API::ModelMatrix, 198  
 Markov::Model< NodeStorageType >, 173  
**nodeValue**  
 Markov::Node< storageType >, 204  
**numberOfPartitions**  
 Markov::API::CUDA::CUDAModelMatrix, 107  
**OpenDatasetFile**  
 Markov::API::CUDA::CUDAModelMatrix, 98  
 Markov::API::MarkovPasswords, 133  
 Markov::API::ModelMatrix, 193  
**operator<<**  
 Markov::API::CLI, 32  
 term.cpp, 306  
**OptimizeEdgeOrder**  
 Markov::API::CUDA::CUDAModelMatrix, 99  
 Markov::API::MarkovPasswords, 133  
 Markov::API::ModelMatrix, 193  
 Markov::Model< NodeStorageType >, 171  
**outputBuffer**  
 Markov::API::CUDA::CUDAModelMatrix, 107  
**outputFile**  
 Markov::API::CUDA::CUDAModelMatrix, 107  
 Markov::API::MarkovPasswords, 138  
 Markov::API::ModelMatrix, 198  
**outputfilename**  
 Markov::CLI::\_\_programOptions, 53  
**parse**  
 Markov::API::CLI::Argparse, 59  
**parser**  
 markopy, 31  
**pass**

Markov::GUI::MarkovPasswordsGUI, 141  
pch.cpp, 297, 298  
pch.h, 298–300  
po  
    Markov::API::CLI::Argparse, 61  
prepKernelMemoryChannel  
    Markov::API::CUDA::CUDAModelMatrix, 99  
ProgramOptions  
    Markov::API::CLI, 32  
QMainWindow, 207  
random, 38  
    Markov::API::CUDA::Random::Marsaglia, 156  
    Markov::Random::DefaultRandomEngine, 112  
    Markov::Random::Marsaglia, 146  
    Markov::Random::Mersenne, 164  
    Markov::Random::RandomEngine, 209  
random-model, 38  
    alphabet, 39  
    f, 39  
random-model.py, 300, 301  
random.h, 301, 303  
RandomNext  
    Markov::Node<storageType>, 204  
RandomWalk  
    Markov::API::CUDA::CUDAModelMatrix, 100  
    Markov::API::MarkovPasswords, 134  
    Markov::API::ModelMatrix, 194  
    Markov::Model<NodeStorageType>, 171  
rd  
    Markov::API::CUDA::Random::Marsaglia, 157  
    Markov::Random::DefaultRandomEngine, 112  
    Markov::Random::Marsaglia, 147  
    Markov::Random::Mersenne, 164  
README.md, 305  
ready  
    Markov::API::CUDA::CUDAModelMatrix, 107  
    Markov::API::ModelMatrix, 198  
RED  
    Markov::API::CLI::Terminal, 211  
report.md, 305  
RESET  
    Markov::API::CLI::Terminal, 211  
RightNode  
    Markov::Edge<NodeStorageType>, 116  
Save  
    Markov::API::CUDA::CUDAModelMatrix, 101  
    Markov::API::MarkovPasswords, 135  
    Markov::API::ModelMatrix, 195  
seperator  
    Markov::API::CLI::\_programOptions, 53  
SetLeftEdge  
    Markov::Edge<NodeStorageType>, 116  
setProgramOptions  
    Markov::API::CLI::Argparse, 60  
SetRightEdge  
    Markov::Edge<NodeStorageType>, 117  
start  
    Markov::GUI::CLI, 63  
StarterNode  
    Markov::API::CUDA::CUDAModelMatrix, 101  
    Markov::API::MarkovPasswords, 135  
    Markov::API::ModelMatrix, 195  
    Markov::Model<NodeStorageType>, 172  
starterNode  
    Markov::API::CUDA::CUDAModelMatrix, 107  
    Markov::API::MarkovPasswords, 138  
    Markov::API::ModelMatrix, 199  
    Markov::Model<NodeStorageType>, 173  
statistics  
    Markov::GUI::CLI, 64  
strchr  
    Markov::API::CUDA, 35  
term.cpp, 305, 306  
    operator<<, 306  
term.h, 307, 309  
    TERM\_FAIL, 309  
    TERM\_INFO, 309  
    TERM\_SUCC, 309  
    TERM\_WARN, 309  
TERM\_FAIL  
    term.h, 309  
TERM\_INFO  
    term.h, 309  
TERM\_SUCC  
    term.h, 309  
TERM\_WARN  
    term.h, 309  
Terminal  
    Markov::API::CLI::Terminal, 211  
TEST\_CLASS  
    Testing::MarkovModel, 39–41  
    Testing::MVP::MarkovModel, 43–45  
    Testing::MVP::MarkovPasswords, 49  
Testing, 39  
Testing::MarkovModel, 39  
    TEST\_CLASS, 39–41  
Testing::MarkovPasswords, 42  
Testing::MVP, 43  
Testing::MVP::MarkovModel, 43  
    TEST\_CLASS, 43–45  
Testing::MVP::MarkovPasswords, 48  
    TEST\_CLASS, 49  
ThreadSharedListHandler  
    Markov::Concurrency::ThreadSharedListHandler, 214  
threadSharedListHandler.cpp, 310, 311  
threadSharedListHandler.h, 311, 312  
total\_edge\_weights  
    Markov::Node<storageType>, 206  
TotalEdgeWeights  
    Markov::Node<storageType>, 205  
totalEdgeWeights  
    Markov::API::CUDA::CUDAModelMatrix, 107  
    Markov::API::ModelMatrix, 199

totalOutputPerKernel  
    Markov::API::CUDA::CUDAModelMatrix, 107

totalOutputPerSync  
    Markov::API::CUDA::CUDAModelMatrix, 108

Train  
    Markov::API::CUDA::CUDAModelMatrix, 102  
    Markov::API::MarkovPasswords, 135  
    Markov::API::ModelMatrix, 196  
    Markov::GUI::Train, 217

train  
    Markov::GUI::Generate, 121  
    Markov::GUI::Train, 218

Train.cpp, 314

Train.h, 315, 317

TrainThread  
    Markov::API::CUDA::CUDAModelMatrix, 103  
    Markov::API::MarkovPasswords, 137  
    Markov::API::ModelMatrix, 197

TraverseNode  
    Markov::Edge< NodeStorageType >, 117

ui  
    Markov::GUI::about, 55  
    Markov::GUI::CLI, 64  
    Markov::GUI::Generate, 122  
    Markov::GUI::MarkovPasswordsGUI, 141  
    Markov::GUI::menu, 160  
    Markov::GUI::Train, 219

UnitTests.cpp, 317, 318

UpdateEdges  
    Markov::Node< storageType >, 205

UpdateTotalVerticeWeight  
    Markov::Node< storageType >, 206

valueMatrix  
    Markov::API::CUDA::CUDAModelMatrix, 108  
    Markov::API::ModelMatrix, 199

vis  
    Markov::GUI::Generate, 121

visualization  
    Markov::GUI::menu, 159

WHITE  
    Markov::API::CLI::Terminal, 211

WIN32\_LEAN\_AND\_MEAN  
    framework.h, 251

wordlistname  
    Markov::API::CLI::\_programOptions, 53

x  
    Markov::API::CUDA::Random::Marsaglia, 157  
    Markov::Random::Marsaglia, 147

y  
    Markov::API::CUDA::Random::Marsaglia, 157  
    Markov::Random::Marsaglia, 147

YELLOW  
    Markov::API::CLI::Terminal, 211

z